Neural Comput & Applic (2008) 17:347–355
DOI 10.1007/s00521-007-0127-5

ORIGINAL ARTICLE

Neurodynamic programming: a case study of the traveling salesman problem

Jia Ma · Tao Yang · Zeng-Guang Hou · Min Tan · Derong Liu

Received: 31 January 2007 / Accepted: 30 April 2007 / Published online: 31 May 2007

Abstract The paper focuses on the study of solving the large-scale traveling salesman problem (TSP) based on neurodynamic programming. From this perspective, two methods, temporal difference learning and approximate Sarsa, are presented in detail. In essence, both of them try to learn an appropriate evaluation function on the basis of a finite amount of experience. To evaluate their performances, some computational experiments on both the Euclidean and asymmetric TSP instances are conducted. In contrast with the large size of the state space, only a few training sets have been used to obtain the initial results. Hence, the results are acceptable and encouraging in comparisons with some classical algorithms, and further study of this kind of methods, as well as applications in combinatorial optimization problems, is worth investigating.

Keywords Neurodynamic programming · Temporal difference learning · Approximate Sarsa · Traveling salesman problem

J. Ma · T. Yang · Z.-G. Hou (✉) · M. Tan
Key Laboratory of Complex Systems and Intelligence Science, Institute of Automation, Chinese Academy of Sciences, Beijing 100080, China
e-mail: zengguang.hou@ia.ac.cn

J. Ma
e-mail: jia.ma@ia.ac.cn

T. Yang
e-mail: yangtao@ia.ac.cn

M. Tan
e-mail: min.tan@ia.ac.cn

D. Liu
Department of Electrical and Computer Engineering, University of Illinois, Chicago, IL 60607, USA
e-mail: dliu@ece.uic.edu

1 Introduction

The traveling salesman problem (TSP) is a hard combinatorial optimization problem. It is easy to state [6]: given a set of cities and a cost matrix (representing distances, travel times, etc., between each pair of them), it is required to determine the least-cost tour passing once and only once through each city. The simplicity of this problem is coupled with its apparent intractability. The number of solutions becomes extremely large for a large number of cities. For instance, even with a symmetric cost matrix, there are $(n-1)!/2$ different tours for a n-city TSP. Therefore, an exhaustive search for the optimal solution cannot be finished in a polynomial time.

As a kind of NP-complete problems, the TSP is made as an ideal platform for exploring new algorithmic ideas due to its simple formulation. Various algorithms have been put forward to solve the TSP. In general, they can be classified as exact algorithms and heuristic algorithms. As for exact methods, the computing time requirements increase either exponentially, or according to very high-order monomial functions [30], so the large-scale TSP cannot be solved using them. For this reason, many heuristic or approximate algorithms have been developed to produce answers which are near optimal but can be obtained within an appropriate time.

Generally speaking, classical heuristic algorithms can be divided into tour construction heuristics, tour improvement heuristics, and composite heuristics [10, 22, 30].

1.1 Tour construction heuristics

Tour constructive heuristics usually start selecting a random city from the set of cities and then incrementally build a feasible TSP solution by adding new cities chosen according to some heuristic rules. The most used and well

known tour constructive heuristics are the nearest neighbor heuristic, the greedy heuristic, the Clarke–Wright heuristic [7], and the Christofides heuristic [5].

1.2 Tour improvement heuristics

Local improvement heuristic start from a given tour and repeatedly modify the current tour by exchanging edges so as to generate a new improved solution until a local optimum is found, namely, until no further improvement is possible via the heuristic. Among local improvement heuristics, the most famous are the 2-Opt [9], 3-Opt [26], and Lin–Kernighan heuristics [21, 27]. In order to overcome the limitations associated with local optimization algorithms, global improvement heuristics such as simulated annealing (SA) [25], generic algorithm(GA) [18] and tabu search [12] are applied. This kind of heuristics provide some mechanism helpful to escape from local optima and continue the search further.

1.3 Composite heuristics

Composite heuristics make use of both construction and improvement techniques. The most successful ones among them are the CCAO heuristic [16], the genius heuristic [15], and the iterated Lin–Kernighan heuristic [21].

Most of the TSP algorithms so far based on artificial neural networks (ANN) cannot be competitive with classical heuristic algorithms. However, this technology is quite potential and great improvements have already been achieved. The potential in tackling the TSP using ANN is due to two aspects. On one hand, its learning capability is very spectacular. The other lies in the fact that many neural network models can be implemented naturally on parallel computers. Hence, the ANN technology could solve optimization problems at a speed that has never been achieved before.

The first application of a neural network approach to the TSP was due to Hopfield and Tank [19]. The Hopfield–Tank model basically performs a gradient descent search to find a local minimum of an appropriate energy function [1]. Hence, it is expected that the local minimum corresponds to a good solution of the TSP being solved. Besides the Hopfield Tank model, the elastic net approach and the self-organizing map approach for the TSP have been reported. The elastic net approach [11] is fundamentally different from the approach of Hopfield and Tank. It is a geometrically inspired algorithm, originating in the work of Willshaw and Von der Malsburg [37]. The self-organizing map approach is a variant on the elastic net approach. It is inspired by the competitive neural networks of Kohonen. A more detailed survey of neural network algorithms can be found in [30].

Neural network approaches to the TSP mentioned above mainly use unsupervised learning, which belongs to the paradigm of learning without a teacher. This paradigm also includes reinforcement learning. It is a ‘‘behavioral’’ learning problem [17]. Specifically, in this problem, the agent must learn behavior through trial-and-error interactions with an environment [23]. This kind of learning has attracted much interest in the machine learning community in recent years. The classical approaches to the study of RL is to learn a highly skilled behavior through a process of punishment and reward [17]. Some methods based on these approaches have been used to solve the TSP. For example, Dorigo [10] has developed an optimization technique known as ant colony system (ACS) for solving the TSP, which was based on the metaphor of ant colonies. Gambardella and Dorigo [14] proposed an Ant-Q algorithm inspired by the work on the ACS, which presented many similarities with Q-learning. Miagkikh and Punch [28] used a multi-agent RL algorithm which combined local and global search characteristics to solve combinatorial optimization problems.

This paper tries to solve the TSP from the perspective of neurodynamic programming. In contrast with the classical one mentioned above, neurodynamic programming is a modern approach to the study of RL, which builds on dynamic programming. Using this approach, an agent learns to make good decisions by observing its own behaviors and considering possible future stages without actually experiencing them [17]. The approach has been successfully applied to solve difficult large-scale problems in many diverse fields [17], which include backgammon [35], elevator dispatching [8], and dynamic channel allocation [32]. The successful applications mentioned above using this approach can be contributed to its emphasis on planning and its learning capability provided by neural networks. Hence, neurodynamic programming has great potential in solving the large-scale TSP.

The rest of this paper is organized as follows. Sections 2 and 3, respectively present the TD learning method and the approximate Sarsa method for solving the TSP, both of which adopt neurodynamic programming techniques. Section 4 describes some computational experiments that are carried out to evaluate the two methods and to compare them with the classical algorithms to the TSP. Finally, Sect. 5 discusses the results of the experiments and concludes the paper by summarizing the difficulties and limitations in solving the TSP by using the neurodynamic programming techniques.

2 Temporal differences learning

In the case of the large-scale TSP, two difficult problems involved in RL must be solved. The first problem is the

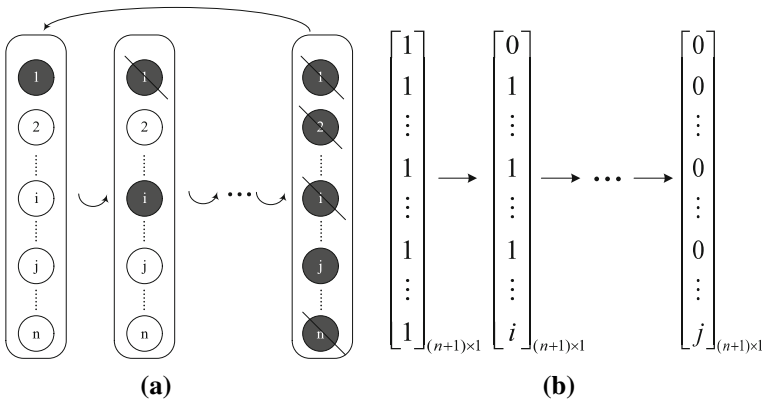

Fig. 1 Illustration of state representations of a certain tour

temporal credit assignment problem. Suppose a learning agent performs a sequence of actions and obtains a total cumulative cost over the sequence. For the purpose of obtaining an optimal policy, it must figure out how to assign credit or blame to each of a set of interacting decisions to capture an efficient tradeoff between immediate and future cost. The second problem is the generalization problem. Faced with a enormous size of the state space, it is costly or impossible for the agent to explore completely. Therefore, in a new situation, the agent must have the ability to predict the cumulative cost over the remaining sequence of the travel based on past experience.

The temporal differences (TD) method is the most popular approach to the first problem, which was firstly proposed by Sutton [33]. This method is similarly driven by the error between temporally successive predictions instead of the error between predicted and actual outcomes. On the other hand, a multilayer perceptron (MLP) trained with the back-propagation algorithm is very suitable for solving the generalization problem. Combining the two methods mentioned above, an approach, named as TD-Gammon, has been successfully applied in playing backgammon game [35]. This TD-Gammon programm can play at a grandmaster level. The proposed approach to the TSP in this paper is closely related to it. In what follows, more details will be given.

The TSP discussed later can be defined as follows.

Let $S = s_1,...,s_n$ be a set of cities, $G = \{(s_1,s_2):s_1,s_2 \in S\}$ be the edge set, and cost(s_1,s_2) be a cost measure associated with the edge $(s_1,s_2) \in G$. The TSP is the problem of finding a closed tour with minimal cost that each city is visited once.

In the case cities s_i, $s_j \in S$ are given by their coordinates (X_{s_i}, Y_{s_i}); (X_{s_j}, Y_{s_j}) and cost(s_i,s_j) is the Euclidean distance between s_i and s_j, the problem is the Euclidean TSP (ETSP), where cost(s_1,s_2) = cost(s_2,s_1). If cost(s_1,s_2) $\neq$ cost(s_2,s_1), then the ETSP becomes the asymmetric TSP (ATSP).

Some terms are defined in the following for later use.

Let n be the number of all cities $\in S$.

Let StateHist (S_r) be a list which memorizes the sequence of cities already visited, where S_r stands for the current city. This list is used to constrain the leaning agent to make action selection so that all cities would be visited once and only once.

Let $x(t)$ be the state of the TSP at step t in the tour. In order to fully represent the state, it would contain both the current city, and the cities to be visited in the future. Hence, in this paper, $x(t)$ uses a column vector with $n + 1$ dimensions to encode the agent's state. The values of the first n elements correspond to particular cities $\in S$ respectively. Specifically, 0 denotes that the city has been visited; 1 denotes the city is the current one or the one to be visited. The last element uses the sequence number of the current city to stand for the current position of the agent. Figure 1 illustrates state representations of a certain tour. In Fig. 1a, the black circle stands for the current city. The black one deleted with a dash stands for the city visited before. The white one stands for the city to be visited. Then we can get the representations of these states shown in Fig. 1b. This kind of representation is used as input representation for the evaluation network Cnet described later.

The cost-to-go function for a policy π starting from state $x(t)$ is defined by

$$J^{\pi}(t) = \text{cost}(t) + J^{\pi}(t+1) \tag{1}$$

where cost(t) is the cost incurred at the transition from state $x(t)$ to state $x(t + 1)$, and the policy π refers to a random selection among the feasible cities $s \in$ StateHist (Sr) at state $x(t)$ in this section. It need to be pointed that discount factor is not considered because the TSP is a finite-horizon problem.

The evaluation network Cnet is depicted in Fig. 2, which aims to provide $J(t)$ as an approximate of $J^{\pi}(t)$ in Eq. (1). The prediction error for Cnet is defined as

$$e(t) = [J(t+1) + \text{cost}(t)] - J(t) \tag{2}$$

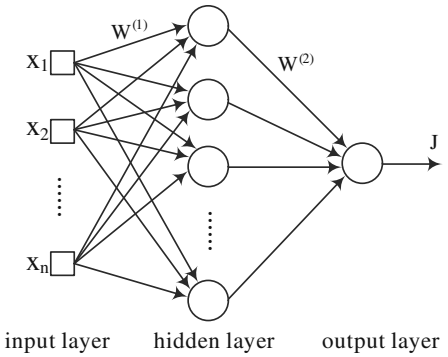

Fig. 2 Illustration of the implementation of a nonlinear evaluation network Cnet using a feedword network with one hidden layer

In the evaluation network Cnet, the output $J(t)$ will be of the form

$$J(t) = \sum_{i=1}^{m} w_i^{(2)}(t) v_i(t) \tag{3}$$

$$v_i(t) = \varphi(u_i(t)) = \frac{1}{1 + exp(-u_i(t))} \quad i = 1, \ldots, m \tag{4}$$

$$u_i(t) = \sum_{j=1}^{n} w_{ij}^{(1)}(t) x_j(t) \quad i = 1, \ldots, m \tag{5}$$

where u_i is the input of the ith hidden node, v_i corresponds to the output of the hidden node, and m is the total number of hidden nodes in the evaluation network Cnet.

Weights $W^{(2)}$ from the hidden layer to the output layer and weights $W^{(1)}$ from the input layer to the hidden layer are updated according to the chain rules described in detail as follows.

1. $\Delta W^{(2)}$ (hidden to output layer)

$$\Delta w_i^{(2)}(t) = \alpha(t) e(t) \sum_{k=1}^{t} \lambda^{t-k} \frac{\partial J(k)}{\partial w_i^{(2)}(k)} \tag{6}$$

$$\frac{\partial J(t)}{\partial w_i^{(2)}(t)} = v_i(t) \tag{7}$$

2. $\Delta W^{(1)}$ (input to hidden layer)

$$\Delta w_{ij}^{(1)}(t) = \alpha(t) e(t) \sum_{k=1}^{t} \lambda^{t-k} \frac{\partial J(k)}{\partial w_{ij}^{(1)}(k)} \tag{8}$$

$$\begin{aligned} \frac{\partial J(t)}{\partial w_{ij}^{(2)}(t)} &= \frac{\partial J(t)}{\partial v_i(t)} \frac{\partial v_i(t)}{\partial u_i(t)} \frac{\partial u_i(t)}{\partial w_{ij}^{(1)}(t)} \\ &= w_i^{(2)}(t) v_i(t)[1 - v_i(t)] x_j(t) \end{aligned} \tag{9}$$

where $\alpha(t)$ is the learning-rate parameter of the back-propagation algorithm of the evaluation network Cnet, which usually decreases gradually to a small value. λ is a recency factor. The TD error for adjusting the network weights can be determined by the discrepancy between not only two but multiple consecutive predictions if $\lambda > 0$.

The total learning procedure of the evaluation network Cnet can be summarized as follows.

1. Initialize the evaluation network Cnet arbitrarily, which is used to produce $J(t)$.
2. For each episode = 1,2,... (where a episode refers to a complete tour from the start city back to the beginning), do the following.

(a) Initialize state $x(1)$.
(b) For each remaining step i to the destination, do what follows.

i. Take action a using random policy π, observe cost(i), $x(i + 1)$.
ii. Adjust the evaluation network by backpropagating the TD error (cost(i) + $J(i + 1)$–$J(i)$) through it with input $x(i)$.
iii. i $\leftarrow$ i+1.
iv. Go to (b).

3. Go to 2.

3 Approximate Sarsa

The standard Sarsa method [34] is a kind of on-policy TD control method, which in essence is a stochastic value iteration algorithm under the situation lack of the prior knowledge of the transition probabilities. A kind of approximate Sarsa method is applied to solving the TSP.

A concept, Q values, is introduced in the form of $Q^{\pi}(s,a)$ due to Watkins and Dayan, which refers to the expected discounted reward or cost for executing action a at state s and following policy π thereafter [36]. The object in the Sarsa method is to estimate $Q^*(s,a)$ for an optimal policy. This method is very similar to the famous Q-learning. One major difference between them is the latter computes Q values based on the action selected off-policy.

The evaluation network Qnet to approximate $Q^*(s,a)$ used in this section is a MLP trained with the back-propagation algorithm based on the standard Sarsa method. The prediction error at state s for Qnet is defined as

$$e(s) = [Q^{e}(s', a') + \text{cost}(s, a)] - Q^{e}(s, a) \tag{10}$$

Table 1 Comparison of the TD(0) and approximate Sarsa with the Hopfield–Tank algorithm on a 10-city random instance of the ETSP

Index	Hopfield	Sarsa	TD(0)
Best	5.4965	5.4518	5.4356
Average	7.4113	5.6132	5.8355
MSE	0.5331	0.0524	0.1073

where s and s' are the states of the TSP at step t and step $t+1$ respectively, a and a' are the actions taken at their corresponding states, ε refers to ε-greedy policy, cost(s,a) is the immediate cost by taking action a at state s.

As for the TSP, the input representation for Cnet described in Sect. 2 contains the information of action selected as well. So the same input representation can be used for Qnet.

The learning algorithm of the evaluation network Qnet can be summarize as follows.

1. Initialize the evaluation network Qnet arbitrarily, which is used to produce $Q^{\varepsilon}(s,a)$.
2. For each episode = 1,2,... (where a episode refers to a complete tour from the start city back to the beginning), do the following.

(a) Initialize state s, choose action a using ε-greedy policy derived from $Q^{\varepsilon}(s,a)$.
(b) For each remaining step i to the destination, do what follows.

i. Take action a, observe cost(s,a), s'.
ii. Choose action a' using ε-greedy policy derived from $Q^{\varepsilon}(s',a')$.
iii. Error signal $e(s) \leftarrow Q^{\varepsilon}(s',a') + \text{cost}(s,a) - Q^{\varepsilon}(s,a)$.
iv. Adjust the evaluation network Qnet by backpropagating error signal through it with input (s,a).
v. $s \leftarrow s'$; $a \leftarrow a'$.
vi. Go to (b).

3. Go to 2.

4 Some computational results

To evaluate the TD and approximate Sarsa methods for solving the TSP, the ETSP and ATSP are taken into account. As for the ETSP, two test sets are considered. The first set is a randomly generated 10-city problem, while the second set is composed of three geometric problems, which are taken from the real world. It is important to run experiments on both sets of instances because difference in structures of the instances makes them difficult for a particular algorithm and at the same time easy for another one [10]. In addition, three test instances of ATSP are all specific real-world ones taken from TSPLIB [31].

In this paper, the recency factor λ in (6) and (8) was set to 0 in the TD method for the reason of simplicity. Hence, the following experiments mainly focus on the evaluation of the TD(0) and approximate Sarsa methods. The initial results are encouraging and show the potential for neurodynamic programming techniques applied to solving large-scale combinational optimization problems.

4.1 Case I

Table 1 represents the results of the first experiment on the random instance of the ETSP. The TD (0) and approximate Sarsa are compared with the method based on the standard Hopfield–Tank network, which is the first neural network approach applied to the TSP.

The continuous Hopfield–Tank network was constructed as an $n \times n$ (n refers to the number of cities) matrix of nodes that were used to encode solutions to the TSP. The energy function of this network and its parameter settings used in this case are described as follows.

$$\begin{aligned} E = & \frac{A}{2}\sum_x\sum_i\sum_{j\neq i} V_{xi}V_{xj} + \frac{B}{2}\sum_i\sum_x\sum_{y\neq x} V_{xi}V_{yi} \\ & + \frac{C}{2}\Big(\sum_x\sum_i V_{xi} - n\Big)^2 \\ & + \frac{D}{2}\sum_x\sum_{x\neq y}\sum_i d_{xy}V_{xi}(V_{y,i+1} + V_{y,i-1}) \end{aligned} \tag{11}$$

where A,B,C,D are parameters used for weighting the various components of the energy. More details about (11) can be found in [30].

We set $A = 0.5$, $B = 0.5$, $C = 0.2$, $D = 0.1$, $\Delta t = 10^{-5}$ in this experiment. The initial input values $U_{xi}(0) = 0.02 \cdot ln(n-1) +$ *random noise*, where random noise distributed within the range from –0.1 to 0.1.

The procedure adopting the Hopfield network run for 636 iterations to obtain the best result listed in Table 1. The average of iterations was 453.

In order to obtain statistically reliable results, the *repeated presentations* [33] paradigm was used for training the two kinds of evaluation networks Cnet and Qnet in this paper. Specifically, the weight vectors were only updated after the complete presentation of a training set. Each training set was presented repeatedly to each learning procedure until the procedure no longer produced any significant changes in the weight vectors.

In the first experiment, we constructed 100 training sets, each consisting of 10 sequences (one sequence means a complete tour), for training both the Cnet and Qnet. The learning curves presented in Fig. 3 show the mean performances of the two methods over this experiment.

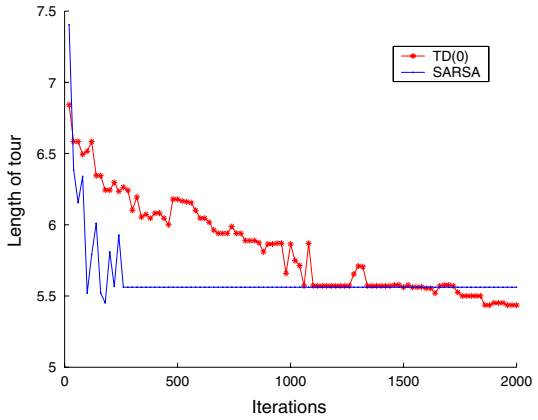

Fig. 3 Learning curves of the TD(0) and Sarsa over the 10-city ETSP

4.2 Case II

The results on the geometric instances of the ETSP are reported in Table 2. These benchmark instances can be found in TSPLIB [31].

In this case, although different initial configurations have been applied to the Hopfield–Tank network, it cannot converge to any feasible tour within 10,000 iterations. The reason mainly lies in the fact that the constraints of the TSP, namely that each city must be visited exactly once, are not strictly enforced but rather introduced into the energy function as penalty terms. So it seems very difficult to find good parameter settings for solving large-scale TSP problems. On the other hand, the topological structures of cities may have some effects on the performances of this method.

In comparison with the Hopfield–Tank network, the SOM network is more effective in solving the ETSP. The SOM network in its original form performs poorly for the ETSP, mainly because the scheme changes the weights based only on the distance of the winning neuron to the various neurons on the ring [1]. Burke et al. [4] modified the above strategy in their guilty net (GN) algorithm. Reported computational results indicate that the GN performs better than the Hopfield–Tank network [30]. Hence, we compare the TD(0) and approximate Sarsa with the GN. The heuristics with which we compare are GA and SA. Comparison is performed on the best results because of the availability of published results. In Table 2, results using GN are from [1], those using GA and SA are from [20], and the optimal values are from TSPLIB for Eil51, st70 and Eil76. In the context, the optimal lengths refer to the best, or at least the best known results.

In the second experiment, we constructed 30 training sets, each consisting of 5 sequences (one sequence means a complete tour), for training both the Cnet and Qnet. The learning curves presented in Figs. 4, 5 and 6 show the mean performances of the two methods over the second experiment.

4.3 Case III

In above comparisons with some neural network approaches and classical heuristics, the TD(0) and approximate Sarsa give acceptable results. Hence, their performance in solving the ATSP is expected. Since the ATSP is based on a directed graph and can be regarded as a generalization of the ETSP, it is much more difficult to solve than the ETSP. Using exact methods, the largest ETSP solved optimally has 2,392 cities [29] while the largest ATSP solved optimally has no more than 200 cities [14]. Although it generally outperforms the Hopfield–Tank algorithm on the ETSP, the SOM approach is restricted to geometric instances and cannot be applied for the ATSP. In contrast, both the TD(0) and approximate Sarsa maintain applicable when applied to the ATSP.

In this case, we compare the TD(0) and approximate Sarsa with some heuristics such as the nearest neighbor construction heuristic [13], the nearest neighbor heuristic combined with a fast-3-Opt tour improvement heuristic (NN-fast-3-Opt)[13], ATSP-GA [13] and ACS-3-opt [10]. The ATSP-GA approach was proposed by Freisleben and Merz. This approach uses a nearest-neighbor heuristic for creating the initial population of a GA, applies the fast-3-Opt heuristic [2] to this initial population for producing a population of local optima, and then uses GA to search the space of local optima in order to find the global optimum. ACS-3-opt presented in [10] is a version of the ACS augmented with a restricted 3-opt tour improvement heuristic. The ACS is a tour construction heuristic which produces a set of feasible solutions mutated from the previous best solution and then applies the restricted 3-opt [24] to locally optimize these solutions.

Table 3 reports the results obtained by the TD(0) and approximate Sarsa on the specific real-world instances, which are also taken from TSPLIB [31]. In this table, comparison is performed on the best results. Results using

Table 2 Comparison of the TD(0) and approximate Sarsa with other algorithms on geometric instances of the ETSP

Problem Name	TD(0)	Sarsa	GN	GA	SA	Optimum
Eil51 (51-city ETSP)	504.5925	503.3179	470.7	428	443	426
st70 (70-city ETSP)	783.0206	843.7116	755.7	NA	NA	675
Eil76 (76-city ETSP)	655.8313	643.3601	614.3	542	580	538

NA not available

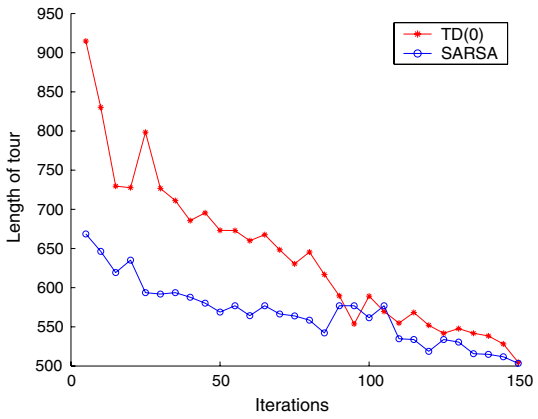

Fig. 4 Learning curves of the TD(0) and Sarsa over the 51-city ETSP

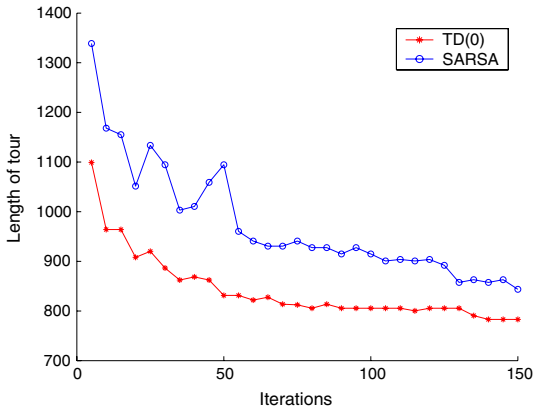

Fig. 5 Learning curves of the TD(0) and Sarsa over the 70-city ETSP

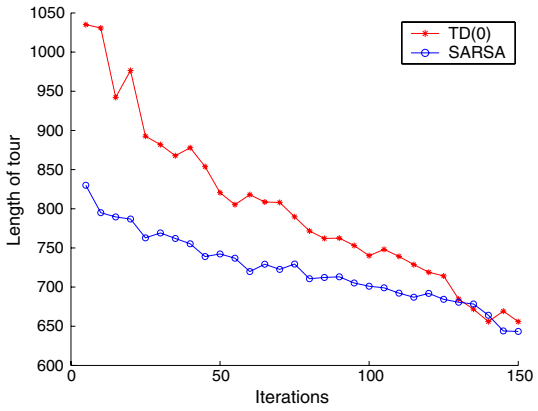

Fig. 6 Learning curves of the TD(0) and Sarsa over the 76-city ETSP

the nearest neighbor and NN-fast-3-Opt are from [13], those using ATSP-GA and ACS-3-opt are from [3], and the optimal values are from TSPLIB [31] for ry48p, ft70 and kro124p.

In this experiment, for instances ry48p and ft70 we constructed 30 training sets, each consisting of 5 sequences (one sequence means a complete tour), for training both the Cnet and Qnet. For instance kro124p, 60 training sets were used, each of which consists of 2 sequences. The learning curves presented in Figs. 7, 8 and 9 show the mean performances of the two methods over the third experiment.

5 Discussion and conclusion

According to the computational results presented in the preceding section, some discussions are made on the following aspects.

Observing the learning curves of each method in Figs. 3, 4, 5, 6, 7, 8 and 9, an apparent fact can be found that the TD(0) usually converges slower than the approximate Sarsa method, whereas the latter is easy to stuck in the local minima. An intuitive explanation is the learning agent following the ε-greedy policy is gradually prone to make the decisions based on past experience instead of trying a new potentially profitable action when parameter ε decreases with time. Therefore, the larger the state space is, the easier the agent plunges in the local minima. On the other hand, the reason why the TD(0) method can work using the random policy is that the training procedure of the evaluation network Cnet is driven by the temporally successive error, so that the output of Cnet can approximate the maximum-likelihood estimate based on past experience. As for the lower speed of convergence to the optimal, the reason is that no reinforcement signal has been used for improving policy during learning.

As can be seen from the Table 1, both the TD(0) and approximate Sarsa methods offer better performance than the method based on the Hopfield–Tank network on the quality of solutions. In the meanwhile, the latter method cannot get any feasible solution in the second experiment. So it seems that both methods presented in this paper are more effective than this classical neural network approach in solving the large-scale TSP.

Based on comparison shown in Tables 2 and 3, the initial results provided by both the TD(0) and approximate Sarsa are not compatible with the ones produced by the classical heuristics such as GA, SA, etc. The reason lies in three difficulties. The first one is the dilemma of the generalization capability implemented by MLP. On the one hand, generalization is a key in finding a suboptimal solution in a large state space. As an evaluation function approximator, MLP would produce reasonable outputs for inputs not encountered during learning based on past experience. On the other hand, it is inevitable for MLP to learn the evaluation value for new states at the cost of changing the mapping values of other states learned.

Another difficulty lies in the traveling rule that the learning agent has to obey. A certain action cannot be taken

Table 3 Comparison of the TD(0) and approximate Sarsa with other heuristics on ATSP instances

Problem name	TD(0)	Sarsa	Nearest-neighbor	NN-fast-3Opt	ATSP-GA	ACS-3-opt	Optimum
ry48 (48-city ATSP)	16,779	17,709	15,368	14,846	14,422	14,422	14,422
ft70 (70-city ATSP)	42,929	42,764	42,270	39,783	38,673	38,781	38,673
krol124p (100-city ATSP)	42,798	43,783	42,462	38,157	36,230	36,241	36,230

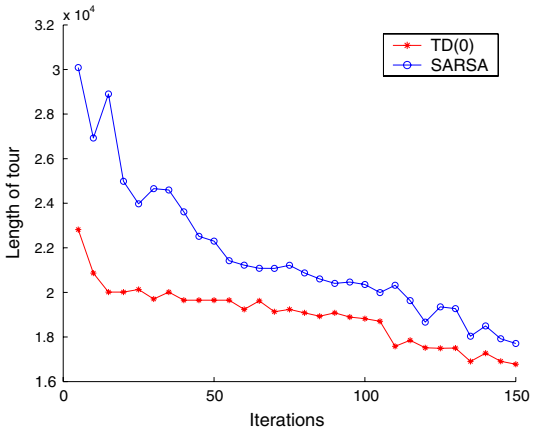

Fig. 7 Learning curves of the 48-city ATSP

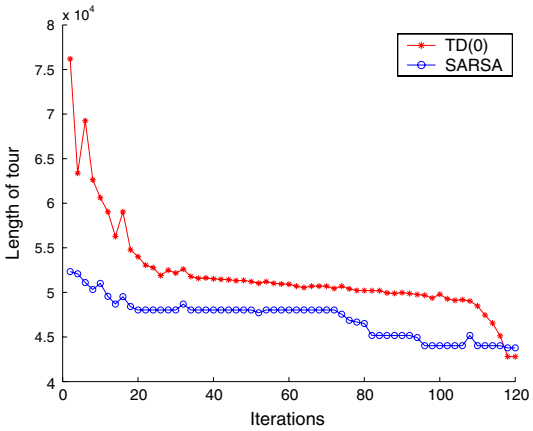

Fig. 9 Learning curves of the 100-city ATSP

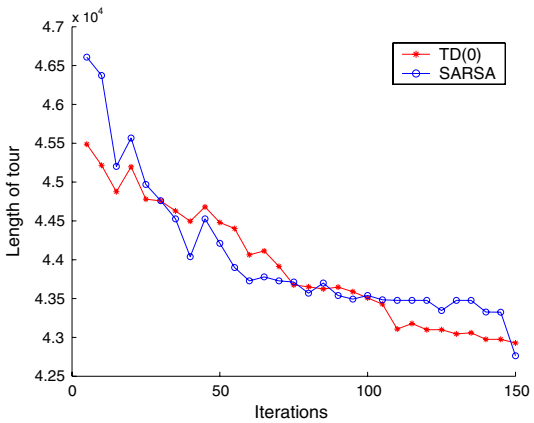

Fig. 8 Learning curves of the 70-city ATSP

when the learning agent moves forward. This leads to an extreme decrease in the scope of the admissible state-action space for the agent to explore. Accordingly, more training sets will be needed to get a satisfactory performance for both methods presented in this paper.

The third difficulty is the tradeoff between exploration and exploitation. This point has been discussed at the beginning of this section.

However, in spite of the difficulties mentioned above, the initial results are very encouraging since only a few training sets have been used in contrast with the large size of the state space. Furthermore, the experimental results have shown that the TD(0) and approximate Sarsa maintain effective in solving both the ETSP and the ATSP. It indicates that neurodynamic programming is an effective strategy for the TSP.

In addition, comparison reported in Table 3 has shown that the combined algorithms such as NN-fast-3-Opt, ATSP-GA and ACS-3-op clearly outperform the classical near-neighbor heuristic. Although it has been proven that some classical heuritics, when applied individually, can produce quite good quality solutions, combined algorithms seem much more robust. So in order to improve the performance of the TD(0) and approximate Sarsa, a possible way is to build some efficient tour improvement heuristics into these neurodynamic programming approaches. It will be the subject of future research.

Acknowledgments This research has been supported in part by the National Natural Science Foundation of China (Grants 60205004, 50475179, 60528002, 60621001, and 60635010), the National Basic Research Program (973) of China under Grant 2002CB312200, and the Hi-Tech R&D Program (863) of China (Grant 2006AA04Z258).

References

1. Aras N, Oommen BJ, Altinel IK (1999) The kohonen network incorporating explicit statistics and its application to the traveling salesman problem. Neural Netw 12(9):1273–1284
2. Bentley J (1992) Fast algorithms for geometric traveling salesman problems. ORSA J Comput 4(4):387–411
3. Bersini H, Dorigo M, Langerman S, Seront G, Gambardella L (1996) Results of the first international contest on evolutionary

optimization (1st iceo). In: Evolutionary computation. Proceedings of IEEE international conference on. Springer-Verlag, Nagoya, pp 611–615
4. Burke LI, Damany P (1992) The guilty net for the traveling salesman problem. Comput Oper Res 19(3–4):255–265
5. Christofides N (1976) Worst-case analysis of a new heuristic for the travelling salesman problem. Tech Rep 388. Carnegie-Melon University, Pittsburgh
6. Christofides N, Eilon S (1972) Algorithms for large-scale traveling salesman problems. Oper Res Quart 23(4):511–518
7. Clarke G, Wright JW (1964) Scheduling of vehicles from a central depot to a number of delivery points. Oper Res 12(4):568–581
8. Crites R, Barto A (1996) Improving elevator performance using reinforcement learning. Adv Neural Inf Process Syst 8:1017–1023
9. Croes GA (1958) A method for solving traveling-salesman problems. Oper Res 6(6):791–812
10. Dorigo M, Gambardella L (1997) Ant colony system: a cooperative learning approach to the travelingsalesman problem. Evolut Comput IEEE Trans 1(1):53–66
11. Durbin R, Willshaw D (1987) An analogue approach to the traveling salesman problem using an elastic net method. Nature 326(6114):689–691
12. Fiechter CN (1994) A parallel tabu search algorithm for large traveling salesman problems. Discrete Appl Math 51(3):243–267
13. Freisleben B, Merz P (1996) A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In: International conference on evolutionary computation, pp 616–621
14. Gambardella LM, Dorigo M (1995) Ant-q: a reinforcement learning approach to the traveling salesman problem. In: International conference on machine learning, pp 252–260
15. Gendreau M, Hertz A, Laporte G (1992) New insertion and postoptimization procedures for the traveling salesman problem. Oper Res 40(6):1086–1094
16. Golden BL, Stewart WR et al (1985) Empirical analysis of heuristics. In: Lawler EL, Lenstra JK, Kan AHGR, Shmoys DB (eds) The traveling salesman problem. A guided tour of combinatorial optimization. Wiley, Chichester, pp 207–249
17. Haykin S (1998) Neural networks: a comprehensive foundation. Prentice Hall, PTR Upper Saddle River
18. Homaifar A, Guan S, Liepins GE (1993) A new approach on the traveling salesman problem by genetic algorithms. In: Proceedings of the 5th international conference on genetic algorithms. Morgan Kaufmann Publishers Inc., San Francisco, pp 460–466
19. Hopfield JJ, Tank DW (1985) Neural computation of decisions in optimization problems. Biol Cybern 52(3):141–152
20. Jayalakshmi G, Sathiamoorthy S, Rajaram R (2001) An hybrid genetic algorithm—a new approach to solve traveling salesman problem. Int J Comput Eng Sci 2(2):339–355
21. Johnson DS (1990) Local optimization and the traveling salesman problem. In: Goos G, Hartmanis J (eds) ICALP '90: proceedings of the 17th international colloquium on automata, languages and programming. Springer-Verlag, London, pp 446–461
22. Johnson DS, McGeoch LA (1997) The travelling salesman: a case study in local optimization. In: Aarts EHL, Lenstra JK (eds) Local search in combinatorial optimization. Wiley, New York, pp 215–310
23. Kaelbling LP, Littman ML, Moore AP (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
24. Kanellakis P, Papadimitriou C (1980) Local Search for the asymmetric traveling salesman problem. Oper Res 28(5):1086–1099
25. Laarhoven P, Aarts E (1987) Simulated annealing: theory and applications. Kluwer, Norwell
26. Lin S (1965) Computer solutions of the traveling salesman problem. Bell Syst Tech J 44(10):2245–2269
27. Lin S, Kernighan BW (1973) an effective heuristic algorithm for the traveling-salesman problem. Oper Res 21(2):498-516
28. Miagkikh VV, Punch WF (1999) An approach to solving combinatorial optimization problems using a population of reinforcement learning agents. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) Proceedings of the genetic and evolutionary computation conference, vol 2. Morgan Kaufmann, Orlando, pp 1358–1365
29. Padberg M, Rinaldi G (1990) Facet identification for the symmetric traveling salesman polytope. Math Programm 47(1):219–257
30. Potvin J (1993) The traveling salesman problem: a neural network perspective. ORSA J Comput 5:328–348
31. Reinelt G (1991) TSPLIB—a traveling salesman problem library. ORSA J Comput 3(4):376–384
32. Singh S, Bertsekas D (1997) Reinforcement learning for dynamic channel allocation in cellular telephone systems. In: Mozer MC, Jordan MI, Petsche T (eds) Advances in neural information processing systems, vol 9. The MIT Press, Cambridge, pp 974–980
33. Sutton R (1988) Learning to predict by the methods of temporal differences. Mach Learn 3(1):9–44
34. Sutton R, Barto A (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
35. Tesauro G (1995) Temporal difference learning and TD-Gammon. Commun ACM 38(3):58–68
36. Watkins C, Dayan P (1992) Technical note: Q-Learning. Mach Learn 8(3):279–292
37. Willshaw D, von der Malsburg C (1979) A marker induction mechanism for the establishment of ordered neural mappings: its application to the retinotectal problem. Philos Trans R Soc Lond B Biol Sci 287(1021):203–243