Neural Comput & Applic (2018) 30:2935–2951
https://doi.org/10.1007/s00521-017-2880-4

ORIGINAL ARTICLE

A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem

Mir Mohammad Alipour[1] · Seyed Naser Razavi[1] · Mohammad Reza Feizi Derakhshi[1] · Mohammad Ali Balafar[1]

Received: 31 March 2016 / Accepted: 10 February 2017 / Published online: 23 February 2017

Abstract In recent years, hybrid genetic algorithms (GAs) have received significant interest and are widely being used to solve real-world problems. The hybridization of heuristic methods aims at incorporating benefits of stand-alone heuristics in order to achieve better results for the optimization problem. In this paper, we propose a hybridization of GAs and Multiagent Reinforcement Learning (MARL) heuristic for solving Traveling Salesman Problem (TSP). The hybridization process is implemented by producing the initial population of GA, using MARL heuristic. In this way, GA with a novel crossover operator, which we have called Smart Multi-point crossover, acts as tour improvement heuristic and MARL acts as construction heuristic. Numerical results based on several TSP datasets taken from the TSPLIB demonstrate that proposed method found optimum solution of many TSP datasets and near optimum of the others and enable to compete with nine state-of-the-art algorithms, in terms of solution quality and CPU time.

Keywords Hybrid genetic and multiagent reinforcement learning algorithm (GA + MARL) · Traveling salesman problem · Smart Multi-point crossover (SMX) · MARL heuristic

✉ Mir Mohammad Alipour
mohammad.alipour@tabrizu.ac.ir

Seyed Naser Razavi
n.razavi@tabrizu.ac.ir

Mohammad Reza Feizi Derakhshi
mfeizi@tabrizu.ac.ir

Mohammad Ali Balafar
Balafarila@tabrizu.ac.ir

[1] Department of Computer Engineering, Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran

1 Introduction

The traveling salesman problem is the most well-known combinatorial optimization problem. Traveling Salesman Problem (TSP) is used to find a routing of a salesman who starts from a home location, visits a prescribed set of cities and returns to the original location in such a way that the total distance travelled is minimized and each city is visited exactly once. TSP is solved very easily when there is less number of cities, but as the number of cities increases it is very hard to solve, as large amount of computation time is required. The numbers of fields where TSP can be used very effectively are traffic and military.

There are two classes of algorithms for solving the TSP: exact algorithms and heuristic algorithms. Exact algorithms are algorithms that always solve an optimization problem to optimality and guarantee to terminate with an optimal solution. Heuristics are designed to solve problems in a faster and more efficient fashion than traditional methods by sacrificing optimality, accuracy, precision or completeness for speed. These algorithms are most often employed when approximate solutions are sufficient and exact methods are rather computationally expensive. Approximate algorithms for the TSP may be subdivided into three classes: tour construction, tour improvement and hybrid algorithms [1].

Dynamic programming, branch and bound, linear programming, cutting plane methods are exact methods with different complexities to solve TSP problems but they all not applicable for large-scale problems [2].

Tour construction algorithms iteratively extend a connected partial tour or iteratively combine tour fragments of the best tour. Greedy-type or nearest neighbor construction heuristics starts with a vertex randomly chosen, and each vertex is connected to the vertex with minimal distance. Although nearest neighbor heuristic does not find high-quality solutions, it is integrated with perturbative search methods by which one or more solution components are modified in each step [3–5].

The tour improvement algorithms start with a given tour and replace two or more links of the tour with other links to obtain a shorter tour. 2-opt, 3-opt and λ-opt local search methods used as a mutation operator in tour improvement algorithms. They remove some edges from the tour and insert the new edge if the new tour is shorter [3].

Hybrid algorithms include both construction and improvement modes. Population-based heuristics such ant colony optimization (ACO) algorithms [6–11], neural networks [12–16], evolutionary algorithms [17–19], simulated annealing [20–24], tabu search [21, 23, 25, 26], artificial bee colony algorithm [9, 27, 28] and particle swarm optimization [20, 29] are also used to solve combinatorial problems by employing tour construction and tour improvement operators.

The ACO algorithm for the traveling salesman problem has been introduced by Dorigo and Gambardella based on the fact that ants are able to find the shortest route between their nest and a source of food. A major deficiency of their algorithm is the premature stagnation of the search. Afterward, they used a different state transition rule and added a local pheromone updating rule to improve the performance of primary ACO [6–8].

Dong et al. [10] presented a hybrid approach, called cooperative genetic ant system (CGAS) which combines GA and ACO for solving TSP. The information exchange between ACO and GA is performed at the end of the each iteration. This exchange is used to ensure the selection of the best solutions for next iteration. Through this cooperation, the algorithm has a better chance to reach a global optimum. The experimental results show that CGAS outperforms the other GA/ACO algorithms on the TSP instances.

Gündüz et al. [9] presented a new hierarchic method based on the ACO algorithm and the artificial bee colony (ABC) algorithm. ACO is used for the path construction, and ABC is used for the path improvements. Thus, their hierarchic method was proposed to achieve a good solution in a reasonable time. The experimental results show that ACO-ABC algorithm has better performance than individual approaches of ACO and ABC.

Yong [11] presented a hybrid max–min ant system (HMMA) with a local search algorithm. First, the max–min ant system is used to find an approximate solution. Afterward, the local paths of adjacent four vertices in the approximate solution are converted into the local optimal paths with the four vertices and three lines inequality to get the better approximate solution. HMMA is experimented with small- and large-scale TSP instances.

GA is one of the computational models inspired by evolution in the nature, which has been used to a large number of real-world problems. Also, GA is used to get approximate solutions for TSP. High adaptability and the generalizing feature of GA help to execute the TSP by a simple structure [17–19].

The combination of local search heuristics and genetic algorithm is a capable approach for finding near-optimum solutions to the TSP. Merz and Freisleben [30] offered an approach in which local search techniques are used to find local optima in a given TSP search space, and genetic algorithms are used to search the space of local optima in order to find the global optimum. They utilized new genetic operators for realizing the proposed approach, and the quality and efficiency of the solutions obtained for a set of symmetric and asymmetric TSP instances are discussed.

White and Yen [31] give details the development of a hybrid evolutionary algorithm for solving the TSP. The stratagem of the algorithm is to broaden the successful results of a genetic algorithm using a distance preserving crossover (DPX) by including memory in the form of ant pheromone during the city selection process. The synergistic combination of the DPX-GA with city selection based on probability found out by both distance and previous success adds additional information into the search mechanism. This combination into a hybrid GA facilitates finding quality solutions for TSP problems with lower computation complexity.

Machado and Lopes [32] proposed hybrid approach that joins PSO, genetic algorithms and fast local search for the TSP. The positions of the particles represent TSP tours as permutations of $|N|$ cities. The value assigned to each particle (fitness) is the rate between a constant Dmin and the cost of the tour represented in the particle's position.

Yang et al. [33] used generalized chromosome genetic algorithm (GCGA) to solve the classical TSP, which was previously proposed for solving generalized TSPs (GTSP) by these authors. Numerical experiments show the advantages of the GCGA for solving a large-scale TSP.

Chen and Chien [20] presented a new method, called the genetic simulated annealing ant colony system with particle swarm optimization techniques (GSAP), for solving the traveling salesman problem. They use the ant colony system to generate the initial solutions of the genetic algorithms. Then, use the genetic simulated annealing techniques to generate better solutions based on the initial solutions. After a predefined number of cycles, the system

uses the particle swarm optimization techniques to exchange the pheromone information between groups.

Wang proposed a new hybrid genetic algorithm with two local optimization strategies (HGA) to solve the STSP [34]. Two local search methods are applied to improve the genetic algorithm. The first local search is a four vertices and three lines inequality applied to local Hamiltonian paths to generate shorter tours. The second local search is then executed to reverse the local Hamiltonian paths with more than two vertices, which also generates shorter tours.

Deng et al. [35] offered an approach in which new initial population strategy based on the *k*-means algorithm (KIP) is used to improve the genetic algorithm for solving TSP. The results show that KIP can decrease best error value of random initial population strategy and greedy initial population strategy with the ratio of approximately between 29.15 and 37.87%, average error value between 25.16 and 34.39% in the same running time.

In addition, the other intelligent algorithms are combined with GA to improve its performance. Lima et al. [36] used a technique of reinforcement learning, the Q-learning algorithm, for the constructive phase of the GRASP metaheuristic and to generate the initial population of a genetic algorithm. The proposed methods are applied to the symmetrical traveling salesman problem.

Liu and Zeng [37] proposed a new genetic algorithm with reinforcement learning (RMGA) to solve the TSP. Their method uses reinforcement mutation to improve the GA. The experimental results show that RMGA outperforms the known EAX-GA and LKH in the quality of solutions and the running time.

Santos et al. [38] proposed a parallel hybrid GRASP/GA using reinforcement learning to solve the symmetric TSP. They improved the performance of a metaheuristic using multiple search trajectories, which act competitively and/or cooperatively and also reinforcement learning is applied for guiding the metaheuristic to regions of promising solutions using the acquisition of information on the problem.

In this paper, we propose a hybrid algorithm including, GA with a novel crossover operator, Smart Multi-point crossover (SMX), as tour improvement heuristic and Multiagent Reinforcement Learning (MARL) [39], as tour construction heuristic to solve traveling salesman problem. In this way, initial population of GA is generated using MARL heuristic, and then, these tours are improved by GA as tour improvement heuristic in order to achieve better solutions based on the given initial solutions.

The paper is organized as follows. After this introduction, some necessary preliminaries about the TSP and GA are listed in Sect. 2. Section 3 gives some details about the MARL heuristic for TSP. Section 4 describes the proposed hybrid algorithm. Section 5 presents the obtained experimental results and its comparison to the other heuristic methods. Finally, Sect. 6 states the conclusion of our paper.

2 Preliminaries

2.1 Traveling salesman problem (TSP)

The TSP can be generally defined as the consideration of a set N of nodes representing the cities, and a set E of edges that fully connect the nodes N, where $W(i,j)$ is the distance between cities i and j (length of the edge $(i,j) \in E$), with $i,j \in N$. The TSP is the problem of finding a minimal length Hamiltonian tour on the undirected graph $G = (N,E)$, where an Hamiltonian tour of graph G is a closed tour that entails visiting once and only once all the $n = |N|$ nodes of G, and where its length is given by the sum of the lengths of all the edges of which it is composed [40]. The aim of solving TSP is to minimize the total closed tour length f and is defined as Eq. (1).

$$f = \sum_{i=1}^{n-1} W(i, i+1) + W(n, 1) \tag{1}$$

where n is the total number of cities.

2.2 Genetic algorithm

The genetic algorithm (GA) is a heuristic search algorithm which has been used to solve search and optimization problems. It is based on the principle of "the survival of the fittest," given by Charles Darwin. It is said to simulate the natural evolution carried out in living beings [41].

GA begin with various problem solutions which are encoded into population, a fitness function is applied for evaluating the fitness of each individual, after that a new generation is created through the process of selection, crossover and mutation. After the termination of genetic algorithm, an optimal solution is obtained. If the termination condition is not satisfied, then algorithm continues with new population. The basic steps of genetic algorithm used are given below:

Initialization An initial population is generated from many individual solutions. A problem depends upon size of the population that contains several hundreds or thousands of possible solutions. The search space contains all the possible solutions from which the population is generated randomly. However, the solutions are seeded in the areas from where the optimal solutions are likely to be found. In this, if there are N numbers of cities, then $N!$ possible number of solution can be made.

Encoding scheme In this, we represent each city with number, for example if there are 10 cities then each city is represented from number 1 to 10 such as 9 1 10 2 4 3 6 7 5 8, and no number will be repeated. For N number of cities, the cities are represented by permutation of integers from 1 to N.

Selection This operator is used to select the fitter chromosomes from the population for next breeding. The selected chromosomes are called parent chromosomes. The chromosomes selected with largest fitness value.

Crossover operator Crossover operators are the backbone of the genetic algorithm. Reproduction makes clones of good strings but does not create new ones. Crossover operators are applied to mating pool with hope that it creates a better offspring. There are varieties of crossover and mutation operators which differ from each other in aspects like having fixed or variable length, being ordered or not and gene repetition. Among them, partially mapped crossover (PMX), order crossover (OX) and cycle crossover (CX), edge recombination (ER) and multipoint crossover (MX).

- *PMX method* PMX builds an offspring by choosing a subsequence of a combination from one parent and preserving the order and position of as many parameters as possible from the other parent [42]. A subsequence of a combination is selected by choosing two random cut points, which serve as boundaries for swapping operations. The PMX crossover exploits important similarities in the value and ordering simultaneously when used with an appropriate reproductive plan. However, there is a disadvantage with this crossover that happens when there is a tie in mapping.
- *OX method* OX builds offspring by choosing a subsequence of combination from one parent and preserving the relative order of parameters from the other parent [43].
- *CX method* CX builds offspring in such a way that each parameter (and its position) comes from one of the parents [44]. The CX preserves the absolute position of the elements in the parent sequence.
- *ER method* ER transfers more than 95% of the edges from the parents to the single offspring [45]. ER had the fastest convergence and converged on better solutions than other operators when optimizing a number of combinatorial problems of size 30–75 parameters.
- *MX method* In MX method, N cutoff points between 1 and length of chromosome are randomly selected to divide each parent chromosome into $N + 2$ parts, then odd parts of the first parent and even parts of the second parent generate the first offspring and odd parts of the second parent and even parts of the first parent generate the second offspring [46].

Mutation operator The mutation operator enhances the ability of the GA to find a near optimal solution to a given problem by maintaining a sufficient level of genetic variety in the population, which is needed to make sure that the entire solution space is used in the search for the best solution. In a sense, this operation avoids premature convergence and escape algorithm from local optima. The mutation operator generates a new offspring by randomly swapping genes. The probability of the mutation is a parameter of the genetic algorithm.

Evaluation The system evaluates the fitness value of each chromosome based on a fitness function, after the crossover operation and the mutation operation.

Termination The termination criteria can be characterized by the maximum number of iterations, the computation time or the number of iterations with no improvement. If the termination criteria are reached, the chromosome with the highest fitness value is the best solution. Otherwise, system continues.

3 MARL heuristic for TSP

In this work, we used the MARL heuristic [39] for constructing the primary tour of TSP, and then, 2-opt and nearest insertion into the convex hull local search (NICH-LS) [47] improve the given primary tour.

The MARL approach for solving TSP starts by creating a graph of TSP instance and placing m cooperative agents at that graph. Each agent is a traveling salesman and its tour is constructed by incrementally selecting cities (nodes or states) until all cities have been visited and coming back at first city of the tour. The MARL model consists of the learning environment, the learning algorithm and a reward function to evaluate the effectiveness of the agents learning [48, 49].

For each agent a scenario like this occurs: Initially, agent randomly placed at one of the graph's nodes as a first city of the tour. As a result, candidate list of first city plays roll of the agent's actions which now placed at that city. Candidate list cities have not been visited yet. Agent chooses one of its actions based on one of the selection methods. Then, agent moves at the selected city. The process of selecting an action and moving the agent at that city is repeated until a tour is created, this means, every node of the graph is visited and coming back at starting city (or making a feasible tour is impossible).

An iteration is finished, once the m agents have created their own tours. Then, best of the tours created by m agents in current iteration is selected, and the environment uses the length of this tour to produce its response. This response, depending on whether it is favorable or unfavorable, causes the selected actions of all states along the traversed path be rewarded if it is favorable according to Q-learning algorithm. This updating encourages the use of shorter routes and increases the probability that future routes will use the links along the best solutions. This process is repeated for a predetermined number of iterations, and after each iteration, all the tours created by m agents, which have shorter length than the best tour from last iterations, are inserted into candidate solutions set as initial population of the next stage. Of course, before adding the tours to candidate solutions set, they are optimized using 2-opt and NICH-LS.

NICH-LS improves the given tour by locally manipulating the order of nodes in the partial tours of the given tour via creating convex hull of the nodes in the partial tour and adding remaining nodes using nearest insertion method, and if the length of manipulated partial tour is smaller than the primary partial tour, then partial tour is replaced with manipulated partial tour, which reduces the length of the primary tour. The outline of the MARL algorithm is shown in Fig. 1.

4 Proposed hybrid algorithm (GA + MARL)

The suggested algorithm combines the advantages of MARL and GA; hence, MARL heuristic drives the exploration of the search space, thus, focusing on the global optimization task and produces an acceptable solution, and then, this solution is optimized using the GA improvement heuristic by visiting the promising subregions of the solution space. Thus, MARL is used as construction heuristic, and some of the best solutions obtained by MARL are given to the GA as initial population.

In this section, we propose an improvement heuristic based on GA and try to improve the tours in the candidate solutions set, taken from MARL + NICH-LS by using GA in order to increase quality of the solution.

4.1 Chromosome representation

The solutions of TSP problem (tours) can be represented by chromosomes that consist of genes. Each chromosome gene indicates a city. The values of these genes and their position in the "gene string" tell the genetic algorithm what solution the individual represents. TSP problem has been solved in different chromosome representations such as binary strings and matrices by the GA algorithm. The binary- and matrix-based representations usually use the

candidate solutions set = empty;
repeat // MNLI times
  for $s \in S$; $a \in A$ **do** Q(s, a) = 1; // Q-values are global and common between m agents.
  for all m agents **do** // each of m agents creates his own tour.
    $s_c = s_s$ = A random state (city) of the TSP graph;
    Disable action correspond to s_c of all states;
    for n times **do**
      $P(a|s_c) = \frac{\exp\{Q(s_c,a)*(W^{-1}(s_c,s_a))^{\beta}\}}{\sum_{\hat{a}\neq a}\exp\{Q(s_c,\hat{a})*(W^{-1}(s_c,s_{\hat{a}}))^{\beta}\}}$
      Select action a based on $\begin{cases} EG\% & \epsilon-\text{greedy method;} \\ SM\% & \text{Softmax method;} \end{cases}$ // where EG + SM = 1
      s_a = state correspond to action a;
      Disable action correspond to s_a of all states;
      $s_c = s_a$ 'move agent to state s_c;
    end for // all cities be visited
    Select action a correspond to s_s (first state) of s_c (last state) to complete the tour;
  end for // end of an iteration, m agents creates your own tours.
  candidate solutions set += each of m current iteration tours which shorter than best_tour;
  tour = best of the m tours from current iteration;
  if length(tour) < length(best_tour) **Then**
    a_s, selected actions of S, states along the tour are updated:
    $Q(S, a_s) \leftarrow Q(S, a_s) + \alpha * rd$;
    best_tour = tour;
  end if
  Enable all disabled actions of states;
until Learning is stopped.
Optimize tours in the candidate solutions set by 2-opt;
Optimize tours in the candidate solutions set by NICH-LS;

Fig. 1 Modified MARL algorithm

binary alphabets for the tour representation. The path representation is the most natural representation of a tour. In this representation, a path is a list of n cities, and if the city x is the y-th element of the list, then city x will be the y-th city to be visited. For example, path $7-3-5-6-4-2-1$ is represented as it is, that is via $7-3-5-6-4-2-1$. In this paper, we considered path representation for encoding the chromosomes.

4.2 Population initialization

As mentioned earlier, in the proposed hybrid algorithm, the initial population is taken from MARL heuristic [39]. On the other words, some of the best solutions obtained by MARL (tours in the candidate solutions set) are given to the GA as initial population.

4.3 Evaluation of fitness function

Since GA is generally applied on maximization problems and the TSP is a minimization problem, the inverse of tour length is considered as the fitness function.

4.4 Selection

Through selection operation step, parents are selected for crossover. Here, the roulette wheel selection method is selected.

4.5 Crossover operator

The main purpose of this component is to create offspring using a given pair of solutions chosen through the selection operation procedure. This paper proposes a new crossover operator. Details of the proposed crossover operator and hybrid GA are explained in Sect. 4.8.

4.6 Mutation operator

Mutation operator is performed to the new chromosomes after crossover. It is used to preserve the genetic diversity of chromosomes at each generation of population. In this paper, exchange mutation operator (EM) is used. The EM randomly selects two cities in a tour and exchanges them. The mutation probability PM is taken as 0.1.

4.7 Termination criteria

In the proposed approach, the maximum number of iterations (MNSMX) is used as the termination criterion. After reaching the termination criteria, the chromosome with the highest fitness value is the best solution.

4.8 Proposed smart multipoint crossover operator

The proposed SMX receives two individuals (parent chromosomes) to be recombined and returns a new individual (offspring chromosome) created based on recombination of the parent chromosomes. The outline of SMX is as shown in Fig. 2.

The recombination of genes occurs as follows: Secific sequences of genes are inserted into the offspring between both chromosomes. To better understand, Fig. 3 presents two individuals to be recombined: Parent1 and Parent2.

- At stage II, random number $m = 2$ is generated; therefore, Parent1's genes belonging to sequence 1–2 (two genes of Parent1 from pnp to pnp + 1) are inserted into offspring (genes 2,5).
- At stage III, parent is a Parent2, pno = 3, random number $m = 4$ is generated, position of the last inserted gene of offspring (gene 5) is 4 at Parent2; therefore, pnp = 5 and Parent2's four genes after position 4 which not exist in the offspring are inserted circularly into offspring (genes 3,9,4,7).
- At stage IV, parent is a Parent1, pno = 7, random number $m = 1$ is generated, position of the last inserted gene of offspring (gene 7) is 7 at Parent1; therefore, pnp = 8 and Parent1's one gene after position 7 which not exist in the offspring is inserted circularly into offspring (gene 1).
- At stage V, parent is a Parent2, pno = 8, random number $m = 3$ is generated, position of the last inserted gene of offspring (gene 1) is 2 at Parent2; therefore, pnp = 3 and Parent2's maximum three genes after position 2 which not exist in the offspring is inserted circularly into offspring (genes 6,8).

If the length of generated offspring is smaller than the best individual length (individual with smallest length in the population), then first this favor offspring is optimized using 2-opt and NICH-LS, next this offspring is added to the population pool, otherwise discard it. This process continues for predefined numbers of iterations (MNSMX). When stopping criteria are satisfied, GA improvement heuristic stops and shows the best individual which is the best tour that has been created by the heuristic. The outline of our GA improvement heuristic for TSP is shown in Fig. 4. In Fig. 5, the working diagram of proposed hybrid algorithm is depicted.

5 Experimental results

In this section, we report on experimental results obtained with proposed hybrid algorithm, on 34 standard benchmark instances (datasets) from the TSPLIB [50]. Since proposed

Fig. 2 Outline of SMX

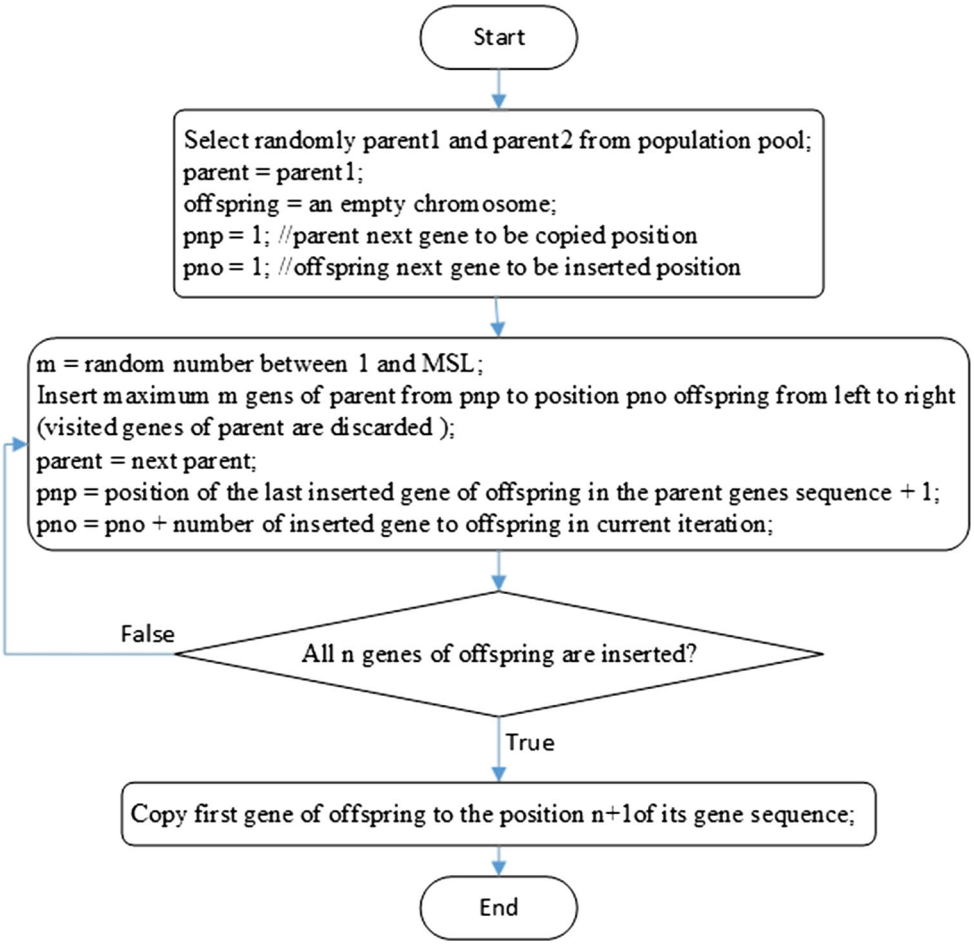
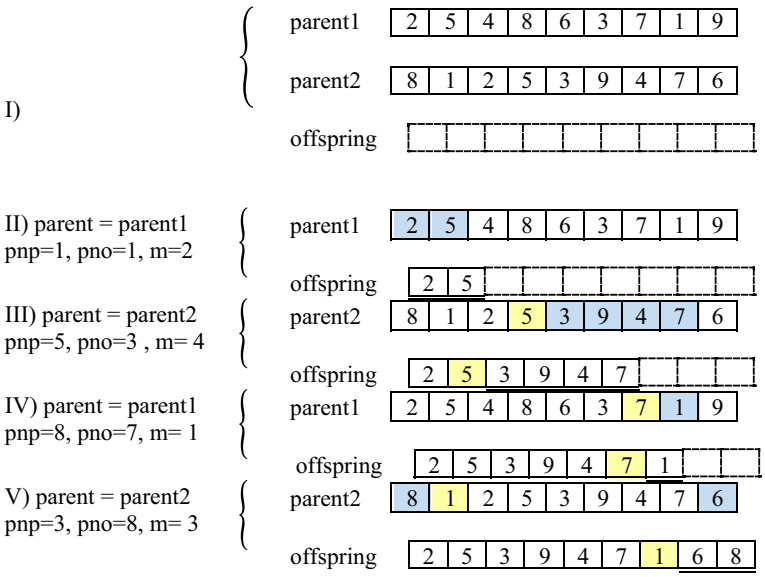


Fig. 3 Recombination of two individuals using SMX operator



algorithm have a stochastic component, thus may produce different solutions over multiple runs on the same dataset. Therefore, each experiment was performed 20 independent runs, and the best, worst and average were recorded for each run and the presented results are the average of 20 runs, for each dataset.

The proposed algorithm has been implemented using Microsoft Visual studio.net 2013 under 64-bit Windows

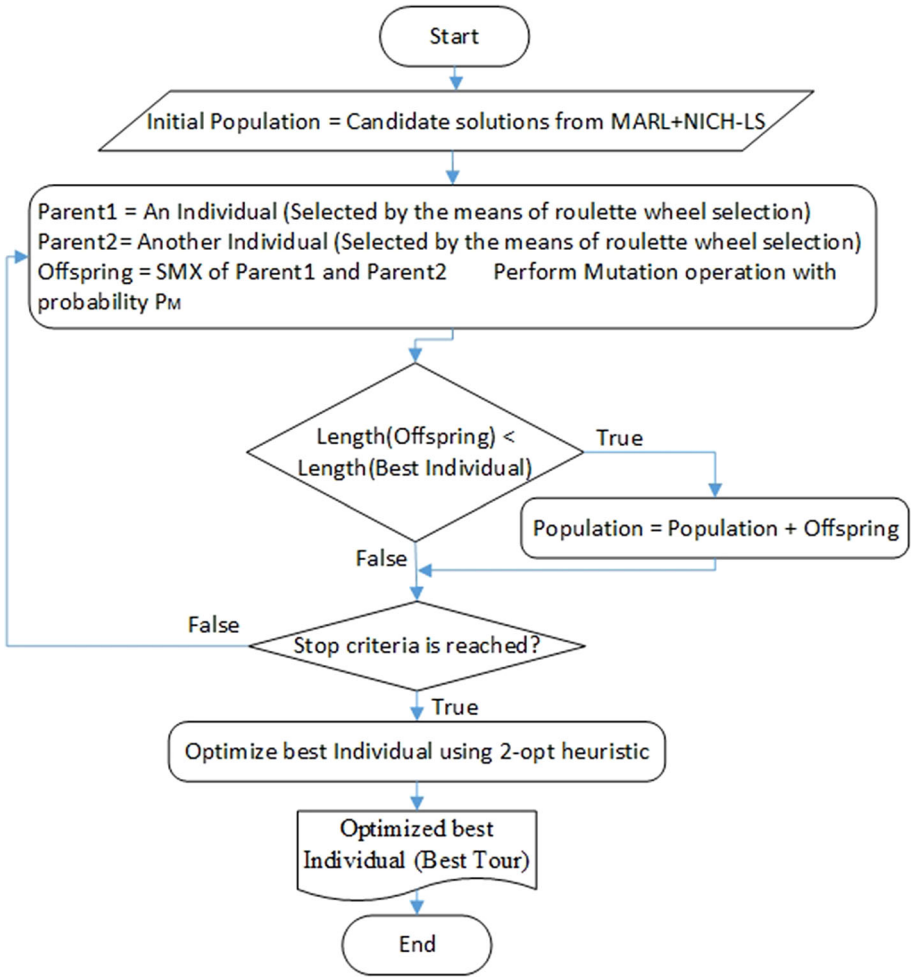


Fig. 4 Outline of GA improvement heuristic for TSP

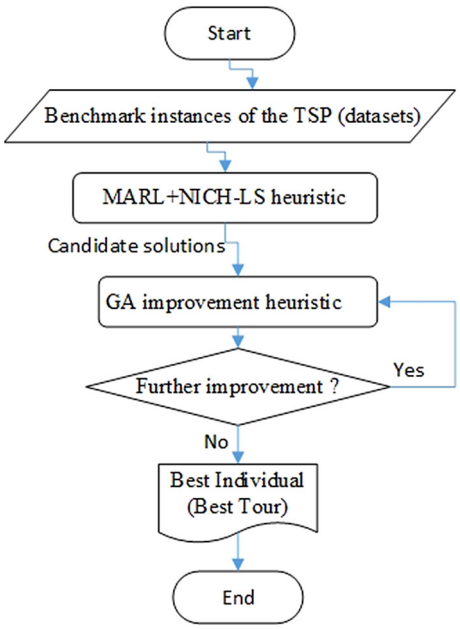


Fig. 5 Working diagram of proposed hybrid algorithm (GA + MARL)

8.1 operating system. Experiments are conducted on a laptop with Intel Core-i5-4200U, 1.6 GHz CPU and 4 GB of RAM. In the proposed algorithm, the values of parameters are selected based on some preliminary trials. The selected parameters are those values that gave the best results concerning both the solution quality and the computational time needed to reach this solution. Table 1 shows the parameter settings of the proposed method.

When developing autonomous learning agents, the performance depends crucially on the selection of reasonable learning parameters, for example, learning rates or exploration parameters [51]. In other words, successful reinforcement learning highly depends on the careful setting of learning parameters in reinforcement learning. Generally speaking, it is crucial that all the learning parameters are carefully tuned to elicit good performance in advance [52].

Figure 6 shows the results of running MARL algorithm on dataset eil101 with different values of learning parameters. For each experiment, a run of 10000 iterations (MNLI) is performed and algorithm was executed 20 times independently.

Table 1 Parameter settings for hybrid algorithm

Parameter	Value	Meaning
Beta (β)	2	Distance ratio
EG	0.85	$\in$-greedy action selection probability
SM	$1 - \text{EG}$	Softmax action selection probability
rd	1	Reward
Alpha (α)	0.75	Learning rate
MNLI	1000	Maximum number of learning iterations
m	5	Number of agents
MSL	n	Maximum substring length
MNSMX	$(100 \times \log_{10} n)^2$	Maximum number of SMX iterations
PM	0.1	Mutation probability

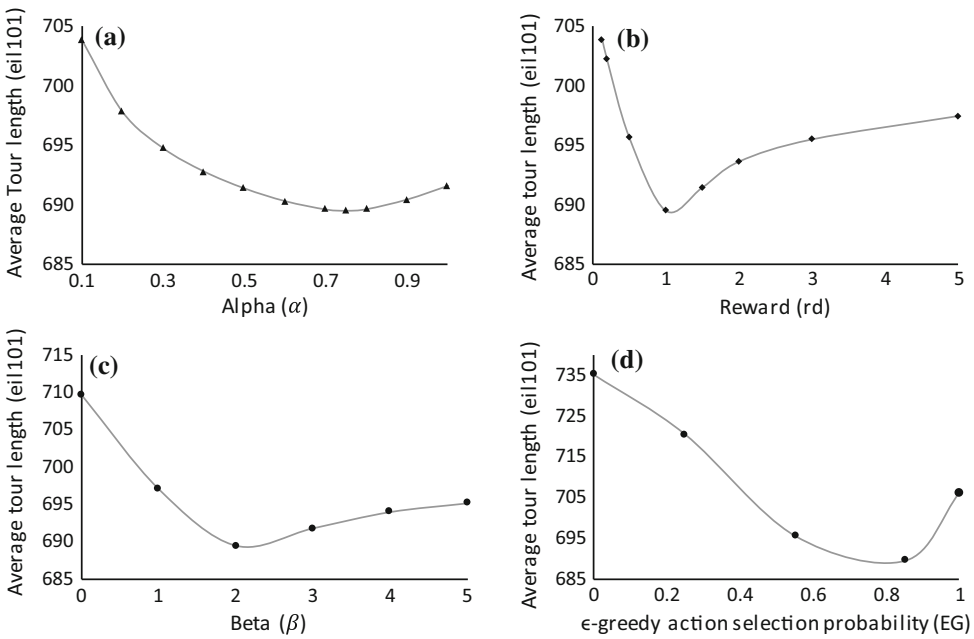


Fig. 6 Results of running MARL algorithm on dataset eil101 with different values of **a** the learning rates (α), **b** the rewards (rd), **c** the Beta (β) and **d** the $\in$-greedy action selection probability (EG)

Figure 6a shows the results obtained by MARL algorithm on different learning rates ($\alpha \in [0,1]$). A small value of α means the learning process will proceed at a low speed, while a too high value of α might affect the convergence property of the algorithm. Obviously, an appropriate α is very important to the performance of the whole algorithm. The results show that for a particular range of learning rates the algorithm performs better than other ranges and performance (average tour length) increases as α is increased to 0.75. But, when α is increased to more than 0.75, the performance reduced gradually. Therefore, we experimented in learning rate 0.75 in proposed algorithm.

In Fig. 6b, when reward (rd) is around 1, algorithm is showing satisfactory results. But, when rd is less than 1 and as much as approaches to zero, the performance of algorithm is reduced significantly. Also, increasing the value of rd to more than 1 causes somewhat negative performance impacts. Therefore, we experimented in reward 1 in proposed algorithm.

Beta (β) is a parameter which determines the relative importance of heuristic value $W^{-1}(i,j)$ ($W(i,j)$ is distance between cities i and j) versus the Q-values. If $\beta = 0$, only Q-value affects on $P(a|sc)$ (probability with which agent at city sc selects action a), and with increasing the value of β, we favor the choice of edges which are shorter. However, increasing the β to more than a certain value, we neglect the importance of Q-values gradually and this may cause negative effects on the learning process. According to Fig. 6c, when β is around 2, MARL algorithm shows satisfactory results. Therefore, we set the parameter β to 2.

In $\in$-greedy action selection method, a greedy action is selected most of the time (one of the learned optimal actions is selected greedily with respect to the Q-values) and—using a small probability—a random action is chosen once in a while. This ensures that after many learning

episodes, all the possible actions will be tried a high number of times, leading to an optimal policy. In contrast, Softmax action selection method differs from ∈-greedy in the way the random action is selected. A weight is assigned to each of the actions depending on their estimated values. A random action is selected based on the weight associated with it, ensuring that worst actions are unlikely to be chosen [53]. In proposed algorithm, each agent chooses one of its actions based on ∈-greedy with probability of EG% and Softmax with probability of SM% (equal to 1- EG), selection methods. As shown in Fig. 6d, when parameter EG is around 0.8 to 0.9, algorithm is showing satisfactory results. Therefore, we set the parameter EG to 0.85.

In order to see the effect of maximum number of learning iterations (MNLI) and maximum number of SMX iterations (MNSMX) to the convergence of the dataset eil101, let us see Fig. 7, which contains convergence graphs of the proposed algorithm on this dataset. In each graph, average running times in seconds are presented above the data label.

In Fig. 7a, we visualized the effect of different values for parameter MNLI. As we can see, after 1000-th iteration of proposed algorithm, the results have not generally shown notable improvements in average tour length proportional to the relatively high increase in running time of the algorithm. In other words, none of the runs have provided considerable convergence after the 1000-th iteration of learning according to intense growth of learning cycles. For this reason, at the rest of paper we set the parameter MNLI to 1000.

From Fig. 7b, we observe that the convergence rate of the proposed algorithm clearly reduces after iteration about 40000. In other words, none of the runs have not provided significant convergence after the 40000-th iteration of SMX $((100 \times \log_{10} n)^2)$. For this reason, at the rest of paper we set the parameter MNSMX to $(100 \times \log_{10} n)^2$ and after $(100 \times \log_{10} n)^2$-th cycle, working of the GA improvement heuristic will be stopped.

Table 2 summarizes the experiment results of different stages of our algorithm on 34 TSP datasets for TSPLIB, where the first column shows the name of the dataset and the optimal solution length taken from the TSPLIB into the parenthesis, and the column "Method (stage)" shows the results of different stages of proposed hybrid algorithm. For example, average solution found by MARL + NICH-LS algorithm, "MARL + NICH-LS" for dataset eil101, is 645.47, which takes 0.76 s and its PD_{avg} is 2.62. And average solution found by proposed hybrid algorithm, "GA-MARL + NICH-LS" (after applying GA improvement on candidate solutions taken from MARL + NICH-LS) on this dataset, is 642.6, which takes 1.08 s (total time needed to MARL + NICH-LS + GA) and its PD_{avg} is 2.16. The column "C.S.S./P. size" shows the average size of candidate solutions set and population, which, respectively, are produced by MARL + NICH-LS and GA. The column "best" shows the length of the best solution found by algorithm, the column "average" gives the average solution length of the each algorithm, the column "worst" shows the length of the worst, the column "CPU time(s)" shows the average running times in seconds for each algorithm, the column "PD_{best}" gives the percentage deviation of the best solution length over the optimal solution length and the column "PD_{avg}" denotes the percentage deviation of the average solution length over the optimal solution length. Percentage deviation of the best found solution to the best known solution PD_{best} and the percentage deviation of the average solution to the best known solution PD_{avg} and are defined as Eq. (2).

$$\begin{aligned} PD_{best} &= \frac{\text{Best solution} - \text{best known}}{\text{Best known}} \times 100 \\ PD_{avg} &= \frac{\text{Average solution} - \text{best known}}{\text{Best known}} \times 100 \end{aligned} \tag{2}$$

The average PD_{best} for MARL + NICH-LS and GA-MARL + NICH-LS are 0.99 and 0.65%, respectively. The average PD_{avg} for MARL + NICH-LS and GA-MARL + NICH-LS are 2.00 and 1.54%, respectively. The average running times for MARL + NICH-LS and GA-MARL + NICH-LS are 23.00 and 29.71, respectively. According to PD_{best} of the GA-MARL + NICH-LS, we can say that 85.29% of the values of PD_{best} are less than

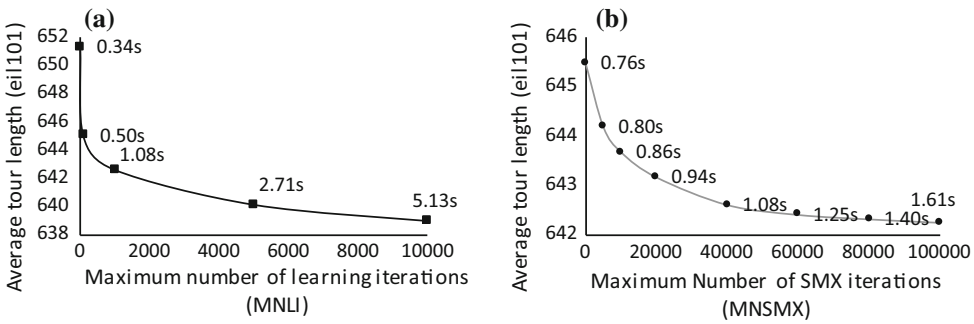


Fig. 7 Convergence of the proposed algorithm on dataset eil101 with different values of **a** MNLIs and **b** MNSMXs

Table 2 Computational results of proposed hybrid algorithm's stages for 34 TSP datasets

Dataset (optimal)	Method (stage)	C.S.S./P. size	Best	Average	Worst	CPU time(s)	PD_{best}	PD_{avg}
bayg29	MARL + NICH-LS	9.3	9073	9074.1	9094	0.16	0	0.01
(9073)	GA-MARL + NICH-LS	9.37	9073	9073.4	9077	0.22	0	0
bays29	MARL + NICH-LS	8.2	2020	2026.43	2041	0.16	0	0.32
(2020)	GA-MARL + NICH-LS	8.43	2020	2024.33	2033	0.22	0	0.21
att48	MARL + NICH-LS	9.2	33522	33,638.73	33,958	0.3	0	0.35
(33522)	GA-MARL + NICH-LS	9.83	33522	33,581.03	33,803	0.41	0	0.18
gr48	MARL + NICH-LS	9.3	5046	5102.13	5189	0.30	0	1.11
(5046)	GA-MARL + NICH-LS	10.2	5046	5069.33	5119	0.42	0	0.46
eil51	MARL + NICH-LS	8.5	426	428.67	437	0.31	0	0.63
(426)	GA-MARL + NICH-LS	9.47	426	427.4	432	0.44	0	0.33
berlin52	MARL + NICH-LS	9.2	7542	7557.4	7749	0.34	0	0.2
(7542)	GA-MARL + NICH-LS	9.3	7542	7550.7	7749	0.46	0	0.12
st70	MARL + NICH-LS	7.37	675	681.03	690	0.45	0	0.89
(675)	GA-MARL + NICH-LS	8.03	675	679.43	687	0.65	0	0.66
eil76	MARL + NICH-LS	8.02	540	548.23	555	0.51	0.37	1.9
(538)	GA-MARL + NICH-LS	9.04	538	545.3	550	0.73	0	1.36
pr76	MARL + NICH-LS	8.77	108,159	110271.63	112,392	0.52	0	1.95
(108159)	GA-MARL + NICH-LS	10.2	108,159	109,556.57	111,267	0.78	0	1.29
rat99	MARL + NICH-LS	8.7	1211	1233.57	1256	0.74	0	1.86
(1211)	GA-MARL + NICH-LS	10.83	1211	1223.3	1235	1.06	0	1.02
kroA100	MARL + NICH-LS	8.63	21,282	21,407.47	21,569	0.81	0	0.59
(21282)	GA-MARL + NICH-LS	10.23	21,282	21,354.4	21,498	1.14	0	0.34
kroB100	MARL + NICH-LS	9	22,141	22360.93	22,879	0.77	0	0.99
(22141)	GA-MARL + NICH-LS	9.87	22,141	22283.4	22,468	1.1	0	0.64
eil101	MARL + NICH-LS	9.07	629	645.47	655	0.76	0	2.62
(629)	GA-MARL + NICH-LS	10.37	629	642.6	653	1.08	0	2.16
lin105	MARL + NICH-LS	9.1	14,379	14,411.63	14,571	0.81	0	0.23
(14379)	GA-MARL + NICH-LS	10.5	14,379	14,385.63	14,479	1.13	0	0.05
pr107	MARL + NICH-LS	8.97	44,303	44,577.1	44,840	0.79	0	0.62
(44303)	GA-MARL + NICH-LS	10.23	44,303	44,424.73	44,570	1.1	0	0.27
pr124	MARL + NICH-LS	8.83	59,030	59,376.53	59,781	0.92	0	0.59
(59030)	GA-MARL + NICH-LS	10.77	59,030	59,208.83	59,602	1.34	0	0.3
bier127	MARL + NICH-LS	9.57	118,678	119,736.43	120,716	1.09	0.33	1.23
(118282)	GA-MARL + NICH-LS	11.03	118,678	119,437.27	120,336	1.53	0.33	0.98
ch130	MARL + NICH-LS	7.67	6150	6230	6344	1.07	0.65	1.96
(6110)	GA-MARL + NICH-LS	9.2	6132	6204.17	6357	1.45	0.36	1.54
ch150	MARL + NICH-LS	9.3	6543	6593.97	6708	1.27	0.23	1.01
(6528)	GA-MARL + NICH-LS	11.43	6528	6547.67	6640	1.84	0	0.71
kroA150	MARL + NICH-LS	8.93	26,737	27,050.4	27,366	1.21	0.8	1.98
(26524)	GA-MARL + NICH-LS	10.9	26,579	26,891.83	27,385	1.79	0.21	1.39
kroB150	MARL + NICH-LS	8.47	26,287	26,588.37	27,025	1.23	0.6	1.75
(26130)	GA-MARL + NICH-LS	11.63	26,130	26,477.33	26,986	1.84	0	1.33
kroA200	MARL + NICH-LS	8.77	29,506	29,810.67	30,368	2.03	0.47	1.51
(29368)	GA-MARL + NICH-LS	11.47	29,435	29,621	29,895	2.74	0.23	0.86
tsp225	MARL + NICH-LS	10.4	3895	3951.9	4037	2.11	0.88	2.35
(3861)	GA-MARL + NICH-LS	13.4	3865	3925.33	3992	3.13	0.1	1.67
pr226	MARL + NICH-LS	8.5	80,426	80,847.33	82,088	2.3	0.07	0.6
(80369)	GA-MARL + NICH-LS	13.6	80,369	80,638.6	82,031	3.29	0	0.34

Table 2 continued

Dataset (optimal)	Method (stage)	C.S.S./P. size	Best	Average	Worst	CPU time(s)	PD_{best}	PD_{avg}
a280	MARL + NICH-LS	8.67	2637	2684.53	2738	2.4	2.25	4.09
(2579)	GA-MARL + NICH-LS	11.33	2595	2655.47	2715	3.51	0.62	2.96
pr299	MARL + NICH-LS	9.1	48,990	49,610.83	50,427	3.23	1.66	2.95
(48191)	GA-MARL + NICH-LS	12.93	48,637	49,200.57	49,781	4.73	0.93	2.09
lin318	MARL + NICH-LS	7.87	42,760	43,244.07	43,778	3.53	1.74	2.89
(42029)	GA-MARL + NICH-LS	11.03	42,255	42,996.63	43,526	4.93	0.54	2.3
pr439	MARL + NICH-LS	8.47	109,116	110,559.37	113,336	5.63	1.77	3.12
(107217)	GA-MARL + NICH-LS	11.57	107,833	109,577.87	111,348	8.33	0.57	2.2
pr1002	MARL + NICH-LS	9.03	268,448	271,578.5	274,163	17.16	3.63	4.84
(259045)	GA-MARL + NICH-LS	10.73	266,886	269,845.97	273,595	24.28	3.03	4.17
rl1323	MARL + NICH-LS	9.57	279851	284,088.83	290,231	31.31	3.57	5.14
(270199)	GA-MARL + NICH-LS	12.33	279462	282,366.27	288,415	42.71	3.43	4.5
fl1400	MARL + NICH-LS	9.53	20,415	20,580.5	20,944	33.57	1.43	2.25
(20127)	GA-MARL + NICH-LS	15.57	20,304	20,444.33	20,944	47.34	0.88	1.58
pr2392	MARL + NICH-LS	10.1	397,388	400,995.13	404,628	83.4	5.12	6.07
(378032)	GA-MARL + NICH-LS	10.87	397,314	400,171.73	403,306	108.21	5.1	5.86
fl3795	MARL + NICH-LS	11.73	29,577	29,873.1	30,296	205.27	2.8	3.83
(28772)	GA-MARL + NICH-LS	21.93	29,191	29,609.67	30,048	262.95	1.46	2.91
Pla7397	MARL + NICH-LS	9.63	24,937,384	25,103,181.53	25,516,196	375.41	5.16	5.75
(23260728)	GA-MARL + NICH-LS	10.3	24,264,784	24,548,808.27	25,062,252	473.36	4.32	5.54
MARL + NICH-LS		Average				23.00	0.99	2.00
GA-MARL + NICH-LS						29.71	0.65	1.54

1%, which means that the best solution found, of the 20 trials, approximates less than 1% of the best known solution.

Figure 8 visualizes the PD_{best}, PD_{avg} and running times of these two algorithms according to Table 2. The results presented in Table 2 and Fig. 8 show that the accuracy of GA-MARL + NICH-LS is quite promising and it can provide good results in reasonable time for both small size and large size datasets, although the running times are a little greater than MARL + NICH-LS.

To compare the proposed algorithm (GA-MARL + NICH-LS) with counterpart algorithms [15, 16, 20, 22, 33–36, 38] in terms of CPU time, we scale the CPU time of each algorithm by an appropriate scaling coefficient related to their processing systems [22]. The processing system, programming language and their scaling coefficients are shown in Table 3.

We compare the experimental results of the our algorithm with nine state-of-the-art algorithms such as the genetic algorithm (GCGA) [33], the adaptive simulated annealing algorithm with greedy search (ASA-GS) [22], the self-organizing neural network (RABNET-TSP) [15], the memetic neural network (Memetic-SOM) [16], the genetic simulated annealing ant colony system with particle swarm optimization techniques (GSAP) [20], the *Q*-learning algorithm for initialization of the GRASP metaheuristic and genetic algorithm (GA-GRASP-Q-lrn) [36], the parallel hybrid implementation using genetic algorithm, GRASP and reinforcement learning (HPM) [38], the hybrid genetic algorithm with two local optimization strategies (HGA) [34] and the improved genetic algorithm with initial population strategy (KIP) [35], and results are shown in Table 4. The meanings of the columns in Table 4 are same as those in Table 2 (time is in second and scaled according to Table 3), and the best results are given in bold. Also, average PD_{best}, PD_{avg} and Time of each algorithm are given in italic (last column, Avg.). More intuitive comparisons are shown in Figs. 9, 10 and 11.

With respect to Table 4, Figs. 9 and 10, it can be seen that, when compared with the GCGA, the RABNET-TSP, the Memetic-SOM, the GA-GRASP-Q-lrn, the HPM and the KIP, our hybrid algorithm not only found the known optimal solution that others succeeded, but also found that the others failed. Also, the average of the results obtained

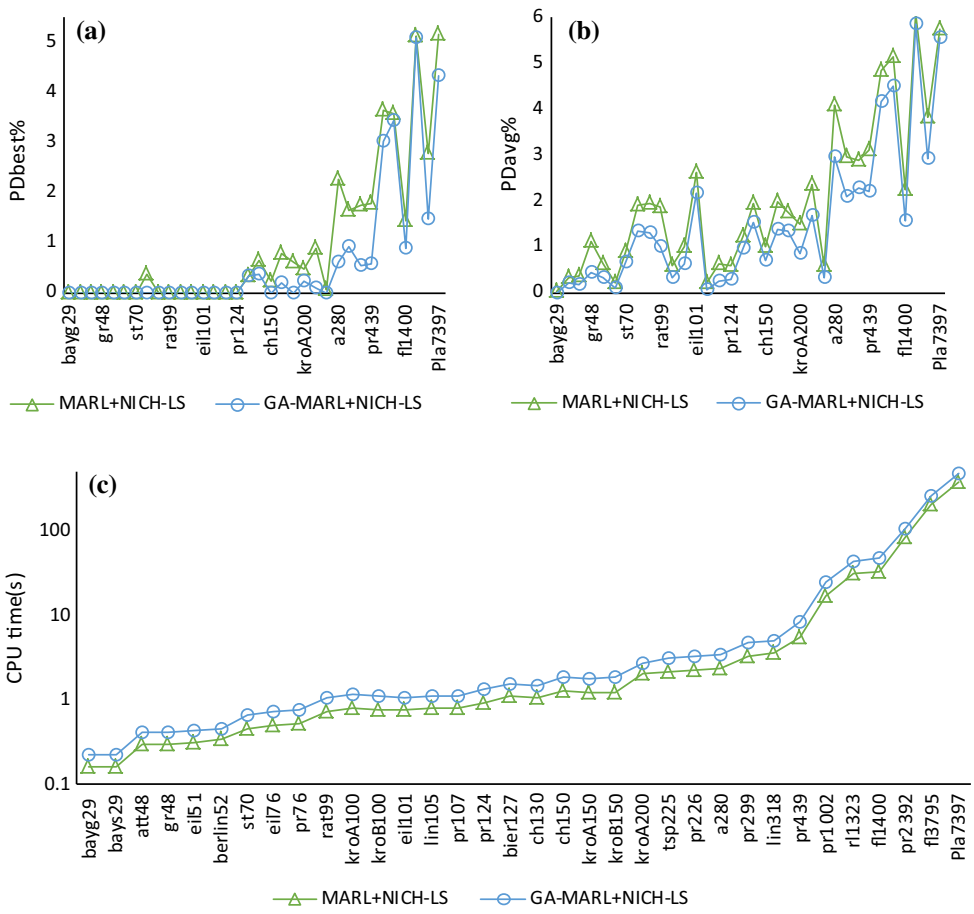


Fig. 8 Comparison graphically, between the MARL + NICH-LS and GA-MARL + NICH-LS based on **a** PD_{best}, **b** PD_{avg} and **c** CPU time(s)

Table 3 Scaling coefficients for adjusting the CPU time of the counterpart algorithms with respect to the CPU time of the GA-MARL + NICH-LS

Counterpart algorithms	Processing system	Programming language	Scaling coefficients
GCGA [17]	2.8 GHz	C++	0.99
ASA-GS [30]	2.8 GHz	C++	1
RABNET-TSP [31]	3.0 GHz	MATLAB	1.06
Memetic-SOM [32]	2.0 GHz	Java	0.71
GA-GRASP-Q-learning [36]	Pentium IV, 2.80 GHz	–	0.65
Hybrid parallel methods (HPM) [38]	Core 2 Duo, 2.33 GHz	–	0.9
Hybrid genetic algorithm (HGA) [34]	2.3 GHz	C ++	0.9
K-means initial population strategy (KIP) [35]	Core i3-2120, 3.30 GHz	MATLAB	1.3
Our algorithm (GA-MARL + NICH-LS)	Core-i5-4200U, 1.6 GHz	.Net 2013	1.4

by our algorithm in terms of PD_{best} and PD_{avg} is usually better than these algorithms. When compared with the ASA-GS, the GSAP and the HGA, for some instances the best solution and average of the results obtained by these algorithms are slightly better than our algorithm and for some instances the results obtained by our algorithm are slightly better. Comprehensively speaking, the performance of our algorithm is much better than the GCGA, the RABNET-TSP, the Memetic-SOM, the GA-GRASP-Q-lrn, the HPM and the KIP and somewhat equal to the ASA-GS, the GSAP and HGA in terms of PD_{best} and PD_{avg}.

Table 4 Results of proposed algorithm are compared with obtained results of nine state-of-the-art algorithms

Method	Dataset														
		bays29	gr48	eil51	berlin52	st70	eil76	pr76	kroA100	eil101	kroB150	ch150	a280	rl1323	fl1400	Avg.
GA-MARL + NICH-LS	PD_{best}	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0.62**	3.43	**0.88**	*0.35*
	PD_{avg}	**0.21**	**0.46**	0.33	0.12	0.66	1.36	1.29	0.34	2.16	1.33	0.71	**2.96**	4.5	**1.58**	*1.29*
	Time	**0.31**	**0.59**	**0.62**	**0.64**	**0.91**	**1.02**	**1.09**	**1.6**	**1.51**	**2.58**	**2.58**	**4.19**	**59.79**	**66.28**	*10.27*
GCGA	PD_{best}			0.23		**0**	2.23	0.14	0.5	1.59	1.63					*0.9*
	PD_{avg}			0.94		0.44	2.42	0.72	1.23	2.7	2.11					*1.51*
	Time			0.97		1.67	1.66	1.65	2.54	2.19	3.54					*2.03*
ASA-GS	PD_{best}			0.67	0.03	0.31	1.18	**0**	0.01	1.78	0.04	0.04		**0.65**	2.59	*0.66*
	PD_{avg}			0.67	0.03	**0.31**	1.18	**0**	**0.01**	1.83	**0.18**	0.18		**1.2**	3.25	*0.8*
	Time			3.91	3.83	5.15	5.50	5.49	7.14	7.42	10.90	10.91		210.16	232.02	*45.68*
RABNET-TSP	PD_{best}			0.24	**0**		0.56		0.24	1.43	0.51	1.13		11.31	3.6	*2.11*
	PD_{avg}			2.69	5.18		3.41		1.13	3.12	1.92	3.22		13.0	4.88	*4.28*
	Time			8.42	9.44		15.97		16.82	23.02	43.04	42.29		1003.10	1663.41	*313.95*
Memetic-SOM	PD_{best}			1.64	**0**	0.59	2.04		0.24	2.07	0.74	1.67			2.66	*1.29*
	PD_{avg}			2.14	2.01	0.99	2.88		1.14	3.15	1.61	2.95			4.86	*2.41*
	Time			1.78	2.14	2.52	2.74		3.74	3.63	5.49	5.56			222.35	*27.77*
GSAP	PD_{best}			0.23	**0**		**0**		**0**	0.16	**0**	0.51		2.75	2.32	*0.66*
	PD_{avg}			**0.3**	**0**		**0.41**		0.42	**0.99**	1.22	1.57		3.69	6.07	*1.63*
	Time															
GA-GRASP-Q-learning	PD_{best}	0.15	4.18									6.38				*3.57*
	PD_{avg}	10.35	18.89									31.11				*20.12*
	Time	73.01	225.32									1314.31				*537.55*
HPM	PD_{best}															
	PD_{avg}	1.18	11.04		12.67		23.13						114.48			*32.5*
	Time	79.2	765		874.8		1889.1						19,224			*4566.42*
HGA	PD_{best}			0.67	0.31	0.31	1.18	**0**	0.2	1.78	**0**	0.04	3.13			*0.76*
	PD_{avg}			0.75	0.31	0.35	1.50	0.09	0.14	2.52	0.79	**0.45**	3.77			*1.07*
	Time			28.31	30.13	77.51	111.15	99.70	269.39	489.84	2351.72	2279.11	11,940.96			*1767.78*
KIP	PD_{best}				**0**				0.10		3.35	5.10				*2.14*
	PD_{avg}				3.01				2.55		5.81	6.78				*4.54*
	Time				21.22				26.43		33.23	33.06				*28.49*

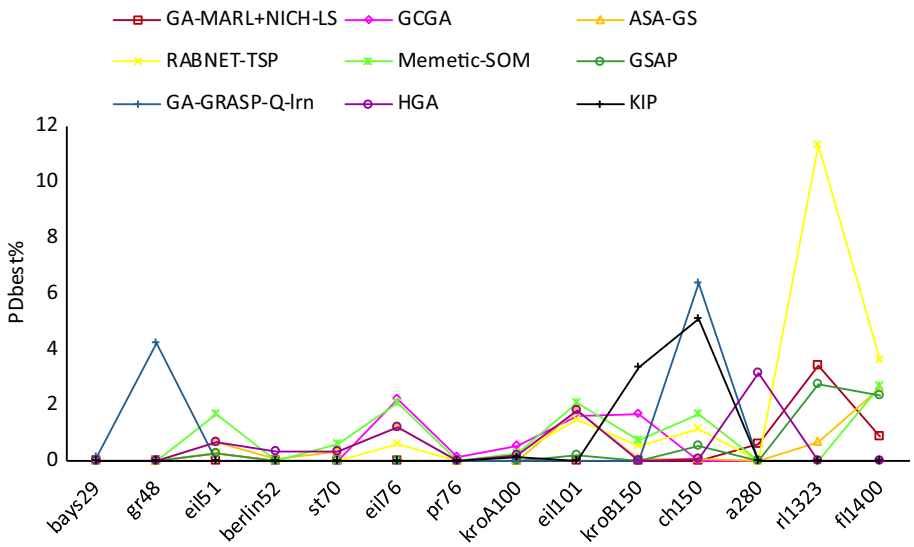


Fig. 9 Comparison graphically, between PD_{best} of GA-MARL + NICH-LS with eight algorithms

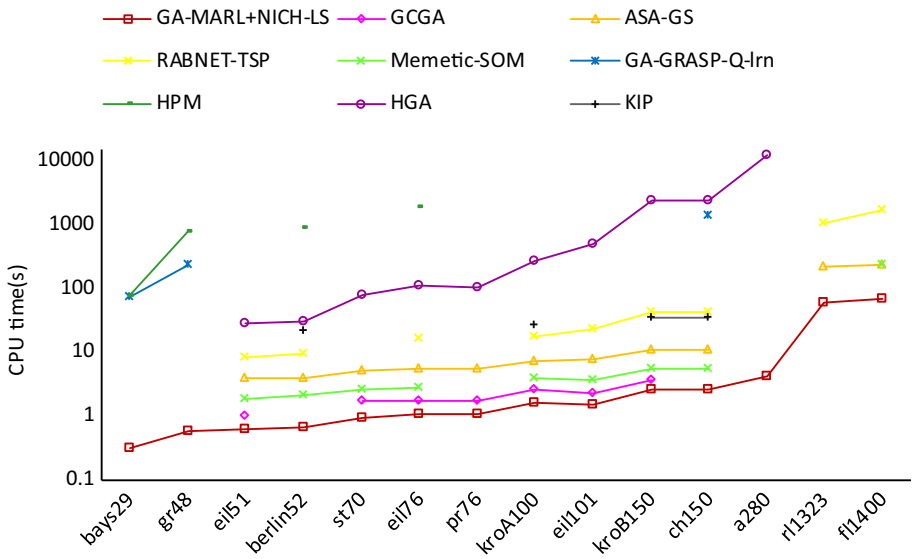


Fig. 10 Comparison graphically, between GA-MARL + NICH-LS with eight algorithms in terms of CPU time

Finally, we considered the average PD_{best}, PD_{avg} and running times of the proposed algorithm and counterpart algorithms on same datasets, as shown in Fig. 11. With respect to results, the average PD_{best} and running time of our algorithm on same datasets are less than of the all nine algorithms. Thus, our algorithm has higher performance than these approaches in terms of PD_{best}, and convergence rate of our algorithm is faster than of the all nine algorithms. Also, the average PD_{avg} of our algorithm on same datasets is less than of the GCGA, the RABNET-TSP, the Memetic-SOM, the GSAP, the GA-GRASP-Q-learning, the HPM and the KIP algorithms and somewhat more than of the ASA-GS and the HGA algorithms. Totally, we can say our approach is superior to most of nine state-of-the-art methods and the speed of computation of our algorithm is considerably fast.

6 Conclusion

In this study, a hybrid algorithm to determine the optimal solution for TSP is presented. The proposed method utilizes MARL approach as tour construction heuristic, which generates initial population of GA and GA with new crossover operator, SMX is applied as tour improvement heuristic.

Our algorithm is tested using 34 symmetric TSP datasets ranging from 29 to 7397 cities. The achieved results indicate that the proposed algorithm has good performance with respect to the quality of solution and the speed of computation and verify its validity. The GA-MARL + -NICH-LS performance is also compared with nine state-of-the-art algorithms, including the GCGA, the ASA-GS, the RABNET-TSP, the Memetic-SOM, the GSAP, the GA-

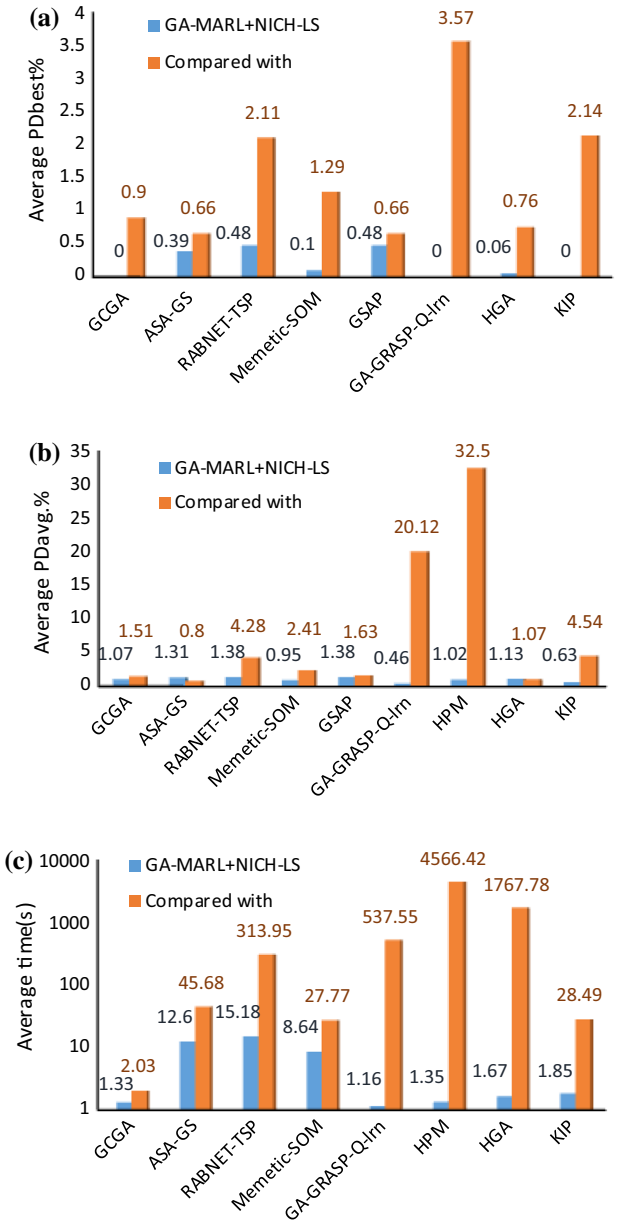


Fig. 11 Comparison of the proposed algorithm with each of nine algorithms based on **a** average PD_{best}, **b** average PD_{avg} and **c** average CPU time(s) on same datasets

GRASP-Q-lrn, the HPM, the HGA and the KIP. These state-of-the-art algorithms are outperformed by GA-MARL + NICH-LS in terms of accuracy and/or CPU time. It provided a better compromise between the CPU time and solution quality.

Compliance with ethical standards

Conflict of interest We (Author A, Mir Mohammad Alipour; Author B, Seyed Naser Razavi; Author C, Mohammad Reza Feizi Derakhshi and Author D, Mohammad Ali Balafar) wish to confirm that there are no known conflicts of interest associated with this article and there has been no significant financial support for this work that could have influenced its outcome.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Helsgaun K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur J Oper Res 126(1):106–130
2. Laporte G (1992) The traveling salesman problem: an overview of exact and approximate algorithms. Eur J Oper Res 59(2):231–247
3. Johnson DS, McGeoch LA (1997) The traveling salesman problem: a case study in local optimization. Local Search Comb Optim 1:215–310
4. Johnson DS, McGeoch LA (2007) Experimental analysis of heuristics for the STSP. In: The traveling salesman problem and its variations. Springer, US, pp 369–443
5. Lenstra JK (1997) Local search in combinatorial optimization. Princeton University Press, Princeton
6. Dorigo M, Gambardella LM (1997) Ant colonies for the traveling salesman problem. BioSystems 43(2):73–81
7. Dorigo M, Gambardella LM (1997) Ant colony system: a cooperative learning approach for the traveling salesman problem. IEEE Trans Evolut Comput 1(1):53–66
8. Dorigo M, Gambardella LM (2016) Ant-Q: a reinforcement learning approach to the traveling salesman problem. In: Proceedings of the twelfth international conference on machine learning (ML-95), pp 252–260
9. Gündüz M, Kiran MS, Özceylan E (2015) A hierarchic approach based on swarm intelligence to solve traveling salesman problem. Turk J Electr Eng Comput Sci 23(1):103–117
10. Dong GF, Guo WW, Tickle K (2012) Solving the traveling salesman problem using cooperative genetic ant systems. Expert Syst Appl 39(5):5006–5011
11. Yong W (2015) Hybrid max-min ant system with four vertices and three lines inequality for traveling salesman problem. Soft Comput 19(3):585–596
12. Haykin S, Network N (2004) A comprehensive foundation. Neural Netw 2:2004
13. Budinich M (1996) A self-organizing neural network for the traveling salesman problem that is competitive with simulated annealing. Neural Comput 8(2):416–424
14. Li R, Qiao J, Li W (2016) A modified hopfield neural network for solving TSP problem. In: 12th world congress on proceedings of the in intelligent control and automation (WCICA), 2016, IEEE
15. Masutti TA, de Castro LN (2009) A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem. Inf Sci 179(10):1454–1468
16. Créput JC, Koukam A (2009) A memetic neural network for the Euclidean traveling salesman problem. Neurocomputing 72(4):1250–1264
17. Thanh PD, Binh HTT, Lam BT (2015) New mechanism of combination crossover operators in genetic algorithm for solving the traveling salesman problem. In: Knowledge and systems engineering. Springer International Publishing, pp 367–379
18. Tsai CW, Tseng SP, Chiang MC, Yang CS, Hong TP (2014) A high-performance genetic algorithm: using traveling salesman problem as a case. Sci World J 2014:14, Article ID 178621. doi:10.1155/2014/178621
19. Sallabi OM, El-Haddad Y (2009) An improved genetic algorithm to solve the traveling salesman problem. World Acad Sci, Eng Technol 52(3):471–474

20. Chen SM, Chien CY (2011) Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. Expert Syst Appl 38(12):14439–14450
21. Malek M, Guruswamy M, Pandya M, Owens H (1989) Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. Ann Oper Res 21(1):59–84
22. Geng X, Chen Z, Yang W, Shi D, Zhao K (2011) Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. Appl Soft Comput 11(4):3680–3689
23. Lin Y, Bian Z, Liu X (2016) Developing a dynamic neighborhood structure for an adaptive hybrid simulated annealing–tabu search algorithm to solve the symmetrical traveling salesman problem. Appl Soft Comput 49:937–952
24. Zhan SH, Lin J, Zhang ZJ, Zhong YW (2016) List-based simulated annealing algorithm for traveling salesman problem. Comput Intell Neurosci 2016:8
25. Fiechter CN (1994) A parallel tabu search algorithm for large traveling salesman problems. Discret Appl Math 51(3):243–267
26. Misevičius A (2015) Using iterated tabu search for the traveling salesman problem. Inf Technol Control 32:3
27. Wong LP, Low MYH, Chong CS (2008) A bee colony optimization algorithm for traveling salesman problem. In: Proceedings of the second Asia international conference on modelling and simulation, IEEE
28. Meng L, Yin S, Hu X (2016) A new method used for traveling salesman problem based on discrete artificial bee colony algorithm. TELKOMNIKA Telecommun Comput Electron Control 14(1):342–348
29. Shi XH, Liang YC, Lee HP, Lu C, Wang QX (2007) Particle swarm optimization-based algorithms for TSP and generalized TSP. Inf Process Lett 103(5):169–176
30. Merz P, Freisleben B (1997) Genetic local search for the TSP: new results. In: IEEE international conference on proceedings of the in evolutionary computation, 1997, IEEE, pp 159–164
31. White CM, Yen GG (2004) A hybrid evolutionary algorithm for traveling salesman problem. In: Congress on proceedings of the in evolutionary computation, 2004. CEC2004, IEEE, vol 2, pp 1473–1478
32. Machado TR, Lopes HS (2005) A hybrid particle swarm optimization model for the traveling salesman problem. In: Adaptive and natural computing algorithms. Springer, Vienna, pp 255–258
33. Yang J, Wu C, Lee HP, Liang Y (2008) Solving traveling salesman problems using generalized chromosome genetic algorithm. Prog Nat Sci 18(7):887–892
34. Wang Y (2014) The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem. Comput Ind Eng 70:124–133
35. Deng Y, Liu Y, Zhou D (2015) An improved genetic algorithm with initial population strategy for symmetric TSP. Math Probl Eng 2015:212794
36. de Lima Junior FC, de Melo JD, Neto ADD (2007) Using q-learning algorithm for initialization of the grasp metaheuristic and genetic algorithm. In: Proceedings of the in 2007 international joint conference on neural networks, IEEE, pp 1243–1248
37. Liu F, Zeng G (2009) Study of genetic algorithm with reinforcement learning to solve the TSP. Expert Syst Appl 36(3):6995–7001
38. dos Santos JPQ, de Lima FC, Magalhaes RM, de Melo JD, Neto ADD (2009) A parallel hybrid implementation using genetic algorithm, GRASP and reinforcement learning. In: Proceedings of the In 2009 international joint conference on neural networks. IEEE, pp 2798–2803
39. Alipour MM, Razavi SN (2015) A new multiagent reinforcement learning algorithm to solve the symmetric traveling salesman problem. Multiagent Grid Syst 11(2):107–119
40. Jünger M, Reinelt G, Rinaldi G (1995) The traveling salesman problem. Handb Oper Res Manag Sci 7:225–330
41. Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, Ann Arbor
42. Goldberg DE, Lingle R (1985) Alleles, loci, and the traveling salesman problem. In: Proceedings of an international conference on genetic algorithms and their applications, vol 154. Lawrence Erlbaum, Hillsdale, pp 154–159
43. Davis L (1985) Applying adaptive algorithms to epistatic domains. In: Proceedings of the in IJCAI, vol. 85, pp 162–164
44. Oliver IM, Smith D, Holland JR (1987) Study of permutation crossover operators on the traveling salesman problem. In: Proceedings of the in genetic algorithms and their applications: proceedings of the second international conference on genetic algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA. Hillsdale, NJ: L. Erlhaum Associates, 1987
45. Whitley LD, Starkweather T, Fuquay DA (1989) Scheduling problems and traveling salesmen: the genetic edge recombination operator. In: Proceedings of the In ICGA, vol 89, pp 133–40
46. Kaya M (2011) The effects of two new crossover operators on genetic algorithm performance. Appl Soft Comput 11(1):881–890
47. Alipour MM, Razavi SN (2016) A novel local search heuristic based on nearest insertion into the convex hull for solving euclidean TSP. Int J Oper Res, (Under Publishing)
48. Russell S, Norvig P (2010) Artificial intelligence: a modern approach. Prentice Hall, Englewood Cliffs
49. Claus C, Boutilier C (1998) The dynamics of reinforcement learning in cooperative multiagent systems. In: Proceedings of the In AAAI/IAAI, pp 746–752)
50. Reinelt G TSPLIB is a library of sample benchmark instances for the TSP (and related problems) from various sources and of various types' [online] http://comopt.ifi.uniheidelberg.de/software/TSPLIB95/. Accessed Feb 2016
51. Tokic M, Schwenker F, Palm G (2013) Meta-learning of exploration and exploitation parameters with replacing eligibility traces. In: Proceedings of the In IAPR international workshop on partially supervised learning. Springer, Berlin, pp 68–79
52. Kobayashi K, Mizoue H, Kuremoto T, Obayashi M (2009) A meta-learning method based on temporal difference error. In: Proceedings of the in international conference on neural information processing. Springer, Berlin, pp 530–537
53. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT press, Cambridge