A Probabilistic Learning Algorithm for the Shortest Path Problem

Yiya Diao[1], Changhe Li[1(✉)], Yebin Ma[1], Junchen Wang[1], and Xingang Zhou[2]

[1] School of Automation and Hubei Key Laboratory of Advanced Control and Intelligent Automation for Complex Systems, China University of Geosciences, Wuhan, China
dwzxdjt@126.com, changhe.lw@gmail.com, benzalus@163.com, jenswang@outlook.com

[2] Communication Security Group of Eastern Theater Command, Nanjing, China
pcm603@163.com

Abstract. Ant colony optimization (ACO) has been shown effectiveness for solving combinatorial optimization problems. Inspired by the basic principles of ACO, this paper proposes a novel evolutionary algorithm, called probabilistic learning (PL). In the algorithm, a probability matrix is created based on weighted information of the population and a novel random search operator is proposed to adapt the PL to dynamic environments. The algorithm is tested on the shortest path problem (SPP) in both static and dynamic environments. Experimental results on a set of carefully designed 3D-problems show that the PL algorithm is effective for solve SPPs and outperforms several popular ACO variants.

Keywords: Probabilistic learning · Ant colony optimization · Dynamic shortest path problems

1 Introduction

The shortest path problem (SPP) is a classical combinatorial optimization problem. In this paper, the problem is defined as follows. Given a graph of a set of nodes linked by directed edges, where each node is connected with eight neighbour nodes, the aim is to find the shortest path from a starting point to a destination point. It is a simple problem and has been widely applied in many applications, e.g., robot routing problem, the shortest path in games, the shortest traffic route, etc.

The Dijkstra algorithm is widely used to find the shortest path in a completely known environment. However, when the environment is partially known, unknown or changing, the Dijkstra algorithm seems unsuitable to solve this problem. To overcome these limitations, many heuristic approaches have been used to address this problem, such as ant colony optimization (ACO).

Inspired by ACO, this paper proposes a novel probabilistic algorithm, namely probabilistic learning (PL), for the SPP. A probabilistic matrix is created and

Y. Shi et al. (Eds.): SEAL 2017, LNCS 10593, pp. 631–643, 2017.
https://doi.org/10.1007/978-3-319-68759-9_51

updated according to weighted values of historical solutions. Each element in the matrix denotes the probability of an edge to be chosen. Different from ACO, PL neither follows the basic operations of ACO nor uses any geographical heuristic. An efficient random search operator is proposed to maintain the population diversity. The proposed algorithm is competitive in comparison with several ACO algorithms on the SPP.

The rest of this paper is organized as follows. Section 2 gives a brief review of related work. Section 3 describes the proposed algorithm in detail. Section 4 presents experimental results and discussions. Finally, conclusions and future work are discussed in Sect. 5.

2 Related Work

There are many research branches for combinatorial optimization problems. However, in this section, we mainly focus on the methods related to ACO. ACO simulates the foraging behavior of real ants for finding the shortest path from a food source to their nest by exploiting pheromone information. While walking, ants deposit pheromone chemicals, and follow, in probability, pheromone previously deposited by other ants.

Ant system (AS) [8] was the first ACO model proposed by Dorigo, and it has gained huge success. In order to simulate the foraging behavior, each ant chooses its next node by:

$$p_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in U}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}} & j \in U \\ 0 & j \notin U, \end{cases} \tag{1}$$

where $p_{ij}^{k}(t)$ means the possibility of ant k moves from node i to node j. If ant k has visited node j, $p_{ij}^{k}(t)$ will be 0; otherwise, the ant will probabilistically choose a candidate node. τ_{ij} means the pheromone intensity of edge (i,j). η_{ij} means the visibility of edge (i,j), which equals to the reciprocal of the length of this edge. α is the weight of pheromone and β is the weight of the visibility. After all ants reach the food point, the pheromone matrix is updated by:

$$\tau_{ij}(t+1) = \rho * \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t), \tag{2}$$

where m is the number of ants, and ρ means the persistence of pheromone. After this update, all ants go back to the start point and repeat the above procedures until the algorithm stops.

AS has a strong ability in searching good solutions. However, it has shortcomings: the slow convergence speed and the stagnation issue. Inspired by the selection principle of genetic algorithms, an elitist AS (EAS) [2] was developed. Compared to AS, EAS takes an ant which performs best as an "elitist", and it leaves more pheromone than other ants. As a result, ants are more likely to choose the elitist path.

The ant colony system (ACS) [7] was proposed based on Ant-Q (AS with Q-learning) [6]. ACS provides two ways to update global pheromone: the local online update and the global offline update. The local online update is used after every ant moves a step, which means the pheromone of the map changes synchronously. On the other hand, the global offline update is used when all ants reach the food point, then the best ant is chosen to update the global pheromone. ACS is quick and it can obtain good solutions, but it is not stable.

A rank-based AS (RAS) [1] was proposed by sorting ants based on route lengths. Like EAS, ants with high rank leave more pheromone than the others as follows. RAS is more likely to find a better solution than EAS, but its convergence speed is slower than EAS. A min-max AS (MMAS) [12] was proposed. The amount of the pheromone of each edge is constrained within a range. Also, like EAS, only the ant with best performance can leave pheromone on its route. The rules of the node selection and pheromone update are the same as in EAS, except that it keeps the pheromone level in a dynamic range. A best-worst AS (BWAS) [3] was proposed where it rewards the best ant and punishes the worst one. In each generation, BWAS increases the pheromone level on the best ant's route and decreases the pheromone level of the worst route.

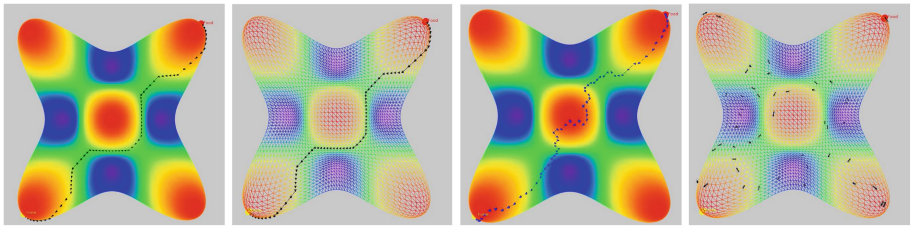
Recently, many other improvements for ACO have been proposed, such as ant colony system with a cooperative learning approach [7], a hybrid method combining ACO with beam search (Beam-ACO) [9], parallel ant colony optimization (PQACO) [15], a hybrid PS-ACO algorithm with the hybridization of the PSO [11], a novel two-stage hybrid swarm intelligence optimization algorithm [4], cooperative genetic ant systems [5], ACO with new fast opposite gradient search [10], advanced harmony search with ACO [16], and so on.

3 The Proposed Method

This section will introduce the PL algorithm in detail. Before the introduction of PL, we first describe the problem to be solved in this paper in a game scenario.

3.1 The Map for the SPP

Figure 1-a shows a game map, which is discretized into a grid shown in Fig. 1-b where each cross point in the grid is represented by a two dimensional integer coordinate (i, j). The color of the point stands for its height from the highest (red color) to the lowest (purple color). The yellow point at the left bottom corner is the home point, and the red point at the right top corner is the food point. The dotted path obtained is the shortest path from the home point to the food point. Figure 1-c shows a blue path, which constructed by an ant. Figure 1-d shows 3D ants moving on the map in the game, where an ant can only be placed on a cross point. Note that, an ant is only allowed to visit its 8-neighborhood points of the point where it is.



a b c d

Fig. 1. An example of a 3D problem, where the dotted path is the shortest path from the home point to the food source labelled by the yellow point at the left bottom corner and the red point at the right top corner, respectively. (Color figure online)

3.2 Probabilistic Learning

In this paper, a population consists of a set of solutions constructed by ants from the home point to the food point. This paper introduces a probability matrix (PM) to learn promising edges found by ants. The idea of the PM was initially proposed in [13] and recently updated in [14] for solving the travelling sales problem. Each item in the PM denotes the probability of each edge to be chosen during the construction of a path. In this paper, we build the PM using a different strategy used in [14] to adapt it to the SPP.

In the SPP, each ant can only visit its 8-neighborhood points and the ant is allowed to revisit a point. A randomly constructed path may contain many loops. We need to design a method to avoid these loops to obtain an effective path.

The Probabilistic Matrix. An element in the PM indicates the frequency of an edge visited by ants. Initially, ants randomly construct their paths since there is no information in the PM. Each ant keeps its historical best path so far and update it once the ant finds a better path. Correspondingly, the PM will be also updated.

The PM in fact reflects the learning process as the evolutionary process goes on: (1) The number of good edges in a particular solution increases; (2) The frequency of appearance of a particular good edge increases in the population.

To implement the above idea, we assume that the shorter the path, the better the solution. So we use the length of a solution to evaluate its fitness. The fitness of a solution is obtained by:

$$F_i = \frac{L^{max} - L^i + 1}{L^{max} - L^{min} + 1} \tag{3}$$

where F_i is the fitness of the ith solution, L^{max} and L^{min} are the maximum and the minimum length of all the solutions, respectively. We assign each ant i a weight W_i based on its historical best solution as follow:

$$W_i = \frac{1}{1 + \mathrm{e}^{-F_i}}, \tag{4}$$

Algorithm 1. Probabilistic Learning Algorithm

```
Initialize a population X=[x_1,x_2,...,x_PS] with PS ants;
Initialize an archive population A;
Initialize a probability matrix PM with element values of zero;
while (termination criteria not satisfied) do
    for (each ant x_i) do
        if rand() ≤ ρ then
            Construct a new solution x'_i by Algorithm 2;
        else
            Construct a new solution x'_i by PM;
        end if
        if x_i gets better then
            Update x_i's personal best solution;
            Update DM_i by Algorithm 3;
        end if
    end for
    Update PM based on the archive population A;
end while
```

The higher value of the fitness, the higher value of the weight. Finally, we compute each element p_{ij} of the matrix as follows:

$$p_{ij} = \sum_{k=1}^{PS} W_k * o_{ij} / \sum_{k=1}^{PS} W_k, o_{ij} = \begin{cases} 1, edge(i,j) \in best\_path^k \\ 0, else \end{cases} \tag{5}$$

where p_{ij} means the possibility of each ant moves from node i to node j, PS is the population size and $best\_route^k$ is the historical best path of ant k.

3.3 Probabilistic Learning Algorithm

The framework of the PL algorithm is presented in Algorithm 1. An archive population is utilized to store the historical best solution of each ant. For each generation, a solution is constructed from the home point to the food point either by the PM or by Algorithm 2 (introduced later) depending on a probability ρ. After all ants obtain a valid path, the PM will be updated according to each ant's historical best solution.

Random Search Operator. Although an ant is able to construct a solution based on the PM, it cannot produce new edges. So we need to find an effective random search operator to maintain the population diversity. The aim is also to adapt PL to dynamic environments by using the diversity maintaining schemes as follows.

In this paper, we design two rules for an ant to efficiently construct a random path from the start point to the end point: (1) Each ant always head to the end point, which helps it avoid loops on its path. To achieve this, we suppose that each ant know the position of the end point; (2) The ant walks toward a fix direction for several steps unless it comes across a visited node, where a step means an ant moves to one of its 8-neighborhood points. A random direction is chosen from a set of neighbourhood nodes if $\overrightarrow{N_cN_e} \cdot \overrightarrow{N_cN_d} \geq 0$, where N_c, N_e,

Algorithm 2. Random search operator

```
Create an archive Cur to store the current position of the ant, which is initially the start point.
Create an archive Way to store the path constructed by this algorithm.
Add Cur to Way.
while Cur is not the end point do
    Randomly select an integer Len between 1 and the maximum number of the height and the
width of the map.
    Randomly select a feasible direction Dir which heads to the end point.
    for i = 1 to Len do
        if next point is not feasible or C is the end point then
            exit from the loop
        end if
        Update Cur by Dir.
        Add Cur to Way.
    end for
end while
```

Algorithm 3. Update Local Distance Matrix

```
Create an archive W based on the path constructed by Algorithm 2.
Create an archive D to remember the distance and initialize it to zero.
Create an archive C to remember the current position of the ant and initialize it to the end
point.
while W is not empty do
    get the last point P from W.
    remove P from W.
    add the distance between C and P to D
    if D ≤ DM[P]) then                    ▷ DM[P] is the distance between P and the end point
        DM[P] ← D
        PreM[P] ← C                                  ▷ P is the next node of node C
    else
        D ← DM[P]
    end if
    C ← P
end while
```

and N_d are positions of the current, ending, and next nodes, respective. An ant sometime might not be able to move forward since all its neighborhood points have been visited. In this case, we allow the ant to reconstruct its path.

Using these rules, this paper introduces a new random search operator to construct an effective path. Algorithm 2 presents the procedures for constructing a new solution. It works as follows. Each ant re-starts from the start point. It randomly select a feasible direction which heads to the end point and selects a random step between 1 and the maximum number between the height and the width of a map. It will change its direction if the next point has been visited. The process is repeated until it reaches the end point.

Local Distance Matrix. Although we can construct an effective path by Algorithm 2, the path may not always get improved since it is randomly created. To fully use the information we get during the random searching process, this paper introduces a matrix for each ant to remember the current shortest distance from each point in the map to the end point, and we will update the matrix DM whenever a shorter path is found.

Adaptation of Parameter ρ. In the beginning of the search, all ants randomly construct solutions, and learning too much from the PM may not benefit the search. As the run goes on, the population will improve as the number of good edges increases, and it will be helpful to increase the probability of learning from the PM. In this paper, we adaptively adjust ρ by the equation below:

$$\rho = \frac{S}{T} * \theta \tag{6}$$

where S is the total number of the points of the best paths found so far by all ants and T is the total number of points which appear in all the historical best paths found by all ants. θ is used to set the lowest possibility of random search.

4 Experimental Studies

Performance comparison between the proposed algorithm and several ACO variants is conducted on a set of 3D problems in this section.

4.1 Test Problems

To test the performance of an algorithm in a game scenario, we have carefully designed a set of 3D maps to simulate real-world environments. Figure 2 shows 15 test problems. We divide these test problems into two groups: asymmetrical and symmetrical. The difficulty of a problem mainly depends on the number of curve segments and the degree of the curvature of curve segments on the shortest path. The asymmetrical problems P00 and P01 have only one global optimum. P01 is more difficult than P00 due to the steep terrain. The symmetrical problems (P03-P14) have more than one global optimum because of its symmetrical characteristics, and are more difficult than the problem in the fist group. These maps are defined in Table 1, where H and W are the height and width of a map, respectively.

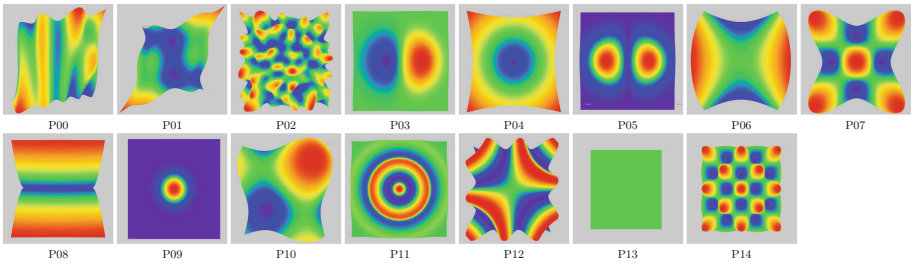


Fig. 2. The 3D maps for all the problems

Table 1. Test problems, where H and W are the height and width, respectively.

Problem	Description	Range
P00	$\frac{1}{10}(W+H)\left(3+3.5xy^3-4.7cos(3x-(2+x)t)sin(2.5\pi x)\right)$	$-0.9 \le x \le 1.2,\ -1.2 \le y \le 1.2$
P01	$\frac{1}{2}(W+H)\left(\left(4-2.1x^2+\frac{1}{3}x^4\right)x^x+xy+\left(-4+4y^2\right)y^2\right)$	$-1.9 \le x \le 1.9,\ -1.1 \le y \le 1.1$
P02	$rand() * (W+H)$	$-1 \le rand() \le 1$
P03	$2*(W+H)\,x*\exp\left(x^2-y^2\right)$	$-2 \le x \le 2,\ -2 \le y \le 2$
P04	$\frac{1}{4}(W+H)\sqrt{x^2+y^2}$	$-5 \le x \le 5,\ -5 \le y \le 5$
P05	$2(W+H)\lvert x\rvert \exp\left(-x^2-\frac{4}{3}y^2\right)$	$-2 \le x \le 2,\ -2 \le y \le 2$
P06	$\frac{1}{4}(W+H)\left(x^2-y^2\right)$	$-2 \le x \le 2,\ -2 \le y \le 2$
P07	$(W+H)\cos x \cos y$	$-4 \le x \le 4,\ -4 \le y \le 4$
P08	$(W+H)\sqrt{\lvert y-0.01x^2\rvert+0.01*\lvert x+10\rvert}$	$-2 \le x \le 2,\ -2 \le y \le 2$
P09	$(W+H)\exp(x^x-y^y)$	$-4 \le x \le 4,\ -4 \le y \le 4$
P10	$\frac{1}{2}(W+H)\sin x \sin y$	$-3 \le x \le 3,\ -3 \le y \le 3$
P11	$(W+H)*\left(0.5+\frac{\left(0.5-sin\left(\sqrt{0.0001+x^2+y^2}\right)^2\right)}{\left(1+0.001(x^2+y^2)*(x^2+y^2)\right)^2}\right)$	$-6 \le x \le 6,\ -6 \le y \le 6$
P12	$\frac{1}{2}(W+H)\sin(xy)$	$-3 \le x \le 3,\ -3 \le y \le 3$
P13	0	-
P14	$\frac{3}{4}(W+H)\cos x \cos y$	$-8 \le x \le 8,\ -8 \le y \le 8$

4.2 Parameter Settings

We set the parameters of ACO variants based on the suggestions of their authors for TSPs. Parameter settings of involved algorithms are as follows. (1) AS: $\alpha = 1.0$, $\beta = 5.0$, $\rho = 0.5$, $Q = 100$, $\tau_{ij}(0) = \frac{2}{W+H}$; (2) ACS: $\alpha = 1.0$, $\beta = 2.0$, $\rho = 0.1$, $Q = 0.9$, $\tau_{ij}(0) = \frac{2}{(W+H)*L^{hunger}}$, L^{hunger} is the length of path constructed by hunger strategy, where we construct a path always by choosing the shortest edges; (3) MMAS: $\alpha = 1.0$, $\beta = 2.0$,$\rho = 0.02$, $length = 20$ (the length of the candidate list), $\lambda = 0.05$, $\tau^0_{max} = \frac{1}{\rho * L^{hunger}}$, $\tau^0_{min} = \frac{\tau^0_{max}}{4.0*(W+H)}$where τ_{max0} and τ_{min0} is the initially min pheromone and the max pheromone respectively. For each iteration, $\tau^t_{max} = \frac{1}{\rho * L_{best}}$, where L_{best} is the length of global best solution, $\tau^t_{min} = \frac{\tau^t_{max}*(1-\exp(\frac{\log(0.05)}{n}))*2}{\exp(\frac{\log(0.05)}{n})*(length+1)}$, where n is the number of points in the global best solution.

All algorithms terminate when the number of iterations is greater than I_{max} or the population meets $L_{worst} - L_{best} \le$1e-5 L_{best}, Len_{best} and Len_{worst} are the current best and worst solutions, respectively. The relative error (RE), which is used to evaluate the performance of an algorithm, is defined as follows.

$$RE = (Dis(x_{best}) - Dis(x^*))/Dis(x^*) \tag{7}$$

where x_{best} and x^* are the best solution found by an algorithm and the global optimum, respectively. All results are averaged over 30 independent runs in this paper. To test the statistical significance between the results of algorithms, the Wilcoxon rank sum test is performed at the significance level $\alpha = 0.05$ in this paper.

4.3 Experimental Results

Minimum Probability. In the PM, the possibility of an edge to be chosen is zero if it does not appear in any solution. To improve its global search ability, we set a lowest possibility of each edge to be selected.

In this test, we set W = H = 50, $I_{max} = 1000$, and $PS = 500$. If the possibility of an edge to be chosen is zero, then we use a minimum possibility $\lambda * W_{max}$, where W_{max} is the weight of the best so far solution and λ is a parameter to be tested. Figure 3 presents the effect of varying λ on all the problems.

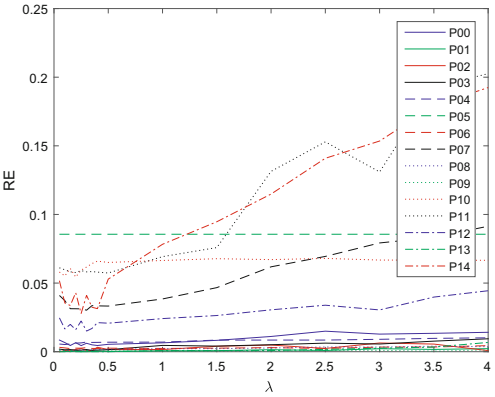


Fig. 3. The effect of varying λ.

The experiment shows that as the value of λ increases, the relative error also increases. The ants in PL will be more likely to perform random search as λ increase since all neighbour nodes have similar possibilities. For example, when the λ is small, e.g., between 0 and 4, its performance stays at almost the same level. In this paper, we set $\lambda = 0.2$ to make sure PL can achieve a good result in a relatively short time on most problems.

Performance Comparison. In this subsection, to compare the global search capability with AS, ACS, and MMAS, we only use the probability matrix for PL. In this test, we set $W = H = 50$, $PS = 500$, and $I_{max} = 1000$. Table 2 shows the performance comparison of all the algorithms, where Worst, Best, and Mean are the worst, the best, and the mean of RE over all runs. The Wilcoxons rank sum test is performed among algorithms on the best mean results on each problem, Y and N denote that the mean results of the best algorithm are significantly better than and statistically equivalent to other algorithms, respectively.

The Results show that PL outperforms ACS, AS, and MMAS on most problems. MMAS performs best on P07, P12, and P14 and ACS on P10 and P11. Although PL performs best on most problems, it could be improved in comparison with MMAS on hard problems P07, P12, and P14.

Table 2. Performance comparison of all the algorithms on SPP.

Algorithm	Problem	Mean	STD	T	Problem	Mean	STD	T	Problem	Mean	STD	T
ACS	P00	6.28e−2	1.41e−2	Y	P01	1.27e−1	1.77e−2	Y	P02	9.59e−2	1.04e−2	Y
AS		4.03e−1	2.78e−2	Y		6.06e−1	3.25e−2	Y		3.82e−2	1.75e−2	Y
MMAS		1.12e−2	4.70e−3	Y		5.02e−3	5.62e−3	Y		1.95e−2	7.51e−3	Y
PL		**6.36e−3**	3.54e−3	N		**5.24e-4**	1.21e−3	N		**1.59e−3**	3.18e−3	N
ACS	P03	4.60e−2	1.20e−2	Y	P04	3.40e−2	9.86e−3	Y	P05	1.19e−1	1.61e−2	Y
AS		2.43e−1	4.92e−2	Y		2.26e−1	2.52e−2	Y		3.38e−1	8.24e−2	Y
MMAS		6.10e−3	2.98e−3	Y		7.27e−3	8.14e−4	Y		8.69e−2	2.10e−3	Y
PL		**1.09e−3**	6.84e−4	N		**5.37e−3**	1.51e−3	N		**8.56e−2**	0.00e−0	N
ACS	P06	9.21e−2	3.00e−2	Y	P07	2.82e−2	6.39e−3	Y	P08	4.53e−2	1.60e−2	Y
AS		3.63e−1	3.27e−2	Y		2.10e−1	5.40e−2	Y		2.75e−1	3.20e−2	Y
MMAS		2.22e−2	5.83e−3	Y		**1.12e−2**	6.88e−3	Y		8.98e−3	6.18e−3	Y
PL		**2.54e−3**	1.40e−3	N		3.03e−2	6.99e−3	N		**1.93e−3**	1.43e−3	N
ACS	P09	3.96e−2	1.44e−2	Y	P10	**4.92e−2**	1.44e−2	Y	P11	**2.89e−2**	8.96e−3	Y
AS		2.82e−1	3.08e−2	Y		2.98e−1	1.95e−2	Y		2.82e−1	6.41e−2	Y
MMAS		2.02e−3	1.02e−3	Y		5.50e−2	1.19e−2	Y		6.49e−2	3.92e−3	Y
PL		**4.32e−5**	1.24e−4	N		5.91e−2	9.65e−3	N		5.98e−2	3.88e−3	N
ACS	P12	1.59e−2	5.42e−3	N	P13	9.55e−2	1.20e−2	Y	P14	7.08e−3	7.51e−3	Y
AS		2.76e−1	1.15e−1	Y		3.86e−1	1.65e−2	Y		1.37e−1	7.00e−2	Y
MMAS		**4.95e−3**	2.95e−3	Y		1.52e−2	6.33e−3	Y		**5.71e−4**	1.72e−3	Y
PL		1.24e−2	7.34e−3	N		**0.00e−0**	0.00e−0	N		4.41e−2	2.10e−2	N

The Possibility to Learn from PM. The PM helps PL quickly find a solution, which often is a local optimum. The random search operator has a strong global search capability but it takes a long time to converge. In this subsection, we test the sensitivity of parameter θ. Figure 4 presents the effect on varying θ on the performance of PL regarding the RE and the average number of iterations which PL takes. In this test, we set $\mathrm{W} = \mathrm{H} = 500$, $\mathrm{PS} = 50$, and $I_{max} = 20000$. The results show that the PL takes less number of iterations to converge as θ increases, but it more likely get stuck at local optima, and vice versa for decreasing θ. From the results, the random search operator does help PL to improve the RE at the price of increasing the number of iterations. From the results, we suggest to take $\theta = 0.2$.

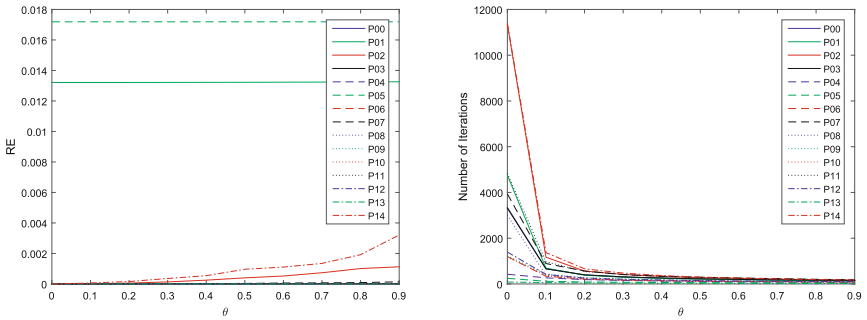


Fig. 4. The effect of varying parameter θ.

Experiment on Dynamic SPP. In this subsection, we carry out experiments on dynamic SPP, where the food point changes to a random location on the map every a certain number of iterations I_g. Given the knowledge that a large value of θ helps PL achieve a fast convergence speed, we set $\theta = 0.9$. The random search operator is enabled. In this test, we set $W = H = 500$ and $PS = 50$. The change interval I_g varies from 5 to 100.

Table 3 shows the results of PL on all problems with different change frequencies. From the results, we can have a common observation: the RE gets better as the change interval increases. It is reasonable since a greater change interval means a longer time for PL to search. PL achieves a good performance even in frequently changing environments. Take $I_g = 5$ for example, where the food point changes every five iterations, PL is able to achieve a small RE on most problems, e.g., $RE < 1e{-}2$. Thanks to the random search operator, the PL is able to maintain the population diversity during the runtime. On the other hand, the

Table 3. Results of PL on dynamic SPP with different change frequencies.

I_g	Problem	Mean	STD	Problem	Mean	STD	Problem	Mean	STD
5	P00	4.87e−2	1.57e−2	P01	2.96e−2	1.02e−2	P02	7.02e−1	4.21e−2
10		1.04e−2	2.82e−3		1.24e−2	3.95e−3		2.53e−1	1.50e−2
20		2.49e−3	1.14e−3		7.80e−4	2.46e−4		1.37e−1	1.42e−2
50		2.90e−4	6.88e−5		1.20e−3	5.59e−4		9.65e−3	1.63e−3
100		1.41e−4	2.71e−5		1.13e−4	2.47e−5		5.40e−3	1.26e−3
5	P03	3.96e−3	1.45e−3	P04	1.18e−4	2.83e−5	P05	1.52e−2	4.94e−3
10		1.51e−3	4.00e−4		1.02e−4	2.80e−5		8.58e−3	3.76e−3
20		1.49e−4	1.78e−5		9.43e−5	6.16e−8		3.32e−4	8.65e−5
50		1.18e−4	1.56e−5		1.06e−4	6.70e−7		8.62e−3	6.55e−8
100		1.10e−4	4.58e−6		8.62e−5	6.55e−8		9.63e−5	2.81e−7
5	P06	3.64e−3	6.45e−3	P07	1.31e−1	1.85e−2	P08	2.24e−3	1.41e−3
10		3.70e−4	6.70e−4		3.97e−2	5.17e−3		1.01e−4	1.48e−5
20		1.08e−4	3.39e−5		4.01e−3	7.14e−4		1.11e−4	1.37e−5
50		9.03e−5	2.27e−7		3.77e−3	1.52e−3		8.87e−5	5.33e−6
100		1.05e−4	6.41e−6		9.00e−4	3.52e−4		8.03e−5	2.52e−9
5	P09	6.21e−4	1.74e−4	P10	7.65e−4	6.33e−4	P11	2.82e−2	4.30e−3
10		4.13e−4	1.28e−4		1.07e−4	1.86e−5		6.65e−3	7.87e−4
20		1.75e−4	4.17e−5		9.69e−5	6.62e−6		1.79e−3	4.45e−4
50		1.26e−4	2.31e−5		9.21e−5	2.51e−6		2.23e−4	4.33e−5
100		1.13e−4	7.78e−6		9.32e−5	2.30e−6		1.94e−4	3.91e−5
5	P12	1.14e−2	2.66e−3	P13	1.11e−4	1.56e−15	P14	6.64e−1	3.93e−2
10		3.98e−3	1.33e−3		9.55e−5	1.21e−15		2.58e−3	2.26e−2
20		3.78e−4	8.73e−5		9.99e−5	1.17e−15		1.32e−1	1.49e−2
50		3.40e−4	1.19e−4		1.01e−4	1.11e−15		3.34e−2	4.18e−3
100		8.96e−5	1.16e−5		1.02e−4	1.09e−15		8.83e−3	3.44e−3

PM helps PL quickly find a reasonable solution. These two components make PL easily adapt to dynamic environments without using extra dynamism handling techniques.

Compared with ACOs, PL has only two parameters (λ and θ) to set and it outperforms the ACOs on most problems. To handle dynamic problems, ACO should reconstruct the pheromone matrix or use some strategies to update its matrix. While PL calculates its matrix based on the ants' best solutions, which will change as the problem changes. Therefore, PL is more suitable to solve dynamic SPP than ACOs.

5 Conclusions

This paper proposes a probabilistic learning algorithm with a random search operator for solving the dynamic SPP. A set of 3D problems are also designed. PL shows competitive performance in comparison with several peer algorithm on static SPP and it also shows good performance on dynamic SPP. In the future, we will compare PL with other algorithm equipped with dynamism handling techniques.

Acknowledgement. This work was supported in part by the National Natural Science Foundation of China under Grant 61673355.

References

1. Bullnheimer, B., Hartl, R.F., Strauss, C.: A new rank based version of the ant system. A computational study (1997)
2. Bullnheimer, B., Hartl, R.F., Strauss, C.: An improved ant system algorithm for thevehicle routing problem. Ann. Oper. Res. **89**, 319–328 (1999)
3. Cordon, O., de Viana, I.F., Herrera, F., Moreno, L.: A new ACO model integrating evolutionary computation concepts: the best-worst ant system (2000)
4. Deng, W., Chen, R., He, B., Liu, Y., Yin, L., Guo, J.: A novel two-stage hybrid swarm intelligence optimization algorithm and application. Soft. Comput. **16**(10), 1707–1722 (2012)
5. Dong, G., Guo, W.W., Tickle, K.: Solving the traveling salesman problem using cooperative genetic ant systems. Expert Syst. Appl. **39**(5), 5006–5011 (2012)
6. Dorigo, M., Gambardella, L.M.: A study of some properties of Ant-Q. In: Voigt, H.-M., Ebeling, W., Rechenberg, I., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 656–665. Springer, Heidelberg (1996). doi:10.1007/3-540-61723-X_1029
7. Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. **1**(1), 53–66 (1997)
8. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **26**(1), 29–41 (1996)
9. López-Ibáñez, M., Blum, C.: Beam-ACO for the travelling salesman problem with time windows. Comput. Oper. Res. **37**(9), 1570–1583 (2010)

10. Saenphon, T., Phimoltares, S., Lursinsap, C.: Combining new fast opposite gradient search with ant colony optimization for solving travelling salesman problem. Eng. Appl. Artif. Intell. **35**, 324–334 (2014)
11. Shuang, B., Chen, J., Li, Z.: Study on hybrid PS-ACO algorithm. Appl. Intell. **34**(1), 64–73 (2011)
12. Stützle, T., Hoos, H.H.: Max-min ant system. Future Gener. Comput. Syst. **16**(8), 889–914 (2000)
13. Xia, Y., Li, C., Zeng, S.: Three new heuristic strategies for solving travelling salesman problem. In: Tan, Y., Shi, Y., Coello, C.A.C. (eds.) ICSI 2014. LNCS, vol. 8794, pp. 181–188. Springer, Cham (2014). doi:10.1007/978-3-319-11857-4_21
14. Yong Xia, C.L.: Memory-based statistical learning for the travelling salesman problem. In: 2016 IEEE Congress on Evolutionary Computation (CEC). IEEE (2016, accepted)
15. You, X.M., Liu, S., Wang, Y.M.: Quantum dynamic mechanism-based parallel ant colony optimization algorithm. Int. J. Comput. Intell. Syst. **3**(Sup01), 101–113 (2010)
16. Yun, H.Y., Jeong, S.J., Kim, K.S.: Advanced harmony search with ant colony optimization for solving the traveling salesman problem. J. Appl. Math. **2013**, 1–8 (2013)