IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 7, 2020, accepted October 22, 2020, date of publication October 30, 2020, date of current version November 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3035058

MEATSP: A Membrane Evolutionary Algorithm for Solving TSP

PING GUO[1,2], (Member, IEEE), MENGLIANG HOU[1], AND LIAN YE[1]
[1]College of Computer Science, Chongqing University, Chongqing 400044, China
[2]Chongqing Key Laboratory of Software Theory and Technology, Chongqing 400044, China

Corresponding author: Ping Guo (guoping@cqu.edu.cn)

ABSTRACT In recent years, heuristic intelligent algorithms have achieved rapid development in solving combinatorial optimization problems. Travelling salesman problem(TSP) is one of the classical NP-hard problems in combinatorial optimization, and it needs for more accurate and faster algorithms. Combining heuristic algorithm and cell-like P system, this article proposes a membrane evolutionary algorithm for solving TSP(MEATSP). MEATSP abstracts the behaviors of biological cells such as fusion, division, cytolysis and selection into operators, and obtains the optimal solution of the problem through the evolution of membrane structure and objects within the membrane. Experiments on TSPLIB data sets show that the proposed algorithm performs well in both optimal solution and average residual, and it has strong stability when solving TSP with different scales.

INDEX TERMS Membrane evolutionary algorithm, heuristic intelligent, TSP, membrane computing.

I. INTRODUCTION

Among combinatorial optimization problems, travelling salesman problem (TSP) is well known as a classical NP-hard problem. It comes from many application areas, such as transportation route planning, circuit board printing, fiber optic wiring, DNA sequencing, etc. The TSP aims to find the shortest route for a travel agent to sell between multiple cities. Because the time complexity of TSP is $O(n!)$ [1],it can be solved very hardly when the cities increases. Therefor finding an algorithm to solve the large-scale TSP quickly and accurately is very necessary, which can avoid the exponentially increasing computational complexity.

The algorithms for solving TSP are divided into two classes: exact algorithm and heuristic intelligent algorithm. Exact algorithm aims to find the optimal solution of TSP, which can guarantee to terminate with an optimal solution. It include branching and shearing (BAC) [2], branching and broadcasting (BAB) [3], branching and weight (BAP) [4] and dynamic programming algorithm (DP) [5].

The purpose of heuristic intelligent algorithms is to obtain the approximate optimal solution of TSP quickly. It is an effective algorithm for large-scale problems. It can be divided into three categories: route construction, route improvement and hybrid algorithm [6]. Greedy algorithm is the most typical route construction algorithm. The algorithm makes the agent starts from a random city and always chooses the nearest city which has not been arrived yet as the next destination. When the agent returns back to the origin, the route he passes can be a solution for TSP. Though greedy algorithm usually cannot obtain the optimal solution for TSP, it can construct an optimal feasible solution in less time. So it is widely used as a strategy in other algorithms [7].

The route improvement algorithm is to improve the given route through certain strategies, such as switching some paths or vertices. Local intelligent like 2-opt [8], 3-opt [9], or λ-opt are often used as a mutation operator in route improvement.

Hybrid algorithm includes route construction and route improvement both, Population-based heuristic such as artificial bee colony algorithm (ABC), ant colony optimization (ACO) [10], genetic algorithm(GA), simulated annealing(SA) [11], particle swarm optimization(PSO), neural network, tabu search, shuffled frog-leaping algorithm(SFA). These algorithms use both initial solution construction and route improvement to obtain approximate optimal solution.

Some hybrid algorithms based on ACO achieves excellent results in solving TSP. PACO-3Opt [12] is a hybrid algorithm of parallel cooperation. It firstly uses ACO to construct the initial population parallelly and then optimizes each individual by 3-Opt. The algorithm enhances the quality and

The associate editor coordinating the review of this manuscript and approving it for publication was Siddhartha Bhattacharyya.

robustness of the solution, and it performs well in solving large-scale TSP problems in an acceptable time.

Analogously, based on the algorithms PSO, ACO and 3-Opt, the hybrid algorithm PSO-ACO-3Opt [13] was proposed. It use PSO to optimize the parameters of ACO algorithm to generate solution. And finally, the 3-Opt algorithm improves the solution to avoid falling into local optimization. The performance and experimental results of it are better than other algorithms in robustness and accuracy of solution.

An effective local search algorithm named adaptive simulated annealing algorithm with greedy search(ASA-GS) [11] is proposed to obtain more accuracy solutions for TSP. It based on the standard simulated annealing algorithm adopts the combination of three kinds of mutations with different probabilities during its search, and uses greedy search to speed up the convergence rate. The result shows it provides better compromise between running time and accuracy. finish Ant colony extended (ACE) [14] is a novel algorithm belonging to the general ACO framework. The ACE has two specific features: there are two kinds of ants with different task, and the implementation of a regulation policy to control the ants during the searching process. It show superior performance in solving TSP.

Algorithm C-PSO-ACO-kOpt [15] is proposed as a new hybrid algorithm. It uses ACO to generate the initial solution required by PSO algorithm, and then, after lifting each individual in the initial solution space by k-Opt, the PSO can find the optimal approximate solution of TSP. The experimental results show that the algorithm has a good performance advantage in the accuracy and running time of the solution.

Similar to ACO, Fruit fly optimization algorithm(FOA) [16] is a swarm-intelligence optimization algorithm to solve TSP. But there are some defects such as slow convergence rate, easily falling into local optimum and an insufficient optimization precision. Improved fruit fly optimization algorithm(IFOA) [17] proposes three methods to improve the convergence and precision.

An hybrid algorithm discrete symbiotic organisms search(DSOS) [18] uses three mutation-base local search operators to reconstruct the given route. This strategy enables individuals in the population to be improved and extended and thus obtain the approximate optimal solution. Experimental results based on TSPLIB show that DSOS can quickly obtain the approximate optimal solution close to the best known results.

The hybrid max–min ant system (HMMA) [19] integrated with a four vertices and three lines inequality is proposed to search the route for TSP. It firstly let the MMA search the approximate optimal routes, and then the local paths of adjacent four vertices in routes are converted into the local optimal paths with the four vertices and three lines inequality to get the better approximation. The results show that the better approximations are computed with the HMMA than those with the MMA under the same preconditions.

In addition, a new hybrid algorithm [20] combines the DPC and ACO to obtain the approximate solution of TSP problem is proposed in 2018. Similar to the divide-and-conquer method, it firstly divides the problem into a sub-problem by DPC, and then solves and optimizes each sub-problem and the overall problem formed by the sub-problem based on the ACO and k-Opt. The algorithm performs well for problems of different sizes in TSPLIB.

GA-MARL-NICH-LS [21], a new hybrid algorithm proposed in 2018, is using MARL which based on QLearning algorithm [22] to initialize the population of GA. This algorithm proposed a new method named SMX in the crossover operator, and each routes in the population will be optimized by NICH-LS [23] or 2-Opt before next iteration. This algorithm has good accuracy and CPU running time consumption in the performance of specific problems.

Păun Gheorghe proposed membrane computing in 2000 [24], it's inspired by living cell membrane. From the latest research results, Peng *et al*. proposed new kind of neural-like P systems called dynamic threshold neural P systems(DTNP) [25] and a coupled neural P system(CNP) [26]. DTNP can be represented as a directed graph, where nodes are dynamic threshold neurons while arcs denote synaptic connections of these neurons. And CNP systems are a kind of distributed parallel-computing model with a directed graph structure like spiking neural P systems.

Based on membrane computing, references [27] proposed a membrane algorithm combines with a cell-like P-system structure and two approximate algorithms in 2006. It runs different evolutionary algorithm in each inner membrane to obtain solutions of optimization problem. This kind of membrane algorithm can solve travelling salesman problem and min storage problem [28] better than applying the approximate algorithms alone.

There are some developments in membrane algorithm with various variants. Zhang et al combined the hierarchical structure with the quantum-inspired evolutionary algorithms (QIEAs) [29], which solves the knapsack problem. And Cheng *et al.* proposed a novel membrane algorithm based on differential evolution for numerical optimization [30] by combining the hierarchical structure and a local search algorithm. In [31], a novel membrane algorithm based on the hierarchical structure and particle swarm optimization (PSO) solved the broadcasting problems. And an adaptive membrane algorithm combining the hierarchical structure and local search was proposed to solve the travelling salesman problem in [32].

Although membrane computing has made a lot of theoretical achievements since it was proposed, due to the current level of scientific research, the use of biological cells to complete membrane computing has not been realized. In the development of membrane computing, it is also an important branch to introduce the idea of membrane computing to design algorithms to solve practical problems on computers, such as membrane algorithm and membrane evolutionary algorithm(MEA) in this article. TSP is a classic NP-Hard problem. Although there are many excellent algorithms that can find excellent solutions in a certain time, heuristics cannot

find an optimal solution of TSP in most cases. How to approach the optimal solution is still a problem worth studying. Moreover, different algorithms have different views on solving the same problem. The addition of MEA may provide new ideas for solving this problem.

In this article, the membrane evolutionary algorithm for solving TSP(MEATSP) is proposed as a new heuristic intelligence algorithm to solve the TSP. The MEATSP is abstracted from the biological characteristics at the cell level, and four operators including fusion operator, division operator, selection operator and cytolysis operator are designed by simulating the activities of cells. Each membrane contains a candidate solution of TSP. The population of the membranes improved through the four operators in each iteration. And when it comes to termination, the approximate optimal solution is in the membrane with highest fitness. Experiments of MEATSP on TSPLIB show its excellent performance in both the best solution and average residual.

Therefore, the main innovations of this article are: (1) A membrane evolution algorithm framework MEAF is proposed, which is inspired by the structure and function of living cell membranes. (2) The membrane evolution algorithm MEATSP, which uses MEAF to solve TSP, is proposed. Four membrane evolution operators are designed: Division, Fusion, Selection and Cytolysis. (3) MEATSP has excellent and stable experimental results on both small and large TSPs.

The rest of this article is arranged as follows. Section II will introduce some heuristic intelligence algorithm and membrane computing. In section III, the membrane evolutionary algorithm framework(MEAF) is detailed. Section IV proposes the MEATSP based on the MEAF. The results of comparison and analysis with other algorithms are arranged in section V. Finally, section VI puts forward some conclusions and prospects for the follow-up work.

II. PRELIMINARIES

A. TRAVELLING SALESMAN PROBLEM

In a completely undirected graph $G = (V, E)$, $V = \{v_1, v_2 \ldots, v_n\}$ represents the set of the vertices in G, and $dist(v_i, v_j)$ represents the distance from vertex v_i to vertex v_j. The TSP based on G is to find a path through each vertex of G once and only once then back to the start vertex. The path $P = v_{i_1}\ v_{i_2} \ldots v_{i_n}$ should make sure that any path formed by swapping vertices in P likes $P' = v_{j_1}\ v_{j_2} \ldots v_{j_n}$ is:

$$dist(P) \leq ist(P') \tag{1}$$

where,

$$D(P) = \sum_{k=1}^{n-1} dist(v_{i_k}, v_{i_{k+1}}) + dist(v_{i_n}, v_{i_1}) \tag{2}$$

$$D(P') = \sum_{k=1}^{n-1} dist(v_{j_k}, v_{j_{k+1}}) + dist(v_{j_n}, v_{j_1}) \tag{3}$$

TSP is a classical NP-hard problem, many intelligent algorithms have been proposed. Especially in recent years, researchers have proposed a batch of algorithms with high efficiency and good solution quality. For example, K-optimization algorithm(K-opt), a based on density peaks clustering and ant colony optimization algorithm(DPC-ACO-KOpt), etc.

B. K-OPT ALGORITHM

K-opt algorithm is a local search algorithm, which is often used to solve the TSP and its similar problems. On the basis of a route of the TSP, k-opt algorithm achieves the possibility of improving the current path by removing K edges for reconnection operation. Because of this property, k-opt algorithm avoids the results of some algorithms falling into local optimality.

For a given route of the TSP, in the case of removing its k edges, the total exists $(k-1)!*k-1$ new routes for reconnecting. Among these new generated routes, there may be some routes better than the previous, which is an improvement of the solution. However, as the k increases, the time consumed by the algorithm will increase accordingly. 2-opt [8] and 3-opt [9] are two types of k-opt algorithms commonly used to solve the TSP.

The Fig. 1 shows the case of 2-opt algorithm, which deletes two edges in the route and reconnects them to build a new one. For the unmodified route is included, the 2-opt algorithm can generate two routes, and then choose the route with the smallest length as an improvement. Similarly, 3-opt algorithm will delete three edges in the route and reconnects the new route based on it. For the unmodified route is included, as shown in Fig. 2, there are 8 possibilities. It can be seen that among the 8 cases generated by the 3-opt algorithm, 3 routes are equivalent to the 2-opt algorithm. It shows that if a reconstructed solution is the result of 2-opt algorithm, it also belongs to 3-opt algorithm [33]. Thus, the 3-opt algorithm can provide better candidate solutions, but it runs much slower.

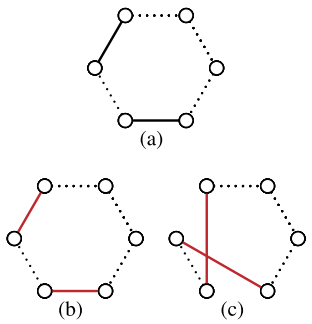


FIGURE 1. All possible reconnections of 2-Opt(The solid line represents the edge to be replaced and the dashed line represents the omitted vertex). Including the original route(b), and a new tour(c).

C. DPC-ACO-KOpt ALGORITHM

The algorithm DPC-ACO-Kopt [20] proposed in 2018 is a hierarchical algorithm based on heuristic intelligent, which based on the idea of divide and conquer to quickly solve TSP, especially when the problem size is particularly large.

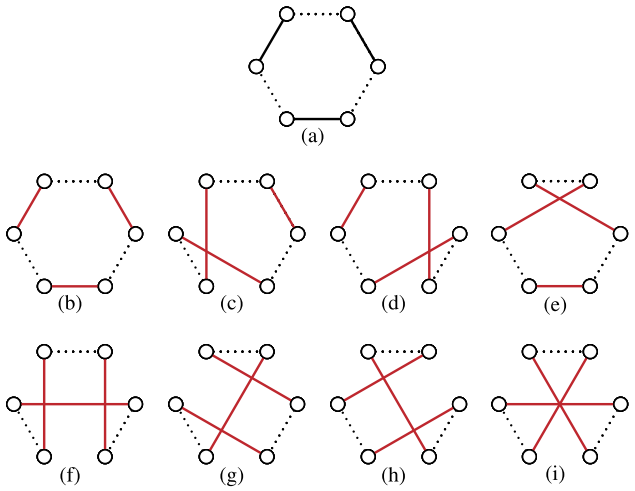
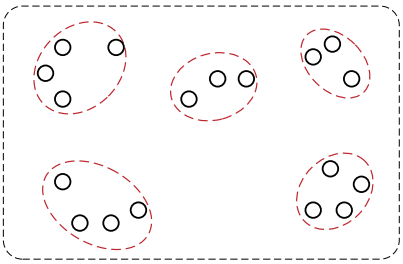
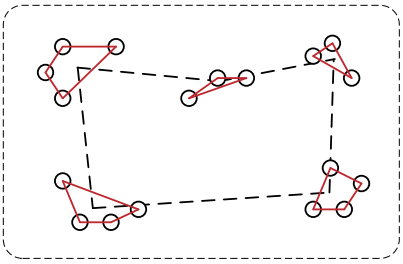


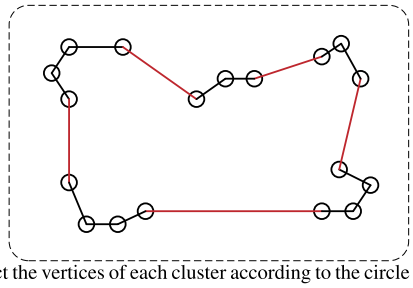
FIGURE 2. 8 possible reconnections in total of 3-Opt. Contains reconnections similar to 2-Opt((c), (d), (e)) and original tour(b), and 4 reconnections swap all the solid edges((f), (g), (h), (i)).



(a) Using DCP to cluster all vertices and find the center point of each cluster



(b) After ACO is implemented, the circle of each cluster and all clusters are found



(c) Reconnect the vertices of each cluster according to the circle of the clusters

FIGURE 3. An example of the procedure of DPC-ACO-Kopt.

Generally, the ant colony algorithm(ACO) is used to solve the TSP problem, but the required running time and the quality of the solution are in the optimal region only when the vertices are less than 40. Otherwise, the result will show a downward trend. It's because the computational complexity of ACO increases rapidly with the increase of TSP, especially when the vertices are more than 200. So the DPC-ACO-KOpt firstly use density peaks clustering(DPC) to cluster the vertices to reduce the size of the problem to a certain extent, which improve the performance of ACO in solving large-scale TSP problem.

DPC [34] is an unsupervised learning algorithm. It centers on some vertices with extremely high density in TSP, and non-central vertices are clustered to the center point with the nearest neighbor with high density. The algorithm requires no iteration and only linear time to divide all vertices into two layers. The lower layer is composed of all clusters after clustering, each cluster is a new small-scale TSP. The upper layer is a collection of the central points of all the clusters, which, like the lower layer, can also be regarded as a small TSP. Since the problem size of each level is reduced by the DPC, both levels can use the ACO to accurately and quickly find the target route of the problem. After that, local TSP routes in each lower cluster will be merged according to the route of the upper central vertices, so as to construct the initial global route. The final global is obtained after optimizing the initial route by k-opt algorithm.

It shows in Fig. 3, globally, each cluster in the upper layer is regarded as a vertex of a small-scale TSP, and a global route can be obtained after the calculation of ACO. In this route, each edge between the clusters will serve as a reference to connect the vertices inside the two clusters. For each cluster in the lower layer, a local Hamilton route is obtained by ACO too. Each cluster need disconnect a vertex in local Hamilton route to connect another cluster which is decided by the route in upper layer. So the vertex should be the closest one to the nearby cluster. Then all the vertices will connect following the upper route to construct the initial global route.

This algorithm decomposes the problem into subproblems, which improves the solving speed. However, since the basic method to solve the TSP problem is ACO, the accuracy and running time of the algorithm will be affected by the classification results. When the number of nodes in the cluster increases, the running time of solving the local loop of the cluster with ACO will become longer and the accuracy of the solution will decrease. These effects will also apply to the global TSP. In addition, on the basis of the initial global route, the algorithm is finally optimized with k-opt algorithm, because the new path generated when local solutions are merged into the initial global route is not necessarily the optimal solution.

D. MEMBRANE COMPUTING

Membrane computing as biological computing model was proposed by professor Păun in 2000 [27]. The biological computing is abstracted from the activity rules of different levels of organisms in nature, and membrane computing fills the gap in the biological computing at the cell level. Prior to this, the biological computing included ant colony algorithm and

swarm algorithm for population-level, and neural network and immune algorithm for cellular tissue level, and genetic algorithm for biomolecular level.

Membrane computing, also known as the P system, is a cellular computing model that performs calculations by defining the class and the structure of the membrane, and the rules for how its materials are reflected. P system includes cell-like P system, tissue-like P system and neural-like P system. The general definition of a cell-like P system is shown in (4).

$$\Pi = (O, \mu, \omega, R, i_o) \tag{4}$$

where O as a set represents all the substances used in the entire P system, and the substances are encoded to cover the input and output of the problem, also include the intermediate results. The μ denotes the structure of a P system, as shown in Fig. 4. The structure includes the membranes, the regions, and the environment. Set ω is a multiplicity of the substances in each region. The substances will react according to the reaction rules in set R, including substance change, ablation, membrane entry membrane exit. Likewise, set R includes splitting, merging, and dissolution for each membrane. After a certain time of reaction until the reaction is stopped, the multiple sets in the region of the output membrane i_o will be the bearing result represent the calculated results.

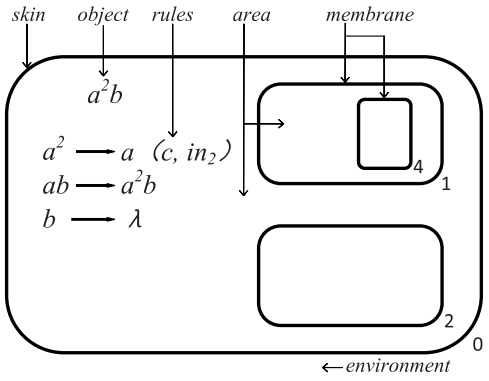


FIGURE 4. The structure of a membrane computing with an example.

The compute capability of membrane computing is proved to be equivalent to Turing machine. Because the execution of the rules in membrane computing is random and parallel, which enables it to solve complex problems such as NP-hard problems [35], a P system model that can solve ALL-SAT problems in polynomial time is designed, also a P system that can solve TSP in polynomial time.

However, due to the limitations of current biotechnology, the research of membrane computing is still in the theoretical stage. However, in the research of membrane computing, we have established the membrane evolution algorithm based on the characteristics of membrane combined computing system and the application of heuristic search algorithm. In next section, we proposed the membrane evolutionary algorithm framework(MEAF).

III. MEMBRANE EVOLUTIONARY ALGORITHM FRAMEWORK

This section briefly introduces the membrane algorithm (MA), and then gives a framework of membrane evolution algorithm (MEAF).

A. MEMBRANE ALGORITHM

The membrane algorithm was first proposed by Nishida in 2006 [27]. It evolves feasible solutions of optimization problems by placing different optimization algorithms (called sub-algorithms) in different membranes (regions).Sub-algorithms in different membranes are executed in parallel, and the best and worst solutions from each evolution are transmitted to the adjacent inner and outer membranes respectively. The next evolutionary population comes from the results selected from the adjacent inner and outer regions and from other possible solutions left in the current membrane. At the end of the algorithm, the best solution of the optimization problem is saved in the innermost membrane. The membrane structure adopted by Nishida's membrane algorithm is shown in Fig. 5.

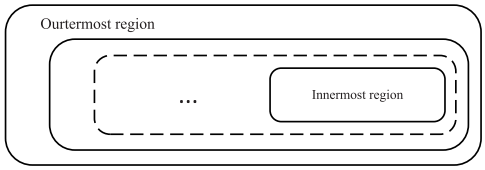


FIGURE 5. The structure of a membrane algorithm.

Reference [36] reviewed the membrane-inspired evolutionary algorithms (MIEAs). This article classifies MIEAs from membrane structure, evolution rules and meta-heuristic algorithm used in P system. According to the membrane structure can be divided into hierarchical structure and network structure. Further, the hierarchical MIEAS is divided into nested membrane structure (NMS), one-level membrane structure (OLMS), hybrid membrane structure (HMS) and dynamic membrane structure (DMS). And network structure MIEAS are classified as statically network structures (SNS) and dynamically network structures (DNS).

Literature [37] discussed in detail the membrane algorithms formed by different meta-heuristic algorithms in MIEAs, including Genetic Algorithm Based on P System (GAPS), quantum-inspired Evolutionary Algorithm Based on P Systems (QEPS), Ant Colony Optimization Based on P Systems (ACOPS), Differential Evolution Based on P Systems (DEPS).

In general, the membrane algorithm has the following basic characteristics:

1) There is a certain communication mechanism between the membranes.
2) Each membrane contains a sub-algorithm (the sub-algorithms of different membranes may be different), such as GA, ACO, etc. The running of these sub-algorithms makes the population evolve.

3) Exchange of feasible solutions between membranes promotes the evolution of feasible solutions within membranes.

B. MEMBRANE EVOLUTION ALGORITHM FRAMEWORK

The membrane structure of MEAF generated during initialization is shown in Fig. 6. The membrane labeled 0 is the skin, it includes $m + 1$ offspring membranes, which m is the size of population. The membrane labeled i ($1 \leq i \leq m$) is the ith individual, witch includes a candidate solution. The membrane labeled *best* is used to store the historical optimal solution. As shown in algorithm 1, membrane evolution is completed by four operators (membrane fusion, membrane division, membrane cytolysis and membrane selection), and membrane evolution is the evolution of feasible solutions within the membrane. Population repair to ensure population diversity and individual population stability. The condition for the end of MEAF can be that evolution reaches the set algebra or that the obtained optimized solution does not change.

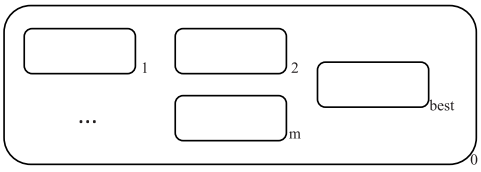


FIGURE 6. The membrane structure of MEAF.

The membrane operated by the four membrane evolution operators is determined by the fitness of itself or is randomly selected. As an algorithm framework, MEAF fitness calculation function should be determined according to the problem solved. For example, for TSP, the fitness function can be the inverse of the route's length.

C. COMPARISON OF MEA WITH GA AND MA

MEA and GA are both abstract algorithms based on the laws of biological activities. They also solve the optimal solution by iterating individuals in a population according to the principle of "survival of the fittest". In addition, both algorithms control the evolution direction of the population by fitness function, making the population evolve towards the optimal solution in each iteration. Similarly, there are some similarities between MEA and MA. Both of them are extensions of the membrane algorithm in practical applications, and both of them are abstract algorithms based on the activity rules of biological cell membranes.

Different from GA, which abstracts three operators for population iteration according to the activity of biological DNA: selection, crossover and mutation. MEA is based on the activity of biological cell membrane abstract four operators: division, fusion, cytolysis and selection to carry out the iteration of population. In addition, MEA performs population repair in each iteration to ensure population size and diversity.

Algorithm 1 MEAF;//Membrane Evolution Algorithm Framework

Input: The description of question, conditions for the solutions, end connditions

Output: Results

(1) **Initialization** *generate initial membrane structure and construct initial population* ;
repeat
  (2) **Evolve** *population*:
    1)Division: divide the selected membrane into two membranes; `// divide the membrane according to its fitness or randomly selected. Objects can be divided into two membranes according to the correlation of objects in the membrane or randomly selected objects`
    2) Fusion: merge the two selected membranes and reconstruct the multiple sets within the membrane into a new feasible solution; `// select the membrane to merge according to the correlation between the membranes, or select the membrane to merge randomly`
    3) Cytolysis: dissolve the selected membrane and the objects in the membrane; `// dissolve the membrane according to its fitness or at random`
    4) Selection: select the membrane with high fitness for replication and enter the next generation, and eliminate the membrane with low fitness; `// At the same time, update the best in-membrane object with the most appropriate in-membrane object`
  (3) **Repair** *population*:
    1) If the multiple sets in multiple branes are the same, only one copy is kept and other copies are deleted;
    2) Several membranes and intramembrane multiple sets are generated randomly to keep the population stable;
until *The ending condition of evolution is satisfied*;
(4) **return** results;

Different from the MA, the MEA takes the candidate solution as each membrane and then iterates the membrane in the population with four operators to search for the optimal solution. however, the membrane of the MA is used to encapsulate different algorithms and complete the process of evolution

and solution through the communication mechanism between the membranes.

MEA has some similarities with GA and MA, but there are also some differences. These differences are also the characteristics of MEA as a new algorithm, which simulates the activity of cell membrane to iterate the population and achieve the goal of solution.

IV. MEATSP DESIGN

This section mainly discusses MEATSP, a new TSP algorithm proposed, which will solve the TSP based on MEAF.

A. DEFINITION

The membrane structure of MEATSP is shown in Fig. 7. Object e_i in the membrane i represents an edge of the graph $G = (V, E)$ and object w_i is its fitness. Therefor each membrane contains n edges for graph G with a total of n vertices. After initialization, the contained edges in each membrane can be connected first to form a tour containing each vertex, that is, a solution corresponding to the TSP. The sum of the weights of all the edges will also be kept in the membrane as the fitness of each membrane.

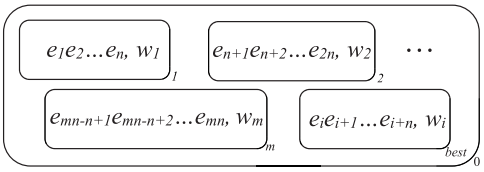


FIGURE 7. The membrane structure of MEATSP.

The fitness of each membrane indicates how healthy the membrane is in the membrane population, and it's the inverse of the total weight in the membrane. Fitness will be used as an indicator of ''survival of the fittest'' in the membrane evolutionary process. Divided offspring number *DON* determines the quantity of offspring of each membrane, one of each offspring will contain a path of father membrane. When $DON = 2$, there are 2 possible to reconnect for the offspring. And there are 8 possible to reconnect the offspring for $DON = 3$. The larger *DON*, the more possibilities for membrane evolution will be provided. Meanwhile, the running time required for reconnection will also increase.

B. PROPOSED MEATSP

This subsection describes the MEATSP used to solve the TSP problem by iteratively executing four operators. After initialization, each membrane follows the population to be iterated, and a new population will be obtained by division, fusion, selection and cytolysis operators respectively. Then, the membrane with the greatest fitness in the population will be used to replace the current optimal membrane of the population before iteration.

Algorithm 2 describes MEATSP in detail. During the initialization, it creates the initial membrane structure and a multiplet set within each membrane as initial population. After initialization, fitness function within each membrane is automatically calculated and updated, and the membrane with the maximum fitness is copied to M_{best} as the current optimal. When the iterations of the population reaches a maximum iterations *MIT*, it will output the M_{best} as the optimal solution.

Algorithm 2 MEATSP

Input: graph $G = (V, E)$, population size *PS*, the maximum iterations *MIT*,divided offspring number *DON*, selection and cytolysis ratio *SCR*
Output: a tour of G

```
(1) Initialization
    MP = InitialPopulation (PS);
    select a membrane with maximal fitness and copy its objects into M_best ;
repeat
    (2) Evolve membrane population:
        MP = Division(MP, DON);
        MP = Fusion(MP, DON);
        MP = CytolysisAndSelection(MP, SCR);
    (3) Repair membrane population:
        MP = Repair(MP);
until the iterations >= MIT;
(4) return M_best;
```

Each iteration can be divided into two stages as the membrane evolution and the population repair. At each stage, different operators will be process to operate the membrane population to obtain a new one, and then the multiple set in the membrane with the maximal fitness will be copied to the M_{best}.

In membrane evolution, four different operators will be used to operate each membrane in the population. Detailed introduction of operators will be described in the next subsection. In this stage, new individuals are generated through division (see subsection IV-C2) and fusion(see subsection IV-C3), and random values are added in this process to avoid the results falling into local optimization. However, after selection and cytolysis(see subsection IV-C4), some membranes with the minimal fitness will be dissolved base on *SCR*, which cause the population size changed and is not conducive to the stability of the population iteration.

The above problems were solved in the population repair(see subsection IV-C5), which performed two stages to ensure the size and the diversity of the population. Firstly, for these identical membranes in the population, it will remove these identical membranes and retain only one of them. Such operations avoid ''inbreeding'' in the iteration, which can easily lead to local optimal results. Then it will reconstruct the new membrane until the size is consistent with the initial population. It both increases the stability of membrane quantity and ensures the diversity of membrane and in the population.

C. OPERATORS OF MEATSP

Each operator of the proposed MEATSP is described in detail in this subsection and includes initialization and population repair.

1) INITIALIZATION

The initialization focuses on the construction of a series of initial TSP solutions. The initial population should be as close as possible to the optimal solution and can only be generated in a short time. At the same time, the initialized membrane population should contain as many different membranes as possible. The former is to improve the quality of the solution as much as possible, while the latter ensures the diversity of the membrane population, so as to avoid the result falling into the local optimal solution.

Algorithm 3 InitialPopulation

Input: graph $G = (V, E)$, population size PS
Output: membrane population MP
for $i = 1$ *to* PS **do**
  M_i = ConstructMembrane(G);
  add Mi *to* MP ;

As shown in algorithm 3, in the initialization phase, new membranes are generated and added to the membrane population MP, until the population reaches PS, and then the initialization is completed. At this stage, since the membrane are independent of each other, they can be constructed in parallel to shorten the running time.

Algorithm 4 with a greedy strategy to select the edge in the graph $G = (E, V)$ to enter the membrane M_i to generate a tour. The e^n shows the quantity of edges existing in M_i. For the newly generated empty membrane M_i $e^n = 0$, and then randomly select an edge from E into it as a initial path. Then, the next edge to be added to the membrane M_i is selected according to the vertex that only appears once in it, also as the endpoint of the path. Besides, the edge is required not to include both endpoints of the path to avoid the appearance of the circle. Greedy strategy needs to select the edge with the least weight, and also randomly selects an edge to move into M_i when the weight of each edge is the same. Algorithm 4 increases the difference between membranes and the diversity of membrane populations while ensuring the rapid generation of a membrane.

Algorithm 4 ConstructMembrane

Input: graph $G = (V, E)$, the quantity of V (n)
Output: a membrane M
M = empty membrane;
e^n = the quantity of edges in M;
randomly *select an edge from* E *into* M ;
while $e^n < n$ **do**
  select *2 edges from* E *with minimal weight and its vertexes in* M *is* $= 1$;
  add *those edges into* M ;
return M;

2) DIVISION

The division operator is the first operator in the membrane evolution phase, each membrane will divide into 2 offspring membranes base on $DON = 2$. Then the offspring M_i' and M_i'' inherit a path of the parent M_i respectively, whtch $M_i' \cap M_i'' = \oslash$.

Algorithm 5 describes the realization process of the division operator in detail. Taking M_i as an example, if it want to be preserved in the next iteration, then according rule of fitness function and the dissolution, two edges with the largest weight should be removed in M_i. Then store these 2 paths in different offspring membranes. Moreover, adding a random strategy to remove edges can increased the diversity of the population.

Algorithm 5 Division

Input: membrane population (MP)
Output: membrane population (MP)
for $\forall M_i \in MP$ **do**
  randomly remove 2 edges with probability p **or** remove 2 edges with maximal weight in M_i;
  M_i' = empty membrane;
  move a path from M_i into M_i';
  add M_i' into MP;
return MP;

3) FUSION

The fusion operator is performed on the basis of the completion of division. It let each offspring membrane reconnect to a new membrane and obtain the it's fitness value.

In algorithm 6, for membrane M_i' and membrane M_i'', the vertexes that only appear once in them represented as the endpoint of path. The new membrane needs to ensure that a

Algorithm 6 Fusion

Input: Graph $G = (V, E)$, membrane population MP
Output: membrane population MP
for $M_i < MP$ **do**
  $E' = \{v_1$ to $v_2 \mid v_1 \in M_i$ **and** v_1<2, $v_2 \in M_i'$ **and** v_2<2$\}$; `// find all the edge witch can reconect the membrane as a loop`
  select *2 edges form* E' *with minimal weight* ***and*** *each vertex shows only once* ;
  add *edges into* M_i ;
  $M_i = M_i \cup M_i'$;
return MP;

completed tour in, so algorithm needs to find the suitable edge to reconnect them. The edges within set E' need to contain the endpoints within both of offspring. Select 2 edges from E 'that can contain all the endpoints and to reconnect a new tour.

In this case, membrane M_i divides into 2 offspring membranes, each offspring containing a path. In the process of fusion, there are 2 possible combinations of edges (including the original path) that can reconnect the paths. The more offspring membrane, the more combinations of alternative edges will be available in fusion, and the greater possibility that fitness value of the new membrane will increase.

4) SELECTION AND CYTOLYSIS

In the selection and cytolysis stage, the membrane selection operator will screen out N membranes with the highest fitness value for retention. As shown in algorithm 7, after each membrane in the completes the fusion, the membrane with the minimum fitness will be dissolved from the membrane population one by one until the number of membranes is equal to N. Such selection conditions help the whole population evolve toward the optimal solution.

Algorithm 7 CytolysisAndSelection

Input: membrane population MP, selection and cytolysis ratio SCR
Output: membrane population MP

sort *the MP base on fitness of each membrane* ;
while *the size of* $MP > SCR$ **do**
    remove *a membrane with the minimal fitness* ;
return MP;

5) POPULATION REPAIR

Last stage will reduce the number of membranes in the membrane population, making it smaller than the initial set membrane population size PS, and there may be multiple identical membranes in the membrane population. Such results make the quantity of membrane population unable to meet the requirements in the process of next iteration, and the simplification of membrane in the population makes the solution easy to fall into the local optimal. To avoid it, the membrane population will be repaired before the next iteration.

As shown in algorithm 8, firstly the same membranes are dissolved and kept with only one of them to left more room for new ones. Then, a new membrane is generated by algorithm 4 to increase the diversity of the membrane population.

Algorithm 8 Repair

Input: graph $G = (V, E)$, membrane population MP,population size PS
Output: repaired membrane population MP

for $M_i \in MP$ **do**
    select *membrane* M_o *which is equal to* M_i ;
    remove M_o ;
while *the size of* $MP < PS$ **do**
    $M =$ ConstructMembrane(G);
    add M *to* MP ;
return MP;

D. EXAMPLE OF MEATSP

To illustrate the implementation of these operators in MEATSP, an example of the TSP is used to demonstrate how MEATSP performs a population iteration. In this example, a fully connected graph is shown in Fig. 8, and the distance matrix for it is equation (5). we assume the membrane population size (PS) is set as 6, and divided offspring number $DON = 2$, and selection and cytolysis ratio $SCR = 4$.

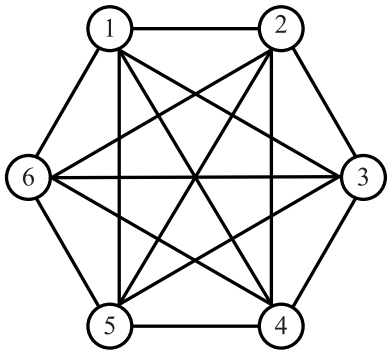


FIGURE 8. A full connection graph with 6 vertices.

Fig. 9(a) shows the membrane population of this example after initialization algorithm is executed with the above setting. Each membrane contains multiple sets of objects representing a solution to the TSP problem and a fitness function. M_5 with the maximal fitness is copied to M_{best}. Furthermore, because of there are fewer vertices in Fig. 8, the population initialized by the initialization algorithm could quickly approach the optimal. In order to demonstrate the practical effect of other operator, for this example, no strict greedy strategy is implemented in initialization.

$$dist(v_i, v_j) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{pmatrix} 0 & 1 & 4 & 3 & 1 & 2 \\ 1 & 0 & 4 & 2 & 1 & 3 \\ 4 & 4 & 0 & 5 & 2 & 1 \\ 3 & 2 & 5 & 0 & 5 & 4 \\ 1 & 1 & 2 & 5 & 0 & 4 \\ 2 & 3 & 1 & 4 & 4 & 0 \end{pmatrix} \end{array} \tag{5}$$

1) DIVISION

Division operator is the firstly executed during iteration. According $DON = 2$, each membrane will divide into 2 offspring. There are 2 strategies to disconnect 2 edges and obtain 2 paths in each offspring. As shown in Fig. 9(b), M_1 and M_5 randomly disconnect 2 edges, others is to disconnect

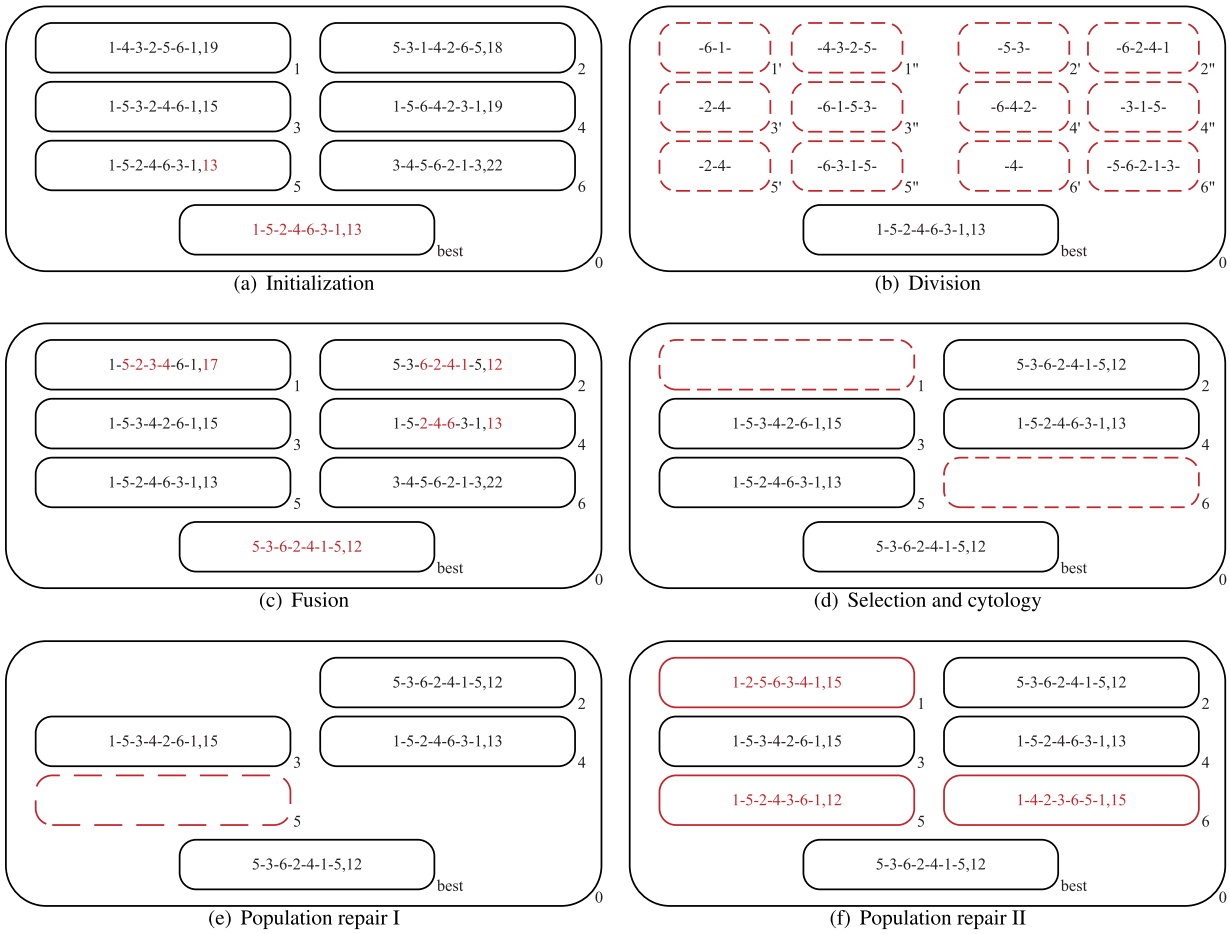


FIGURE 9. An iteration on example of MEATSP.

2 edges with the maximal weight. And if there are more than 2 edges with maximal weight, it will randomly disconnect 2 edges of these maximal edges. After division, except for M_{best}, each membrane divide into two offspring.

2) FUSION

Division operator divides each membrane into two offspring according to the *DON*, and then the fusion operator reconnects these offspring according to the fitness function to create a new membrane population. For this case where 2 offspring have two possibilities of reconnection, the fusion operator compares the fitness of these 2 possibilities to determine the way of offspring fusion. Fig. 9(c) shows that after fusion, the M_1 with increased fitness and the M_3 with original tour. If there is a membrane with a fitness greater than M_{best}, as shown in Fig. 9(c), the tour and fitness function in M_{best} are replaced by M_2.

3) SELECTION AND CYTOLOGY

Selection and cytology operators dissolve the membranes of membrane population by the principle of ''survival of the fittest''. It allows the population to leave more room for other membranes. In the case that $SCR = 4$, as shown in Fig. 9(d), It dissolve M_6 and M_1 with the minimum fitness one by one, until the population size is equal to *SCR*.

4) POPULATION REPAIR

On the basis of the previous operator, firstly, the population repair further dissolves more membranes to leave enough space for the others. It finds some membranes with the same fitness and tour in the population, dissolve these membranes and keep only one copy of them. In Fig. 9(e), M_4 and M_5 include the same tour and fitness. It dissolves M_5 and retains M_4 to avoid a large quantity of redundant membranes, which can easily let the MEATSP falls into the local optimal solution.

The next stage of population repair is to repair the population size to its original. Fig. 9(f) shows that, after the first stage of population repair only 3 membranes were left in this case. Then, the population size is restored to the previous by adding membranes to the population using algorithm 4. After population repair, the population size and the diversity of the membranes are guaranteed, and preparations are made for the next iteration.

V. EXPERIMENT AND RESULTS ANALYSES

This section discusses the experimental results of MEATSP, testing different scale instances of TSPLIB and comparing them with other algorithms proposed in recent years to illustrate the performance of the MEATSP.

A. EXPERIMENTAL ENVIRONMENT AND PARAMETERS

The MEATSP algorithm proposed in this article is compiled and implemented by C++ programming language. All experiments were evaluated on an Windows based on 2 cores of an Intel i7-4800MQ 2.5 GHz CPU and 32 G Byte RAM.

We selected some instances in the data set TSPLIB for the experiment, and the data set is divided into small-scale TSP problem and large-scale TSP problem according to the quantity of vertices in the instance. TSPLIB contains multiple TSP data sets for testing algorithms and experimental analysis, most of which are derived from practical applications, such as *berlin*52 from 52 cities in Berlin, Germany.

For each instance, The MEATSP run 10 times to consider its performance and the maximum iterations *MIT* is 200 and the population size *PS* is 20. For other parameters, the division offspring number *DON* is set at 2, and the selection and cytolysis ratio *SCR* is set at 0.75, the better robustness can be achieved

For the setting of *PS*, 5 instances with similar results but different vertices are selected for the experiment. *PS* is set from 10 to 50. After running MEATSP, the current iteration of output(*OI*) result can be obtained. EMQ means the quantity of effective membranes, which is the product of *OI* and *PS*. The smaller EMQ means the least membranes are used by MEATSP under the *PS*, and the higher the efficiency is. According to the results in Table 1, determine the *PS* as 20.

TABLE 1. The quantity of effective membranes on different instances.

PS	eil76		kroA100		ch150		pr226		pcb442	
	OI	EMQ	OI	EMQ	OI	EMQ	OI	EMQ	OI	EMQ
10	15.2	152.0	17.2	172.0	27.5	275.0	31.5	315.0	47.6	476.0
15	9.5	142.5	12.4	186.0	15.4	231.0	20.6	309.0	29.1	436.5
20	7.0	140.0	8.4	168.0	10.2	204.0	14.6	292.0	20.7	414.0
25	7.3	182.5	8.2	205.0	7.9	197.5	13.2	330.0	16.4	410.0
30	6.9	207.0	8.1	243.0	7.6	228.0	12.9	387.0	15.8	474.0
35	6.5	227.5	7.8	273.0	7.6	266.0	13.5	472.5	15.2	532.0
40	6.5	260.0	8.0	320.0	7.0	280.0	13.0	520.0	15.0	600.0
45	6.2	279.0	7.6	342.0	7.5	337.5	12.4	558.0	13.7	616.5
50	6.3	315.0	7.4	370.0	6.8	340.0	11.8	590.0	13.1	655.0

The results of MEATSP are shown in Table 2 and Table 3. After counting the results of 10 runs of each instance, we take the average and compare it with other algorithms from different dimensions. Column 1 is the name of MEATSP and the comparison algorithm, column 2 *avg*(*Res.*)is the average residual of the algorithm over all instances, and column 3 is the comparison of different dimensions of each algorithm. The 1st line is the name of each instance with *OTL* witch means the preset optimal tour length. The 2nd line, *Best* represents the best result of these algorithms running on a case. The 3rd line, *Avg.* is the average result of these algorithm running on each instance. The 4th line, *SD.* represents the square deviation of these algorithms running on an instance. And the 5th line, *Res.*, represents the average residual of the algorithm. The definition of it in the calculation formula is shown as (6). In each table, the best results are shown with underline and '−' indicates that the algorithm has no test data in the current sample experiment.

$$Res. = \frac{Avg. - OTL}{OTL} \tag{6}$$

B. EXPERIMENTAL RESULTS

In this subsection, MEATSP's experimental results are divided into two parts according to the size of TSP and compared with different algorithms.

1) MEATSP SOLVING THE SMALL-SCALE TSP

Table 2 shows MEATSP's results compared to other algorithms running on small-scale TSP. To verify MEATSP's excellent and stable running results on instances with fewer vertices, in this section, the instances of TSPLIB with vertices between 50 and 200 are selected for experiment. The instance names sorted by vertices are: eil51, berlin52, st70, eil76, rat99, kroA100, eil101, lin105, ch150, and kroA200. They cover different types of instances in small-scale TSP. And there are 12 algorithms on these instances for comparison, respectively as follows: DPC-ACO-KOpt(2018) [20], PACO-3OPT (2016) [12], PSO-ACO-3OPT (2015) [13], ACE(2015) [17], ASA-GS(2011) [11], SA-ACO-PSO (2011) [38], WFA-3OPT (2013) [39], ACO-Taguchi(2013) [40], IFOA (2017) [16], DSOS (2017) [14]. It is important to note that these algorithms are not implemented here, but directly refer to the results of these algorithms on these instances.

As can be seen from Table 2, MEATSP has absolutely excellent experimental results in small-scale TSP compared with the other 12 algorithms, and its residual is almost 0. This result shows that MEATSP has an advantage over *Avg.*, *Res.* for all instances. As can be seen from the experimental results of eil101, MEATSP's *Avg.* is slightly larger than the OTL of eil101. However, such a result is also better than the suboptimal result of C-PSO-ACO-KOPT 0.17%.

2) MEATSP SOLVING THE LARGE-SCALE TSP

There are 10 examples selected from TSPLIB for MEATSP's experiment on large-scale problems. The results are shown in Table 3. The number of points in these instances is between 400 and 1000, sorted by number of vertices: rd400, fl417, pr439, pcb442, d493, rat575, p654, d657, u724, rat783.They contain different kinds of large-scale TSP problems. Meanwhile, the algorithm compared with MEATSP's experimental results is as follows: DPC-ACO-KOpt(2018) [20], PACO-3Opt(2016) [12], PSO-ACO-3Opt(2015) [13], HMMA(2015) [19], PMSOM(2015) [41], HCACO (2014) [42], HGA (2014) [43], ASA-GS (2011) [11], and DSOS(2017) [14].It is worth noting that these algorithms are not implemented in this article, and their experimental results are quoted from the original paper.

In Table 3, each experimental result of MEATSP has an absolute advantage over other algorithms in terms of

TABLE 2. Experimental results of MEATSP and other algorithms on small-scale TSP.

Algorithm	avg(Res.)	instance	eil51 (426)	berlin52 (7542)	st70 (675)	eil76 (538)	rat99 (1211)	kroA100 (21282)	eil101 (629)	lin105 (14379)	ch150 (6528)	kroA200 (29368)
MEATSP	0.00	Best	426.00	7542.00	675.00	538.00	1211.00	21282.00	629.00	14379.00	6528.00	29368.00
		Avg.	426.00	7542.00	675.00	538.00	1211.00	21282.00	629.80	14379.00	6528.00	29368.00
		SD.	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.00	0.00	0.00
		Res.(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
DPC-ACO-KOpt(2018)	0.07	Best	426.00	7542.00	675.00	538.00	1211.00	21282.00	629.00	14379.00	6528.00	29368.00
		Avg.	426.25	7542.00	675.25	538.30	1211.90	21283.65	630.35	14380.10	6536.50	29368.40
		SD.	0.44	0.00	1.11	0.80	1.25	5.50	2.01	4.92	14.36	39.08
		Res.(%)	0.06	0.00	0.04	0.06	0.07	0.01	0.21	0.01	0.13	0.10
PACO-3Opt(2016)	0.40	Avg.	426.35	7542.00	677.85	539.85	1217.10	21326.80	630.55	14393.00	6601.40	29644.50
		SD.	0.49	0.00	0.99	1.09	4.01	33.72	2.63	19.76	15.01	53.43
		Res.(%)	0.08	0.00	0.42	0.34	0.50	0.21	0.25	0.10	1.12	0.94
PSO-ACO-3Opt(2015)	0.49	Avg.	426.45	7543.20	678.20	538.30	1227.40	21445.10	632.70	14379.15	6563.95	29646.05
		SD.	0.61	0.00	1.47	0.47	1.98	78.24	2.12	0.48	27.58	114.71
		Res.(%)	0.11	0.00	0.47	0.06	1.35	0.77	0.59	0.00	0.55	0.95
ACE(2015)	0.21	Avg.	426.82	7543.04	676.42	538.31	1213.29	21298.60	633.62	14385.50	6550.00	-
		SD.	0.58	13.37	2.69	1.14	4.12	42.46	3.90	26.22	13.61	-
		Res.(%)	0.19	0.01	0.21	0.06	0.19	0.08	0.73	0.05	0.34	-
ASA-GS(2011)	0.51	Avg.	428.87	7544.37	677.11	544.37	1219.49	21285.40	640.52	14383.00	6539.80	29438.40
		SD.	-	-	-	-	-	-	-	-	-	-
		Res.(%)	0.67	0.03	0.31	1.18	0.70	0.01	1.83	0.02	0.16	0.23
SA-ACO-PSO(2011)	0.52	Avg.	427.27	7542.00	-	540.20	-	21370.30	635.23	14406.37	6563.70	29738.73
		SD.	0.45	0.00	-	2.94	-	123.36	3.59	37.28	22.45	356.07
		Res.(%)	0.30	0.00	-	0.41	-	0.41	0.99	0.19	0.55	1.27
WFA-2Opt(2013)	0.52	Avg.	426.65	7542.00	-	541.22	-	21282.00	639.87	14739.00	6572.13	29646.50
		SD.	0.66	0.00	-	0.66	-	0.00	2.88	0.00	13.84	151.42
		Res.(%)	0.15	0.00	-	0.60	-	0.00	1.72	0.00	0.68	0.97
WFA-3Opt(2013)	0.66	Avg.	426.60	7542.00	-	539.44	-	21282.00	633.50	14459.40	6700.10	29646.50
		SD.	0.52	0.00	-	1.51	-	0.00	3.47	1.38	60.82	110.91
		Res.(%)	0.14	0.00	-	0.27	-	0.00	0.72	0.56	2.64	0.95
ACO-Taguchi(2013)	2.45	Avg.	435.40	7635.40	-	565.50	-	21567.10	655.00	14475.20	-	-
		SD.	-	-	-	-	-	-	-	-	-	-
		Res.(%)	2.21	1.24	-	5.11	-	1.34	4.13	0.67	-	-
IFOA(2017)	0.88	Avg.	427.53	7542.00	677.26	-	1237.20	21357.00	642.05	14427.06	6618.20	-
		SD.	1.26	0.00	2.33	-	15.17	43.77	4.77	44.67	31.68	-
		Res.(%)	0.36	0.00	0.34	-	2.16	0.35	2.08	0.33	1.38	-
DSOS(2017)	1.18	Avg.	427.90	7542.60	679.20	547.40	1228.37	21409.50	650.60	-	-	-
		SD.	1.20	0.00	2.80	3.90	14.32	149.15	4.57	-	-	-
		Res.(%)	0.45	0.01	0.62	1.75	1.43	0.60	3.43	-	-	-
C-PSO-ACO-Kopt(2017)	0.14	Avg.	426.29	7543.29	676.00	538.15	1213.90	21319.50	631.20	14379.29	-	29642.00
		SD.	0.46	3.90	1.73	0.65	0.99	47.79	1.50	1.30	-	145.00
		Res.(%)	0.07	0.01	0.14	0.02	0.07	0.17	0.17	0.00	-	0.46

best, *Avg*., and *Res*.. And the *Avg*(*Res*.) obtained by the MEATSP for large-scale TSP problems is the smallest. We sorted all the algorithm's *Avg*(*Res*.) discovered that the *Avg*(*Res*.) of MEATSP is less than 1/2 of the result of 1.45% of the next-closest algorithm, DPC-ACO-Kopt. It is much smaller than ASA-GS result 1.65% and PACO-3Opt result 2.4%.Therefore, it can be shown that MEATSP has a smaller residual in solving large-scale TSP problems.

C. RESULTS STATISTICS AND RUNTIME ANALYSIS

Based on the experimental results obtained in the previous subsection, we further analyze the differences between MEATSP and other algorithms from different statistical analysis methods.

1) STATISTICAL ANALYSIS

MEATSP's improvement over other algorithms can be described more comprehensively through statistics. Fig. 10 shows MEATSP's comparison with other algorithms on small-scale problems more intuitively. The *Avg*(*res*.) is the result of the *Res*. region average of the algorithm over all instances. In the figure, MEATSP has a very small *Avg*(*Res*.), which is 0.07% superior to DPC-ACO-KOpt and 0.17% superior to C-PSO-ACO-KOPT. From another

TABLE 3. Experimental results of MEATSP and other algorithms on large-scale TSP.

Algorithm	avg(Res.)	instance	rd400 (15281)	fl417 (11861)	pr439 (107217)	pcb442 (50778)	d493 (35002)	rat575 6773	p654 (34643)	d657 (48912)	u724 (41910)	rat783 (8806)
MEATSP	0.66	Avg.	15353.80	11861.00	107394.20	50989.80	35096.00	6818.00	34646.20	49239.60	42037.60	8883.60
		Best	15302.00	11861.00	107261.00	50942.00	35007.00	6804.00	34643.00	49160.00	41973.00	8865.00
		Res.(%)	0.48	0.00	0.17	0.42	0.27	0.66	0.01	0.67	0.30	0.88
DCP-ACO-KOpt(2018)	1.45	Avg.	15387.25	11890.15	107752.15	51158.95	35347.50	7135.00	34979.55	49317.65	42282.70	9192.10
		Best	15333.00	11870.00	107412.00	51001.00	35237.00	7071.00	34693.00	49103.00	42118.00	9118.00
		Res.(%)	0.70	0.25	0.50	0.75	0.99	5.34	0.97	0.83	0.89	4.38
PACO-3Opt(2016)	2.40	Avg.	15613.90	11987.40	108702.00	52202.40	35841.00	7012.40	35075.00	50277.50	41322.50	9127.30
		Best	15578.00	11972.00	108482.00	51961.00	35735.00	7003.00	35045.00	50206.00	42764.00	9111.00
		Res.(%)	2.18	1.07	1.39	2.81	2.40	3.53	1.25	2.79	2.89	3.65
PSO-ACO-3Opt(2015)	2.65	Avg.	15691.30	11980.40	108965.40	52368.10	35973.80	7018.60	35098.20	50475.50	43300.30	9138.10
		Best	15594.00	11947.00	108530.00	52131.00	35789.00	6987.00	35052.00	50291.00	43172.00	9128.00
		Res.(%)	2.69	1.01	1.63	3.13	2.78	3.63	1.31	3.20	3.32	3.77
HMMA(2015)	11.29	Avg.	16723.68	12897.84	116855.91	55670.17	38410.25	7745.63	37931.40	55554.31	47132.71	10256.23
		Best	16534.00	12543.00	114095.00	54401.00	37187.00	7575.00	37044.00	55163.00	46662.00	10149.00
		Res.(%)	9.44	8.74	8.99	9.63	9.74	14.36	9.49	13.58	12.46	16.47
PMSOM(2015)	5.75	Avg.	-	-	-	53362.60	-	-	36399.40	-	43980.35	-
		Best	-	-	-	52631.00	-	-	35953.00	-	43704.00	-
		Res.(%)	-	-	-	5.09	-	-	5.07	-	4.94	-
HCACO(2014)	8.58	Avg.	-	-	-	55326.00	-	-	-	53234.00	-	9505.00
		Best	-	-	-	54838.00	-	-	-	52665.00	-	9453.00
		Res.(%)	-	-	-	8.96	-	-	-	8.84	-	7.94
HGA(2014)	2.93	Avg.	15853.74	-	109249.66	52376.26	-	-	-	-	-	-
		Best		-	-	-	-	-	-	-	-	-
		Res.(%)	3.74	-	1.90	3.15	-	-	-	-	-	-
ASA-GS(2011)	1.65	Avg.	15429.80	12043.80	110226.00	51269.20	-	6904.82	-	-	42470.40	8982.19
		Best	15351.00	11904.00	110020.00	51064.00	-	6872.00	-	-	42275.00	8954.00
		Res.(%)	0.97	1.54	2.18	0.97	-	1.95	-	-	1.34	2.00
DSOS(2017)	4.22	Avg.	-	-	-	-	-	7117.32	-	-	-	9102.67
		Best	-	-	-	-	-	7073.00	-	-	-	9045.00
		Res.(%)	-	-	-	-	-	5.08	-	-	-	3.37

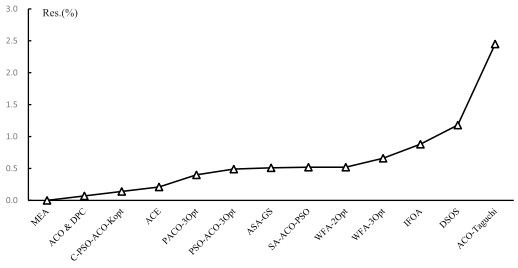


FIGURE 10. The avg(Res.) between MEATSP and other algorithms on small-scale TSP.

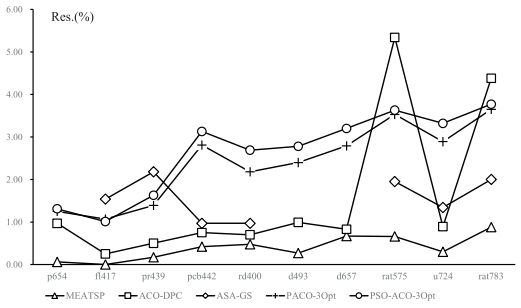


FIGURE 11. Average residual of MEATSP and other algorithms on large-scale TSP.

dimension, MEATSP's square deviation on each instance is also almost 0, which is better than other algorithms. It also shows MEATSP's stability in solving small-scale TSP.

The results of MEATSP compared with other algorithms shown in Fig. 11. For the DPC-ACO-Kopt algorithm which has the lowest average *Res*.. in Table 3 except MEATSP. It shows the *Res*. of each instance, and then sorts the results based on the error value of the MEATSP. It can be seen that, the results of all instances are generally smaller than DCP-ACO-Kopt's. In particular, there are two instances of rat types (rat575 and rat738) are much higher for DPC-ACO-Kopt. In our study, the *rat* sample has a large number of dense vertices, while the suboptimal algorithm has clustering operations, which makes it difficult to achieve excellent results in this type of data. Therefore, focus on another algorithm, ASA-GS, that performs well in each instance.

Also in Fig. 11, It can be seen that the MEATSP get lower *Res*. on each instance than the ASA-GS algorithm.

Among them, MEATSP get better result than ASA-GS on rat type data, which make other algorithms get higher results.

In order to further describe MEATSP's improvement effect on suboptimal algorithm in different scale TSP, we compare MEATSP with the suboptimal algorithm in Table 2 and Table 3 could be efficient. However, it can be seen from the table that different algorithms have their own advantages for different instances. In order to find the suboptimal algorithm in these instances, the algorithm is scored according to the result on each instance, the best gets 1 score, the second best gets 2 score, and so on. When the algorithm has no result on the corresponding instance, it gets the same score as the previous algorithm. The score of each algorithm is shown in Table 4 and Table 5. The lower the score of the algorithm is, the better algorithm is.

TABLE 4. Score results of MEATSP and other algorithms on small-scale TSP(best 6 only).

Algorithm	Score
MEATSP	10
DPC-ACO-KOpt	25
C-PSO-ACO-KOpt	51
PSO-ACO-3Opt	55
PACO-3Opt	57
ACE	60

TABLE 5. Score results of MEATSP and other algorithms on large-scale TSP(best 6 only).

Algorithm	Score
MEATSP	10
DPC-ACO-KOpt	28
PACO-3Opt	34
PSO-ACO-3Opt	42
ASA-GS	46
DSOS	62

In Table 4 and Table 5, except MEATSP, the algorithm with the best score is DPC-ACO-KOpt. *IE* represent MEATSP's improved efficiency of each instace on suboptimal algorithm.The calculation formula is shown in (7). Table 6 shows MEATSP's enhancement efficiency for DPC-ACO-KOpt for on small TSP instances, and Table 7 for large TSP instances.

$$IE = \frac{Avg_{sub-opt} - Avg_{MEATSP}}{Avg_{sub-opt}} \tag{7}$$

MEATSP comparison and algorithm DPC-ACO-KOpt have improved or equal results in each instance. For small-scale TSP instances, the maximum improvement was 0.13%, while for large-scale TSP, the maximum efficiency improvement was 4.44% of rat575 and 3.36% of rat783. According to Table 5, with the increase of vertices in the instance, MEATSP improves the result more efficiently than the suboptimal algorithm, And it proved that MEATSP can achieve better results on different types of TSP.

TABLE 6. MEATSP compared with suboptimal algorithm on small-scale TSP with improved efficiency.

Instances	eil51	berlin52	st70	eil76	rat99
IE(%)	0.06	0.00	0.04	0.06	0.07
Instances	**kroA100**	**eil101**	**lin105**	**ch150**	**kroA200**
IE(%)	0.01	0.09	0.01	0.13	0.00

TABLE 7. MEATSP compared with suboptimal algorithm on large-scale TSP with improved efficiency.

Instances	rd400	fl417	pr439	pcb442	d493
IE(%)	0.22	0.25	0.33	0.33	0.71
Instances	**rat575**	**p654**	**d657**	**u724**	**rat783**
IE(%)	4.44	0.95	0.16	0.58	3.36

2) RUNTIME ANALYSIS

In order to ensure MEATSP's fairness compared with other algorithms, the runtime of the algorithm is also an important standard to measure the algorithm. Based on the comparison results of other algorithms in Table 2, the runtime of the algorithm with excellent results is selected, as shown in Table 8. The runtime of MEATSP is the time it takes to directly output the result when M_{best} is already the optimal value of current instance.

TABLE 8. Runtime comparison on different instances(unit:second).

Instance	MEATSP	DPC-ACO-KOpt	ASA-GS	ACO-Taguchi	HMMA	PACO-3Opt
eil51	**1.17**	3.03	3.91	3.32	7.22	2.39
berlin52	**0.89**	1.48	3.83	3.15	8.19	2.10
st70	**1.22**	3.89	5.15	5.38	13.07	6.97
eil76	**1.32**	4.69	5.50	6.12	15.25	8.18
rat99	**2.76**	5.60	7.34	7.90	25.36	19.79
kroA100	**0.82**	6.34	7.14	11.52	25.81	21.10
eil101	6.85	**6.55**	7.42	20.75	26.00	20.79
lin105	**0.54**	5.65	7.68	21.45	27.92	14.57
ch150	6.75	**5.95**	10.91	33.28	55.75	79.35
kroA200	13.50	**9.46**	14.26	48.48	98.48	213.12
Average	3.58	5.26	7.31	16.14	30.31	38.84
Count	**7**	**3**	**0**	**0**	**0**	**0**

As can be seen from Table 8, although MEATSP does not always outperform other algorithms such as kroA200, ch150, and eil101 in the elapsed time of each instance. But the average time it takes to run on all instances is the smallest, it's 68% of the second-fastest algorithm. For some instances such as kroA100 and lin105, MEATSP is able to find the optimal solution in a very short time, thanks to its ability to search for results in very few iterations. *Count* in this table indicates the fastest output per algorithm, while MEATSP has seven instances that are the fastest of all.

$$T(n) = O(N \times (m \times (n^2 + n \times d + \lceil d^3 \rceil + r \times n^2))) \tag{8}$$

Let n be the size of TSP. Set population size of MEATSP to m, and d is offspring of each membranes during division. The r means the ratio of cytolysis in the selection and set the iterations as N. According to the four operators, computational complexity of MEATSP is calculated in formula is shown in (8). Except for the initialization phase, other

operators require linear time. N and m are constant, and d in general is small. Therefor the time complexity of MEATSP is $O(n^2)$.

In summary, MEATSP has the lowest residual on all of these instances. In addition, by comparing the results with all other algorithm, we found that MEATSP has better performance on different types of TSP, and some algorithms can achieve excellent results in some instance types but not all types. MEATSP is stable because the increase of residual on different instances is only related to the vertices and is not influenced by the type, it can be seen that the *Res*. of MEATSP for large-scale TSP problems will grow steadily and slowly as the problem grows. Therefore, the MEATSP has good stability in solving large-scale TSP problems, and its results are also excellent.

VI. CONCLUSION

The main work are the following two parts. Firstly, we proposed MEAF, which includes four operators derived from living cells: division, fusion, cytolysis and selection, and how these operators solve practical problems. MEAF is completely different from genetic algorithms, which are inspired by the activity of DNA, and membrane algorithms, which focus on how to combine different heuristic algorithms. MEAF relies on its own evolutionary operators to evolve the population and the objects in each membrane, so as to obtain the optimal solution.

Then we designed the MEATSP, which is an application of MEAF to the TSP problem to further illustrate the effectiveness and stability of MEAF in solving problems. It can be seen from the experimental results that MEATSP is superior to other algorithms in both small-scale and large-scale TSP. Based on the average residual, MEATSP is able to achieve the smallest residual on each instance. In terms of variance, MEATSP has the smallest square deviation on each instance. It is also an excellent performer on the runtime analysis. Therefore, the MEATSP is excellent and stable.

In addition, MEATSP also has its limitations. A common problem with heuristic algorithms is that when the algorithm is terminated, it is difficult to prove that its output is the optimal solution to current problem. A similar situation exists MEATSP. And when the current optimal value of the population approaches the optimal value of the problem, the regeneration rate of the whole population will be lower. How to make MEATSP more efficient in iteration, including improved population initialization and population repair is also worth studying in the future.

On the other hand, the algorithm MEATSP proposed in this article is only experimented on the test data set TSPLIB. It is also very meaningful to combine MEATSP with practical application problems.

REFERENCES

[1] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, "Erratum: The traveling salesman problem: A guided tour of combinatorial optimization," *J. Oper. Res. Soc.*, vol. 37, no. 6, p. 655, Jun. 1986.

[2] H. Hernández-Pérez and J.-J. Salazar-González, "A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery," *Discrete Appl. Math.*, vol. 145, no. 1, pp. 126–139, Dec. 2004.

[3] Y. Jiang, T. Weise, J. Lassig, R. Chiong, and R. Athauda, "Comparing a hybrid branch and bound algorithm with evolutionary computation methods, local search and their hybrids on the TSP," in *Proc. IEEE Symp. Comput. Intell. Prod. Logistics Syst. (CIPLS)*, Dec. 2014, pp. 148–155.

[4] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Oper. Res.*, vol. 46, no. 3, pp. 316–329, Jun. 1998.

[5] Ö. Ergun and J. B. Orlin, "A dynamic programming methodology in very large scale neighborhood search applied to the traveling salesman problem," *Discrete Optim.*, vol. 3, no. 1, pp. 78–85, Mar. 2006.

[6] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *Eur. J. Oper. Res.*, vol. 126, pp. 106–130, Oct. 2000.

[7] G. Gutin and A. P. Punnen, *The Traveling Salesman Problem and Its Variations*, vol. 12. Boston, MA, USA: Springer, 2007.

[8] L. Muyldermans, P. Beullens, D. Cattrysse, and D. Van Oudheusden, "Exploring variants of 2-Opt and 3-Opt for the general routing problem," *Operations Res.*, vol. 53, no. 6, pp. 982–995, Dec. 2005.

[9] A. Levin and U. Yovel, "Nonoblivious 2-Opt heuristics for the traveling salesman problem," *Networks*, vol. 62, no. 3, pp. 201–219, Oct. 2013.

[10] C.-B. Cheng and C.-P. Mao, "A modified ant colony system for solving the travelling salesman problem with time windows," *Math. Comput. Model.*, vol. 46, nos. 9–10, pp. 1225–1235, Nov. 2007.

[11] X. Geng, Z. Chen, W. Yang, D. Shi, and K. Zhao, "Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search," *Appl. Soft Comput.*, vol. 11, no. 4, pp. 3680–3689, Jun. 2011.

[12] Å. Gälca, M. Mahi, Ö. K. Baykan, and H. Kodaz, "A parallel cooperative hybrid method based on ant colony optimization and 3-Opt algorithm for solving traveling salesman problem," *Soft Comput.*, vol. 22, no. 5, pp. 1669–1685, Mar. 2018.

[13] M. Mahi, Ö. K. Baykan, and H. Kodaz, "A new hybrid method based on particle swarm optimization, ant colony optimization and 3-Opt algorithms for traveling salesman problem," *Appl. Soft Comput.*, vol. 30, pp. 484–490, May 2015.

[14] J. B. Escario, J. F. Jimenez, and J. M. Giron-Sierra, "Ant colony extended: Experiments on the travelling salesman problem," *Expert Syst. Appl.*, vol. 42, no. 1, pp. 390–410, Jan. 2015.

[15] I. Khan, M. K. Maiti, and M. Maiti, "Coordinating particle swarm optimization, ant colony optimization and K-opt algorithm for traveling salesman problem," in *Mathematics and Computing*, vol. 655, D. Giri, R. N. Mohapatra, H. Begehr, and M. S. Obaidat, Eds. Singapore: Springer, 2017, pp. 103–119.

[16] W.-T. Pan, "A new fruit fly optimization algorithm: Taking the financial distress model as an example," *Knowl.-Based Syst.*, vol. 26, pp. 69–74, Feb. 2012.

[17] L. Huang, G. C. Wang, T. Bai, and Z. Wang, "An improved fruit fly optimization algorithm for solving traveling salesman problem," *Frontiers Inf. Technol. Electron. Eng.*, vol. 18, no. 10, pp. 91–99, 2017.

[18] A. E.-S. Ezugwu and A. O. Adewumi, "Discrete symbiotic organisms search algorithm for travelling salesman problem," *Expert Syst. Appl.*, vol. 87, pp. 70–78, Nov. 2017.

[19] W. Yong, "Hybrid Max–Min ant system with four vertices and three lines inequality for traveling salesman problem," *Soft Comput.*, vol. 19, no. 3, pp. 585–596, Mar. 2015.

[20] E. Liao and C. Liu, "A hierarchical algorithm based on density peaks clustering and ant colony optimization for traveling salesman problem," *IEEE Access*, vol. 6, pp. 38921–38933, 2018.

[21] M. Alipour, S. N. Razavi, M. R. Feizi Derakhshi, and M. A. Balafar, "A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem," *Neural Comput Appl.*, vol. 30, no. 9, pp. 2935–2951, Nov. 2018.

[22] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[23] M. M. Alipour and S. N. Razavi, "A new local search heuristic based on nearest insertion into the convex hull for solving Euclidean TSP," *Int. J. Oper. Res.*, vol. 34, no. 3, p. 409, 2019.

[24] G. Păun, "Computing with membranes," *J. Comput. Syst. Sci.*, vol. 61, no. 1, pp. 108–143, 2000.

[25] H. Peng, J. Wang, M. J. Pérez-Jiménez, and A. Riscos-Nñez, "Dynamic threshold neural p systems," *Knowl.-Based Syst.*, vol. 163, pp. 875–884, Jan. 2019, doi: 10.1016/j.knosys.2018.10.016.

[26] H. Peng and J. Wang, "Coupled neural p systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 6, pp. 1672–1682, Jun. 2019, doi: 10.1109/TNNLS.2018.2872999.
[27] T. Nishida, "Membrane Algorithms," in *Proc. Int. Workshop Membrane Comput.*, vol. 3850, 2005, pp. 55–66.
[28] A. Leporati and D. Pagani, "A membrane algorithm for the min storage problem," in *Membrane Computing* (Lecture Notes in Computer Science), vol. 4361, H. J. Hoogeboom, G. Păun, G. Rozenberg, and A. Salomaa, Eds. Berlin, Germany: Springer, 2006, pp. 443–462.
[29] G.-X. Zhang, C.-X. Liu, and H.-N. Rong, "Analyzing radar emitter signals with membrane algorithms," *Math. Comput. Model.*, vol. 52, nos. 11–12, pp. 1997–2010, Dec. 2010.
[30] J. Cheng, G. Zhang, and X. Zeng, "A novel membrane algorithm based on differential evolution for numerical optimization," *Int. J. Unconventional Comput.*, vol. 7, no. 3, pp. 159–183, 2011.
[31] G. Zhang, "A novel membrane algorithm based on particle swarm optimization for solving broadcasting problems," *J. Universal Comput. Sci.*, vol. 18, no. 13, pp. 1821–1841, 2012.
[32] J. He, J. Xiao, and Z. Shao, "An adaptive membrane algorithm for solving combinatorial optimization problems," *Acta Math. Sci.*, vol. 34, no. 5, pp. 1377–1394, Sep. 2014.
[33] G. Sierksma, "Hamiltonicity and the 3-Opt procedure for the traveling salesman problem," *Appl. Math.*, vol. 22, no. 3, pp. 351–358, 1993, doi: 10.4064/am-22-3-351-358.
[34] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, Jun. 2014.
[35] G. Păun, "Computing with membranes: Attacking NP-complete problems," in *Unconventional Models Computer*. London, U.K.: Springer, 2001, pp. 94–115.
[36] G. Zhang, M. Gheorghe, L. Pan, and M. J. Pérez-Jiménez, "Evolutionary membrane computing: A comprehensive survey and new results," *Inf. Sci.*, vol. 279, pp. 528–551, Sep. 2014.
[37] G. Zhang, M. J. Pãrez-Jimãnez, and M. Gheorghe, *Real-Life Applications With Membrane Computing*, vol. 25. Cham, Switzerland: Springer, 2017.
[38] S.-M. Chen and C.-Y. Chien, "A new method for solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques," in *Expert Syst. Appl.*, vol. 38, no. 12, pp. 14439–14450, Nov. 2011, doi: 10.1016/j.eswa.2011.04.163.
[39] Z. Ali Othman, N. H. Al-Dhwai, A. Srour, and W. Yi, "Water flow-like algorithm with simulated annealing for travelling salesman problems," *Int. J. Adv. Sci., Eng. Inf. Technol.*, vol. 7, no. 2, p. 669, Apr. 2017, doi: 10.18517/ijaseit.7.2.1837.
[40] M. Peker, B. Åen, and P. Y. Kumru, "An efficient solving of the traveling salesman problem: The ant colony system having parameters optimized by the taguchi method," *TURKISH J. Electr. Eng. Comput. Sci.*, vol. 21, pp. 2015–2036, 2013.
[41] B. Avåar and D. E. Aliabadi, "Parallelized neural network system for solving Euclidean traveling salesman problem," *Appl. Soft Comput.*, vol. 34, pp. 862–873, Sep. 2015, doi: 10.1016/j.asoc.2015.06.011.
[42] J. Jiang, J. Gao, G. Li, C. Wu, and Z. Pei, "Hierarchical solving method for large scale TSP problems," in *Advances in Neural Networks*, vol. 8866, Z. Zeng, Y. Li, and I. King, Eds. Cham, Switzerland: Springer, 2014, pp. 252–261.
[43] Y. Wang, "The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem," *Comput. Ind. Eng.*, vol. 70, pp. 124–133, Apr. 2014, doi: 10.1016/j.cie.2014.01.015.

PING GUO (Member, IEEE) was born in Meishan, Sichuan, China, in 1963. He received the Ph.D. degree in computer software and theory from Chongqing University, Chongqing, China, in 2004. He is currently a Professor with the Chongqing Key Laboratory of Software Theory and Technology, College of Computer Science, Chongqing University. His research interests include different aspects of artificial intelligence and biological computing. He has authored/coauthored more than 150 refereed publications.

MENGLIANG HOU was born in Qionglai, Sichuan, China, in 1994. He received the bachelor's degree in computer science and technology from Chongqing University, Chongqing, China, in 2017. He is currently a Graduate Student with the College of Computer Science, Chongqing University. His research interests include heuristic algorithm and biological computing.

LIAN YE was born in Chongqing, China, in 1980. She received the Ph.D. degree in computer science from Chongqing University, China, in 2012. She is currently a Teacher with Chongqing University. Her research interests include the different aspects of artificial intelligence, artificial immune systems, and membrane computing.