IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 4, 2019, accepted April 30, 2019, date of publication May 8, 2019, date of current version May 21, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2915550

MEAMVC: A Membrane Evolutionary Algorithm for Solving Minimum Vertex Cover Problem

PING GUO[1,2], CHANGSHENG QUAN[1], AND HAIZHU CHEN[3]
[1]College of Computer Science, Chongqing University, Chongqing 400044, China
[2]Chongqing Key Laboratory of Software Theory and Technology, Chongqing 400044, China
[3]Department of Software Engineering, Chongqing College of Electronic Engineering, Chongqing 401331, China

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

This work was supported by the Science and Technology Research Program of Chongqing Municipal Education Commission under Grant KJZD-K201803101.

ABSTRACT Since the membrane algorithm was proposed, it has been used for many optimization problems such as, traveling salesman problem, the knapsack problem, and so on. In membrane algorithms, the membranes have two functions: container and comparator. As a container, each membrane contains one evolutionary algorithm like genetic algorithm and ant colony algorithm. These algorithms are called sub-algorithms and used to evolve individuals. As a comparator, the membrane will compare the results of sub-algorithms, and select the best as the base of the next evolvement. This paper proposes a novel evolutionary algorithm called membrane evolutionary algorithm framework (MEAF). Unlike the presented membrane algorithms, the membranes in MEAF will be evolved to solve problems by using four operators that are abstracted from the life cycle of living cells. Based on MEAF, a membrane evolutionary algorithm called MEAMVC is proposed to solve the minimum vertex cover (MVC) problem. The experimental results show the advantages of MEAMVC when MEAMVC is compared with two state-of-the-art MVC algorithms proposed in recent years.

INDEX TERMS Evolutionary algorithm, membrane computing, membrane evolutionary algorithm framework, minimum vertex cover problem.

I. INTRODUCTION

Membrance Computing is a fast-growing branch of natural computing, which abstracts the distributed parallel computing models from the architecture and functioning of the living cells. The computing systems, which are built based on membrane computing models for solving problems, are called P systems. Since membrane computing was introduced, many P systems have been proposed [1]–[6]. These P systems can be classified into three kinds: cell-like, tissue-like and neural-like P systems. In the compartments of them, objects evolve according to given rules in a synchronized, nondeterministic, maximally parallel manner. It has been proven that most of the P systems have the same ability with Turing Machine [1]–[3]. By using P systems, many NP-hard problems can be solved in polynomial time in theoretical level, such as knapsack problem [7], [8], Hamiltonian cycle problem [9], maximum clique problem [10], All-SAT problem [11], and so on. In addition, P systems have many real-life applications, such as image processing [12] and fault propagation paths modeling [13].

In 2006, a membrane algorithm combined with a cell-like P-system structure and two approximate algorithms was proposed in [14]. In this membrane algorithm, each inner membrane runs a different evolutionary algorithm to obtain or improve solutions of the optimization problem. Through this kind of combinations, the travelling salesman problem [14] and min storage problem [15] have been solved and yielded a better result when compared with applying the approximate algorithms alone.

Various variants of the membrane algorithm have been proposed so far. In [16], [17], Zhang *et al.* combined the hierarchical structure with the quantum-inspired evolutionary algorithms (QIEAs) to solve the knapsack problem and analyse the radar emitter signals. Combining the hierarchical structure and a local search algorithm, Cheng et al. proposed

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

a novel membrane algorithm based on differential evolution for numerical optimization in [18]. Another novel membrane algorithm based on the hierarchical structure and particle swarm optimization (PSO) solved the broadcasting problems in [19]. In [20], an adaptive membrane algorithm combining the hierarchical structure and local search was proposed to solve the travelling salesman problem.

The netted structure of tissue-like P systems has also been used as a part of membrane algorithms. In [21], the netted structure was combined with differential evolution algorithms to solve constrained manufacturing parameter optimization problems. In [22], the netted structure was combined with a quantum-inspired evolutionary algorithm to solve the distribution network reconfiguration problem. In [23], the netted structure was combined with a particle swarm optimization local search algorithm to solve constrained optimization problems.

Membrane algorithms combine membrane structure with other algorithms like QIEAs, PSO, and local search. Their core computing is performed by other algorithms, the membrane acts as a container and compares the computational results, without participating in the evolution process of objects. This paper proposes a novel evolutionary algorithm framework called Membrane Evolutionary Algorithm Framework (MEAF for short). MEAF abstracts four characteristics from the life cycle of living cells which forms its core operators: division, fusion, cytolysis, and selection. Each membrane operator allows the membranes to evolve their multisets. In MEAF, the membranes evolve iteratively for a number of times, and the best one in the ending configuration is the solution to the solving problem. Based on MEAF, this paper also proposes a membrane evolutionary algorithm called MEAMVC in solving the minimum vertex cover problem. Meanwhile, the results of MEAMVC are compared with FastVC [24] and FastVC2 [25] which are two of the state-of-art algorithms proposed in recent years.

The rest of this paper is organized as follows. In section II, some related works about membrane algorithms and the minimum vertex cover problem are introduced. Then, MEAF is introduced in section III, including its operators and algorithm framework. Base on MEAF, MEAMVC is designed to solve the minimum vertex cover problem in section IV. Section V gives the experiment results on large real-world graph data set. Finally, some conclusions are given in section VI.

II. RELATED WORKS

Firstly, we briefly introduce the membrane algorithm and its application proposed. After that, we review the studies of the MVC problem in order to compare the results of this paper.

A. MEMBRANE ALGORITHMS

Membrane algorithms provide a natural framework for approximate algorithms. Moreover, the paradigm of P systems can bring to the approximate algorithms area ''space'' and ''time'' varying strategies.

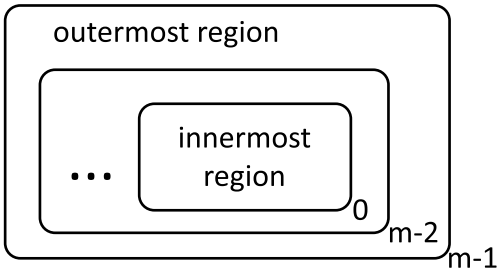


FIGURE 1. One kind of membrane structure of membrane algorithms.

As shown in Fig. 1, the membrane structure of the membrane algorithm [14] consists of a set of layered membranes. For the regions separated by nested membranes, each of them contains a subalgorithm and a few tentative solutions. And for two adjacent regions, they will exchange solutions for communication.

Algorithm 1 Membrane Algorithms

```
Initialization;
repeat
    for each region do
        Update the tentative solutions by using the
        associated subalgorithm;
        Send the best and worst solutions to the adjacent
        inner and outer regions, respectively;
        Select the next population from the solutions in
        the region by using the associated subalgorithm;
until the terminate conditions are satisfied;
return the best solution in the innermost region;
```

In membrane algorithms, the algorithm associated with each region might be identical or different algorithms. For example, in [14], when to solve the travelling salesman problem, the innermost region uses a tabu search, and the other regions use an algorithm which resembles genetic algorithm. In each iteration, the solutions in each region will be improved by the associated subalgorithm, and the best solution will be sent to the inner adjacent region, and the worst solution will be sent to the outer adjacent region. In [15], when to solve the min storage problem, all the regions use the same local search algorithm called LocalSearch4MS. And in each iteration, the best and the worst solution will be sent to the inner adjacent and outer adjacent region, respectively.

B. MINIMUM VERTEX COVER PROBLEM

The MVC problem is a classic combinatorial optimization problem. It has many important applications in scheduling, VLSI design, industrial machine assignment, and network security. Given an undirected graph $G(V, E)$, the target of the MVC problem is to find a minimum sized subset $V' \subseteq V$, so that for every edge $e \in E$ at least one endpoint of e belongs to V'. It's been proved that the MVC problem is NP-hard to approximate within a factor of 1.3606 [26].

Many exact algorithms and approximate algorithms have been proposed to solve the MVC problem. Branching algorithms are the main exact algorithms in solving MVC, such as branch-and-bound methods and LP-based branch-and-cut methods [27]–[29]. For many large instances, exact algorithms cannot get a solution in a reasonable time. So, approximate algorithms play an important role in solving large instances, including ant colony algorithms [30], [31] (for weighted vertex cover), genetic algorithms [32], [33], local search algorithms [24], [25], [34], memetic algorithms [39] (for partial vertex cover), rough set based algorithms [38] (for hypergraphs) and so on. Among these algorithms, local search algorithms are superior to other approximation algorithms. For large instances, the state-of-art local search algorithms mainly include FastVC [24] and FastVC2 [25]. This paper compares the results with FastVC and FastVC2.

Algorithm 2 is the algorithm framework of FastVC and FastVC2. FastVC and FastVC2 solve MVC by iteratively exchanging vertices in the tentative solution, and try to find a vertex cover near the tentative solution. The difference between FastVC and FastVC2 is that FastVC2 use three construction algorithms to construct the initial solution, while FastVC only use one construction algorithm.

Algorithm 2 Framework of FastVC and FastVC2

Input: graph $G = (V, E)$, the cutoff time
Output: vertex cover of G

```
C = a constructed vertex cover;
while elapsed time< cutoff do
    if C covers all edges then
        Update the current optimal C* to C;
        Remove a vertex from C;
    else
        Choose and remove a vertex u from C;
        Choose and add a vertex v to C;
return C*;
```

The construction algorithm used in FastVC is called EdgeGreedyVC. Starting with an empty set C, for each uncovered edge e, add one endpoint v of e with higher degree to C. When all the edges are covered, remove all the redundant vertexes in C. When removed these vertexes, C remains a cover. FastVC2 use EdgeGreedyVC and two other construction algorithm called GreedyVC and MatchVC. When constructing a cover, GreedyVC iteratively adds the vertex to C which has the most adjacent uncovered edges. When constructing a cover, MatchVC iteratively chooses one of the uncovered edges and adds the two endpoints of the chosen edge to C.

III. MEMBRANE EVOLUTION ALGORITHM FRAMEWORK

Being inspired from the characteristics of biological individuals and groups is the basis of biological computing models or frameworks. For example, genetic algorithms are inspired from the genetic evolution of organisms, which forms a framework with three operators: crossover, mutation and selection. In this paper, the proposed membrane evolution algorithm framework is a framework of evolutionary algorithm, which consists of four evolution operators: fusion, division, cytolysis and selection.

A. EVOLUTION OPERATORS

1) FUSION

In biological systems, the merging process of cells is called cell fusion. The computing process of merging the objects of two membranes into one new membrane is called fusion operator as shown in Fig. 2.

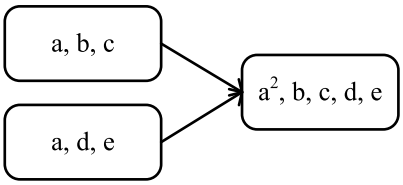


FIGURE 2. Fusion operator.

2) DIVISION

The process of dividing one cell into at least two cells is called cell division. The computing process of splitting objects of one membrane into two new membranes using certain rules is called division operator. The process is shown in Fig. 3.

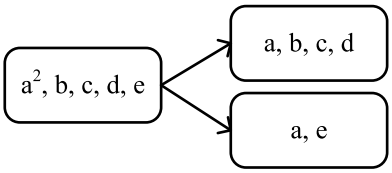


FIGURE 3. Division operator.

3) CYTOLYSIS

In nature, when the membrane of a cell loses its biological functions, the substances will scatter or dissolve in water. This phenomenon is called cytolysis. Correspondingly, the process of consuming all the objects in one membrane and deleting the membrane is called cytolysis operator.

4) SELECTION

The cells that adapt to the environment will have more chances to be selected to multiply themselves in biological systems. The multiplying process of highly adaptable membranes is called selection operator. Selection operator will multiply the highly adaptable membranes, which have high fitness. The multiplied membranes will enter the next generation.

B. ALGORITHM FRAMEWORK

According to the four evolution operators, the framework of MEAF is shown as Algorithm 3.

Algorithm 3 Membrane Evolutionary Algorithm Framework (MEAF)

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

(1) **Initialization** *create membrane structure and construct the initial population* ;
repeat
  (2) **Evolve** *membranes in parallel***:**
    1) Fusion operator: merge membranes which are selected by the correlation between membranes or randomly;
    2) Division operator: divide membranes which are selected by their fitness or randomly. The objects of a membrane can be divided into two new membranes randomly or according the correlation between objects;
    3) Cytolysis operator: dissolve membranes randomly or according to their fitness;
    4) Selection operator: multiply the membranes with high fitness;
  (3) **Repair** *the population***:**
    1) Keep stability and variety of the population;
    2) Repair membranes to satisfy the needs of questions;
until *the end conditions are satisfied*;
(4) **return** solutions;

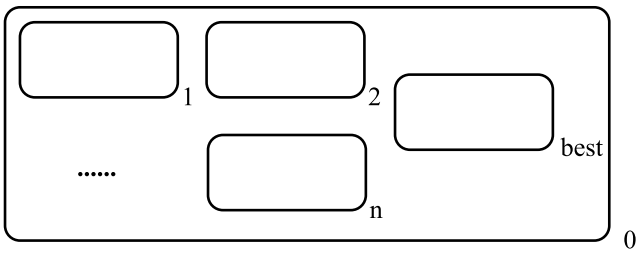


FIGURE 4. The initial membrane structure of a membrane evolution algorithm.

Fig. 4 shows the initial membrane structure of MEAF. The membranes labeled 1 to n constitute the population of MEAF which is denoted as $P = \{M_1, \ldots\ldots, M_n\}$, where n is the size of the population, and the membrane labeled *best* is a duplicate of the current best membrane in P. $\forall M_i \in P$ is one of the individuals in the population, and the objects in M_i form a feasible solution to the problem being solved. In each iteration, MEAF will evolve the population in parallel and produce the next generation population, where the size and diversity of the population will be maintained.

In the evolution process, the calculation of individual's fitness (membrane's fitness) is related to the solving problems. For example, for MVC problem, an individual's fitness is calculated according to how close this individual to a smaller vertex cover. And the end conditions are also related to the problem to be solved.

For one specific problem, four operators can be used alone or in combination. For example, we can combine the selection operator with cytolysis operator, or combine the division operator with the fusion operator. In section IV, we implement the membrane evolution algorithm framework to solve the MVC problem.

When to compare MEAF with membrane algorithms and other evolutionary algorithms, we can see the differences are:

(1) Differences from membrane algorithms (see subsection II-A): first, the membranes $M_1, \ldots, M_n$ in MEAF form the population, and the whole population evolve through iterations. Hence, MEAF itself is an evolution framework rather than just using membranes as containers. Second, solutions of the target problem can be obtained by evolving membranes and their inner objects through the four operators. The individuals can be created by fusion, division and selection. And each membrane can be fused, divided, and dissolved. The objects inside membranes (or sub-membranes) will be reallocated through the evolvement of membranes.

(2) Differences from genetic algorithms: MEAF needn't to encode individuals. When we use the operators of MEAF, the fitness of individuals can be computed, compared and improved. Besides, we can also improve the efficiency of evolvement by averaging the fitness of objects or object set.

(3) Differences from ant colony algorithms: the membranes and the objects in MEAF both carry heuristic information. These heuristic information is much more abundant than the heuristic information carried by the path (the path will be chosen by ants), which leads to a faster speed and a better solution of the problem to be solved.

Section IV describes the algorithm MEAMVC which is an application of MEAF. Section V gives the results of MEAMVC, and compares and analyzes the results with other MVC algorithms.

IV. SOLVING MVC BASED ON MEAF

In this section, we propose a membrane evolution algorithm to solve MVC, which is named MEAMVC (membrane evolution algorithm for MVC problem).

A. DEFINITIONS AND NOTATION

In MEAMVC, the membrane structure is shown in Fig. 4. Each object o in M_i ($1 \leq i \leq n$) represents one vertex in graph $G = (V, E)$. That vertex o is in M_i is denoted as $o \in M_i$, and the size of M_i is denoted as $|M_i|$. Obviously, the object set of M_i is one tentative solution of MVC. In the rest of the paper, we use $CE(M_i)$ to represent the set of edges of G which are covered by the objects in M_i, and we use $|CE(M_i)|$ to represent the size of $CE(M_i)$. $M_i - o$ denotes the membrane formed by the remaining objects after removing the object o from the membrane M_i, and $M_i + o$ denotes the membrane by adding the object o into the membrane M_i.

We use (1) to evaluate the fitness of object o to membrane M_i, which is denoted as $fit(o, M_i)$.

$$fit(o, M_i) = \begin{cases} |CE(M_i)| - |CE(M_i - o)|, & o \in M_i \\ |CE(M_i + o)| - |CE(M_i)|, & o \notin M_i \end{cases} \quad (1)$$

From (1), the fitness of object o with respect to M_i is the size of the new uncovered edges after removing object o from M_i (for $o \in M_i$) or the size of the new covered edges after adding object o to M_i (for $o \notin M_i$). For $u, v \in V$, if $fit(u, M_i) > fit(v, M_i)$, then u is more suitable for staying in M_i. Moreover, the state change of object o, in M_i or not in M_i, will affect the fitness value of its neighbors which have an edge with o in graph G. Therefore, when the state of the neighbor is changed, the vertex fitness should be dynamically updated.

For two membranes M_i and M_j in the population, the fitness of M_i to M_j is defined as (2).

$$fit(M_i, M_j) = \sum_{o \in M_i \text{ and } o \notin M_j} fit(o, M_j) \quad (2)$$

The fitness value $fit(M_i, M_j)$ shows the improvements of M_j if we fuse M_i and M_j. The improvements of M_i is represented by $fit(M_j, M_i)$, which might be different with $fit(M_i, M_j)$.

Equation (1) and (2) are used by division and fusion operators respectively, which focus much more on the fitness between membranes and objects. For cytolysis and selection operators, they focus much on the fitness of the membrane itself. For a membrane M_i, the fitness of M_i is denoted as $fit(M_i)$, which is defined as (3). And in (3), we use $UE(M_i)$ to represent the set of edges in G which is not covered by M_i. $Fit(M_i)$ is related to the worst case of M_i to become a cover of G by adding one endpoint of each uncovered edge. Obviously, the membrane with larger fitness value is more adaptive to the environment.

$$fit(M_i) = -1 \times (|M_i| + |UE(M_i)|) \quad (3)$$

B. MEAMVC ALGORITHM

This section introduces the algorithm of MEAMVC for solving the MVC problem. MEAMVC solves the MVC problem by iteratively performing fusion, division, cytolysis, and selection operators. In each iteration, MEAMVC will continuously solve the problem that if there exists a little smaller cover than the current best.

The algorithm of MEAMVC is described in Algorithm 4. At the beginning, the initial population is constructed by the ConstructPopulation function (see Algorithm 5). After that, the smallest membrane in the population is chosen as the currently optimal (called *best* in the rest of the paper). The fitness of each object in each membrane is automatically updated when their states or the states of their neighbors are changed. Until the maximum runtime (MRT for short) being reached, MEAMVC will iteratively try to find a smaller cover for graph G.

Algorithm 4 MEAMVC

Input: graph $G = (V, E)$, population size PS, the maximum runtime MRT, cytolysis and selection period CSP
Output: a cover of G

```
(1) Initialization
    P = ConstructPopulation(PS);        // see IV-C
    Construct two membranes by using MatchVC and
    EdgeGreedyVC ;                      // see II-B
    Add the two membranes to P;
    Remove the worst two membranes in P;
    best = the smallest membrane in P;
repeat
    (2) Evolve membranes in parallel:
        division(P);                    // see IV-D
        fusion(P);                      // see IV-E
        if the elapsed time from last update of best >
        CSP then
            CytolysisAndSelection(P);   // see IV-F
    (3) Repair the population:
        for ∀ M ∈ P do
            while M uncovers G and |M|<|best|-1 do
                add one object o ∉ M to M with
                maximal fitness value
            while |M| ≥ |best| do
                remove one object from M with minimal
                fitness value;
            if M covers G then
                best = M;
                remove one object from M with
                minimum fitness value;
until elapsed time >= MRT;
(4) return best;
```

At the start of each iteration, the size of each membrane will be kept under $|best|$ by removing the objects with lower fitness values (note that the objects with higher fitness values are more suitable to stay in a membrane). If the size of membrane is not under $|best|$, we will have less possibility to find a better solution than *best*, and have more chances to find a solution which is of equal or bad quality when compared to the *best*. After that, we can update the *best* if these exists one membrane which is a cover of G.

After the initializing of each iteration, the four operators will be performed on the population, which will be described in the following subsections. Note that we perform the selection and cytolysis operators periodically, which are used to avoid the local optimal. The period is a parameter called *CSP* in MEAMVC. After the four operations, some of the membranes might haven't performed the fusion operator. Therefore, we need to add some objects to them

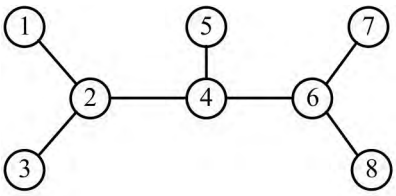


FIGURE 5. An example of the environment definition.

(lines 14-17), so that they can keep their size to $|best| - 1$. This process is simple. If one membrane isn't a cover of G and its size is under $|best| - 1$, we will continually add one endpoint with greater fitness value of a randomly selected uncovered edge of this membrane. If two endpoints of the selected edge have the same fitness value, the vertex with larger time distance from now to the last time its state changed will be selected.

We use the graph in Fig. 5 (denoted as G) to illustrate the procedure of MEAMVC. Fig. 6 shows how MEAMVC works in one loop (Algorithm 4, line 7-22) based on G with parameter $NS = 2$(see subsection IV-D) and parameter $N = 1$(see subsection IV-F). At the beginning of this loop, we assume the initial states of the population P and the membrane *best* are as shown in 6(a). In the following, we use $M_{x(y)}$ to represent the membrane x in subfigure y.

1) DIVISION

MEAMVC will firstly perform the division operator on P. According to equation (1), the fitness values of the objects in $M_{1(a)}$ are 3, 1, 1, and 1 respectively, and in $M_{2(a)}$, the fitness values are 2, 0, 0, and 0 respectively. For each membrane in P, division operator will separate the two objects (for $NS = 2$) with lowest fitness values and the rest objects. For $M_{1(a)}$,there are three objects which have a lowest fitness value 1. Consequently, there are totally three division situations. Because of the randomness in the algorithm, these three situations can be true. Here, we assume that $M_{1(a)}$ is divided to $M_{1(b)} = \{2, 5\}$ and $M_{1'(b)} = \{7, 8\}$. For the same reason, we can assume that $M_{2(a)}$ is divided to $M_{2(b)} = \{4, 7\}$ and $M_{2'(b)} = \{6, 8\}$. According to (4), $\mathrm{T}(M_{1(b)}) = 2$, $\mathrm{T}(M_{2(b)}) = 2$. We use P' to represent the population formed by membrane $1'$ and $2'$.

2) FUSION

Each membrane in P will randomly choose one membrane in P', and try to fuse the chosen membrane. Because of the randomness, we assume $M_{1(b)}$ chooses $M_{2'(b)}$ and $M_{2(b)}$ chooses $M_{1'(b)}$. $M_{1(b)}$ will successfully fuse $M_{2'(b)}$ for $fit(M_{2'(b)}, M_{1(b)}) = 4 > T(M_{1(b)})$. $M_{2(b)}$ will fail to fuse $M_{1'(b)}$ for $fit(M_{1'(b)}, M_{2(b)}) = 1 \leq T(M_{2(b)})$. After fusion is finished, P' will be deleted.

3) REPAIR THE POPULATION

After fusion, if the time interval between the current time and the time when last update the *best* is not greater than the given parameter *CSP*, MEAMVC will repair the population. $M_{2(c)}$ doesn't cover G, therefore it will be repaired to $M_{2(d)} = \{4, 7, 2\}$. $M_{1(c)}$ is a cover of G and smaller than best, therefore $M_{best(d)} = M_{1(c)}$, and $M_{1(c)}$ will become $M_{1(d)} = \{2, 5, 6\}$ by removing one object.

4) CYTOLYSIS AND SELECTION

If the time interval between the current time and the time when last update the *best* is greater than the given parameter *CSP*, MEAMVC will perform cytolysis and selection before repairing the population. In this step, all the objects in $M_{2(c)}$ will be replaced by $|best| - 1$ objects selected from *best* randomly. We can assume after the replacement, $M_{2(c)} \rightarrow M_{2(e)} = \{2, 6, 7, 8\}$. After cytolysis and selection, MEAMVC will also repair the population.

This loop is ended after MEAMVC finishing repairing the population. The next loop will also start from division.

C. INITIALIZATION

The quality of the initial population has great impact on the solution quality of the MVC problem, especially in large instances. MEAMVC uses three construction algorithms, MatchVC, EdgeGreedyVC (see subsection II-B), and ConstructPopulation. After all the membranes are constructed, remove two worst from P, and select the smallest membrane as *best*. ConstructPopulation is a new construction algorithm described in Algorithm 5. The main idea of ConstructPopulation is trying to construct membrane by iteratively adding vertex not belongs to the membrane with highest fitness value. If two vertices have the same fitness value, the vertex with fewer occurrences in the constructed membranes will have the priority to be considered.

Algorithm 5 ConstructPopulation

Input: graph $G = (V, E)$, population size PS
Output: the membrane population (P for short)

```
for each vertex v in G do
    count[v] = 0;
for i = 1 to PS do
    M = an empty membrane;
    while M isn't a cover of G do
        add the vertex v ∉ M with maximal fit(v, M),
        breaking ties in favor of the vertex with smaller
        count value;
        count[v] = count[v] + 1;
    remove the vertices from M whose fitness values
    equal to 0;
    update the count value of the removed vertices;
    add M to P;
return P;
```

At the beginning of ConstructPopulation, the occurrence counting value of each vertex is initialized to 0. In each iteration of constructing a new membrane M, ConstructPopulation will iteratively add the vertex $v \notin M$ which has greater

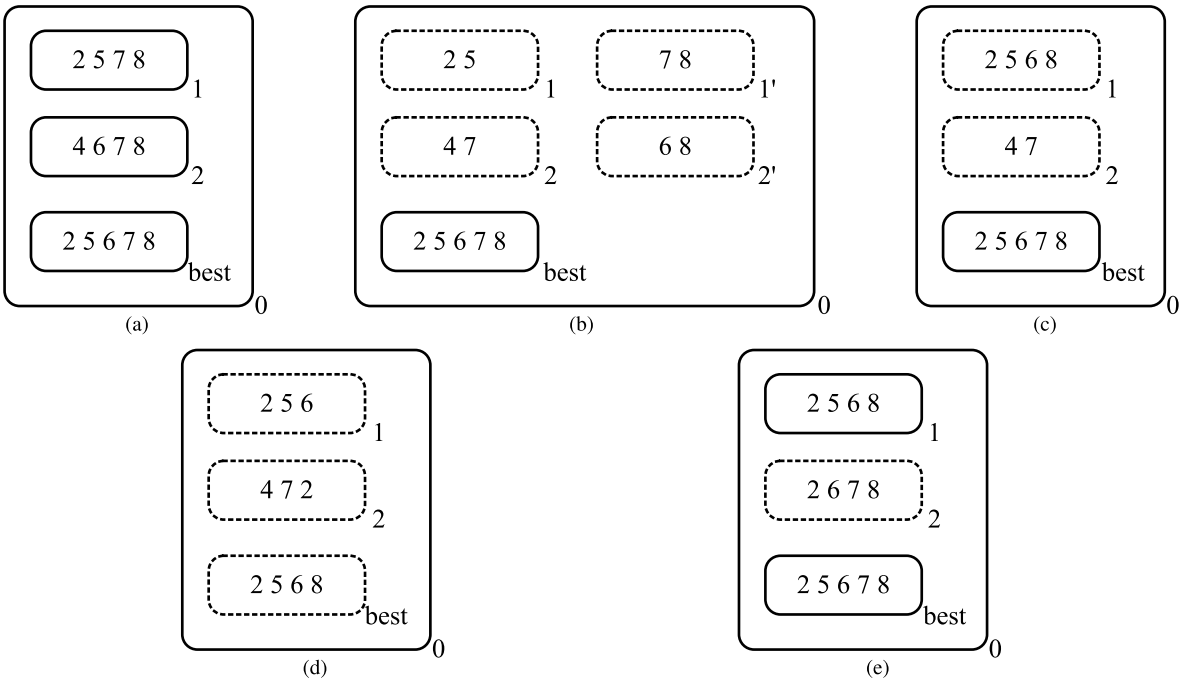


FIGURE 6. An example of the procedure of MEAMVC. (a) Initial state. (b) After division. (c) After fusion. (d) After repairing the population. (e) After cytolysis and selection.

fitness value and smaller counting value until M covers the graph G. Before finishing constructing M, M will be shrunk by removing vertices whose fitness values equal to 0. During the process, the occurrence counting values will be automatically updated.

The complexity of ConstructPopulation: the direct implementation of ConstructPopulation is $O(PS \times |V|^2)$, because of the two nested loops for the construction of each membrane (lines 5-7). The direct implementation is too time-consuming for large instances. A smarter implementation is to use a priority tree to manage the vertices not belongs to the membrane under construction. The vertices in the tree are in the order of their fitness values and counting values, where the fitness value is the first key, and the counting value is the second key. By doing this, the complexity of ConstructPopulation can be reduced to $O(PS \times |V| \times log|V|)$.

D. DIVISION

The division operator will select some objects with lower fitness values in each membrane to form new membranes. The size of the new membranes is a user defined parameter called NS. In order to selecting objects quickly, we use the idea of BMS strategy [24] that has a time complexity of O(1).

The detailed implementation is proposed in Algorithm 6. For each membrane, NS sized objects with lower fitness values will be selected and removed from them by using the BMS strategy with a parameter k (lines 5-7). The removed objects of each membrane will form a new small membrane which will be stored in P'. And P′ will be used for the fusion operator.

For two membranes M and M', which were divided from the same membrane, the fusion threshold of M is defined as $T(M)$, which is defined in (4).

$$T(M) = \sum_{o \in M'} fit(o, M) \tag{4}$$

Algorithm 6 Division

Input: graph $G = (V, E)$, the size of the new membranes NS

```
1  for ∀M ∈ P do
2      create a new membrane M′;
3      for j = 1 to NS do
4          bestv = randomly selected vertex ∈ M;
5          for m = 1 to k do // BMS strategy
6              randv = randomly selected vertex ∈ M if
               fit(bestv, M) > fit(randv, M) then
7                  bestv = randv;
8          remove bestv from M;
9          add bestv to M′;
10     add M′ to P′;
```

E. FUSION

For each membrane M in the population, a membrane M' in P' will be selected according to $fit(M', M)$. The two membranes will be fused according to $fit(M', M)$ and $T(M)$.

If $fit(M', M) > T(M)$, membrane M will fuse membrane M'. After fusing, $M = M \cup M'$, and membrane M' will

not change and can be fused again by other membranes. If $fit(M', M) \leq T(M)$, membrane M will not fuse M', and no other membranes will be selected for M in this iteration.

Algorithm 7 Fusion

Input: graph $G = (V, E)$, the thresholds of each membrane
Output: the updated population P

```
for ∀M ∈ P do
    Select a membrane M′ ∈ P′:
        if fit(M′, M) > T(M) then
            M = M ∪ M′;
return P;
```

F. CYTOLYSIS AND SELECTION

As introduced in section III, the selection operator will multiply some best membranes, and the cytolysis operator will dissolve some worst membranes. In MEAMVC, the two operators are implemented in CytolysisAndSelection (Algorithm 8).

In the implementation, we firstly find the worst membrane M, which has the smallest $fit(M)$. After that, the vertices in M will be replaced by $|best| - N$ randomly selected vertices of the *best* membrane, where N is a parameter in our experiments.

Algorithm 8 CytolysisAndSelection

Input: graph $G = (V, E)$, N the size of vertexes that won't be copied to *worst* ($N \ll |best|$)

```
Select the membrane with the smallest fitness value as worst;
Replace all the vertices of worst with |best| − N randomly selected vertices of best ;
```

V. EXPERIMENTAL RESULTS

To demonstrate the effectiveness of MEAF, this section compares the experimental results of MEAMVC with FastVC and FastVC2 (see subsection II-B Algorithm 2). FastVC [24] is one of the methods which can quickly and effectively solve MVC on large instances in recent years. FastVC2 [25] has replaced the construction algorithm in FastVC with three different algorithms.

A. THE BENCHMARKS AND EXPERIMENTAL ENVIRONMENT

In our experiments, we downloaded 102 instances from the Network Data Repository online [35]. We exclude four extremely large ones, therefore, here remains 98 instances, which are used for evaluating the algorithms in our experiments. Many of these 98 giant real-world graphs have millions of vertices and dozens of millions of edges.

In our comparisons, we use the results of FastVC and FastVC2 from [25]. In [25], both FastVC and FastVC2 are implemented in C++, and compiled by g++ (version 4.4.5) with the '−O3' option. And all the experiments of [25] are carried out on a workstation under Ubuntu Linux (version 14.04), using 2 cores of an Intel i7-4800MQ 2.5 GHz CPU and 32 G Byte RAM. MEAMVC is also implemented in C++, and has been compiled by g++ (version 6.3.0) with the "−O3" option for our experiments. All experiments of MEAMVC are carried out on a server under Ubuntu Linux (version 16.04), using 12 cores of an Intel Xeon(Skylake) Platinum 8163 2.5 GHz CPU and 48 G Byte RAM.

B. RESULTS AND ANALYSIS

We run MEAMVC ten times on each instance with a time limitation of 1000 seconds for each run (the same time limitation with FastVC and FastVC2). For the other parameters PS and NS of MEAMVC, we set PS to 5 and NS to 6, which is based on preliminary experiments. For PS, we test $PS \in [5, 20]$ with an increment step of 5. We find MEAMVC with a PS of 5 performs best. The short time limitation and the giant tested graph might be the reason why smaller population has the best performance. For NS, we test $NS \in [2, 12]$ with an increment step of 2. In addition, we find MEAMVC with $NS = 2$ finds fewer best solutions than others. For other NS values, the best solutions they found are pretty close, but $NS = 6$ use less time than others, so we choose $NS = 6$. For other parameters, CSP is set to 10, and N is set to 20 according to preliminary experiments, and k is set to 50 as the same as [24].

As shown in Table. 1, we compared the solution quality of FastVC, FastVC2 and MEAMVC. For each algorithm on each instance, we report the minimum size (min) and the averaged size (avg) of vertex covers found by the algorithm. For two algorithms A and B on one instance, we say A is better than B if A found a smaller minimum size, or if A and B found the same minimum size but A found a smaller averaged size. To make the comparisons clearer, the best results of each instance are in bold, and the worst results of each instance are in italic.

Table. 2 shows the best, medium, and worst instance numbers of each algorithm. From Table. 2, we can observe that MEAMVC and FastVC2 have the most best instance numbers, but MEAMVC slightly precedes FastVC2 on the medium instance numbers; as for FastVC, it has the most worst instance numbers, and the fewest best instance numbers. Furthermore, for 8 instances, MEAMVC found smaller vertex covers which haven't been found by FastVC and FastVC2.

For further performance evaluation, we divide all the instances into two categories. The first category contains the instances whose minimum vertex cover found by the three algorithms is no more than 500,000 vertexes. The second category contains the instances whose minimum vertex cover of being found is more than 500,000 vertexes.

TABLE 1. Experimental results on large real-world graphs.

No	Graph	Best	MEAMVC min(avg)	FastVC min(avg)	FastVC2 min(avg)
1	soc-karate	14	**14(14)**	**14(14)**	**14(14)**
2	rt-retweet	32	**32(32)**	**32(32)**	**32(32)**
3	soc-dolphins	34	**34(34)**	**34(34)**	**34(34)**
4	ia-reality	81	**81(81)**	**81(81)**	**81(81)**
5	ia-enron-only	86	**86(86)**	**86(86)**	**86(86)**
6	ia-infect-hyper	90	**90(90)**	**90(90)**	**90(90)**
7	ca-netscience	214	**214(214)**	**214(214)**	**214(214)**
8	rt-twitter-copen	237	**237(237)**	**237(237)**	**237(237)**
9	web-polblogs	244	**244(244)**	**244(244)**	**244(244)**
10	bio-celegans	249	**249(249)**	**249(249)**	**249(249)**
11	bio-diseasome	285	**285(285)**	**285(285)**	**285(285)**
12	ia-infect-dublin	293	**293(293)**	*293(293.2)*	**293(293)**
13	soc-wiki-Vote	406	**406(406)**	**406(406)**	**406(406)**
14	bio-yeast	456	**456(456)**	**456(456)**	**456(456)**
15	ca-Erdos992	461	**461(461)**	**461(461)**	**461(461)**
16	web-google	498	**498(498)**	**498(498)**	**498(498)**
17	ca-CSphd	550	**550(550)**	**550(550)**	**550(550)**
18	ia-fb-messages	578	**578(578)**	**578(578)**	**578(578)**
19	ia-email-univ	594	**594(594)**	**594(594)**	**594(594)**
20	tech-routers-rf	795	**795(795)**	**795(795)**	**795(795)**
21	ia-email-EU	820	**820(820)**	**820(820)**	**820(820)**
22	web-edu	1451	**1451(1451)**	**1451(1451)**	**1451(1451)**
23	inf-power	2203	**2203(2203)**	**2203(2203)**	**2203(2203)**
24	ca-GrQc	2208	**2208(2208)**	**2208(2208)**	**2208(2208)**
25	tech-WHOIS	2284	**2284(2284)**	**2284(2284)**	**2284(2284)**
26	web-spam	2297	**2297(2297)**	*2298(2298)*	**2297(2297)**
27	soc-twitter-follows	2323	**2323(2323)**	**2323(2323)**	**2323(2323)**
28	bio-dmela	2630	**2630(2630)**	**2630(2630)**	*2632(2632)*
29	web-webbase-2001	2652	**2652(2652)**	**2652(2652)**	**2652(2652)**
30	tech-as-caida2007	3683	**3683(3683)**	**3683(3683)**	**3683(3683)**
31	socfb-MIT	4657	**4657(4657)**	**4657(4657)**	**4657(4657)**
32	socfb-CMU	4986	**4986(4986.3)**	4986(4986.5)	*4987(4987)*
33	web-BerkStan	5384	**5384(5384)**	**5384(5384)**	**5384(5384)**
34	tech-internet-as	5700	**5700(5700)**	**5700(5700)**	**5700(5700)**
35	ca-HepPh	6555	**6555(6555)**	**6555(6555)**	**6555(6555)**
36	web-indochina-2004	7300	**7300(7300)**	**7300(7300)**	**7300(7300)**
37	socfb-Duke14	7683	*7683(7683.5)*	**7683(7683)**	**7683(7683)**
38	socfb-Stanford3	8517	**8517(8517.9)**	**8517(8517.9)**	*8518(8518)*
39	soc-douban	8685	**8685(8685)**	**8685(8685)**	**8685(8685)**
40	soc-epinions	9757	**9757(9757)**	**9757(9757)**	**9757(9757)**
41	socfb-UCSB37	11261	11261(11262.8)	*11261(11263)*	**11261(11262.5)**
42	ca-AstroPh	11483	**11483(11483)**	**11483(11483)**	**11483(11483)**
43	ca-CondMat	12480	**12480(12480)**	**12480(12480)**	**12480(12480)**
44	ia-enron-large	12781	**12781(12781)**	**12781(12781)**	**12781(12781)**
45	socfb-UConn	13230	*13230(13231.8)*	13230(13231.5)	**13230(13230.9)**
46	socfb-UCLA	15222	**15222(15225)**	15223(15224.3)	*15224(15225.3)*
47	tech-p2p-gnutella	15682	**15682(15682)**	**15682(15682)**	**15682(15682)**
48	socfb-Berkeley13	17210	*17211(17213.2)*	17210(17212.7)	**17210(17211.2)**
49	ia-wiki-Talk	17288	**17288(17288)**	**17288(17288)**	**17288(17288)**
50	socfb-Wisconsin87	18383	*18383(18385.6)*	18383(18385.1)	**18383(18385)**
51	soc-BlogCatalog	20752	**20752(20752)**	**20752(20752)**	**20752(20752)**
52	soc-brightkite	21190	**21190(21190)**	**21190(21190)**	*21190(21190.1)*
53	soc-slashdot	22373	**22373(22373)**	**22373(22373)**	**22373(22373)**
54	socfb-Indiana	23315	*23315(23319)*	**23315(23317.1)**	**23315(23317.1)**
55	socfb-UIllinois	24091	*24094(24096.4)*	**24091(24092.2)**	24093(24095.2)
56	socfb-UF	27306	*27308(27310.8)*	27306(27309)	**27306(27308.6)**
57	socfb-Texas84	28165	*28168(28172.8)*	28167(28171.1)	**28165(28168.7)**
58	soc-buzznet	30624	**30624(30624)**	*30625(30625)*	**30624(30624)**
59	socfb-Penn94	31162	*31164(31166.9)*	**31162(31164.8)**	31163(31165.2)
60	socfb-OR	36548	**36548(36548.7)**	*36548(36549.2)*	**36548(36548.7)**
61	soc-LiveMocha	43427	**43427(43427)**	**43427(43427)**	**43427(43427)**
62	rec-amazon	47605	**47605(47605.2)**	47606(47606)	47606(47606)
63	sc-nasasrb	51240	51242(51243)	*51244(51247.3)*	**51240(51243.2)**
64	web-sk-2005	58173	**58173(58173)**	**58173(58173)**	**58173(58173)**
65	tech-RL-caida	74710	74739(74741.1)	*74930(74938.9)*	**74710(74713.1)**
66	soc-lastfm	78688	**78688(78688)**	**78688(78688)**	**78688(78688)**
67	rt-retweet-crawl	81040	**81040(81040)**	*81048(81048)*	**81040(81040)**
68	sc-pkustk11	83911	**83911(83912.5)**	**83911(83912.5)**	*83912(83913.2)*
69	soc-gowalla	84222	84222(84222.5)	**84222(84222.3)**	*84223(84223)*
70	soc-delicious	85532	85583(85601.4)	*85686(85696.4)*	**85532(85536.1)**
71	sc-pkustk13	89217	*89219(89221.6)*	**89217(89220.6)**	89218(89220.3)
72	soc-FourSquare	90108	**90108(90108.6)**	90108(90109.2)	*90109(90109.7)*
73	soc-flixster	96317	**96317(96317)**	**96317(96317)**	**96317(96317)**
74	soc-digg	103239	**103239(103239.8)**	*103244(103245.3)*	103242(103242)
75	web-arabic-2005	114426	**114426(114427.2)**	**114426(114427.2)**	*114426(114427.3)*
76	sc-shipsec1	117263	**117263(117296.2)**	*117318(117337.5)*	117281(117297.5)
77	ca-dblp-2010	121969	**121969(121969)**	**121969(121969)**	**121969(121969)**
78	web-uk-2005	127774	**127774(127774)**	**127774(127774)**	**127774(127774)**
79	ca-citeseer	129193	**129193(129193)**	**129193(129193)**	**129193(129193)**
80	ca-MathSciNet	139951	**139951(139951)**	**139951(139951)**	**139951(139951)**
81	soc-youtube	146376	**146376(146376)**	**146376(146376)**	*146377(146377)*
82	sc-shipsec5	147000	**147000(147009.3)**	*147137(147173.8)*	147067(147077.4)
83	soc-flickr	153271	153271(153271.2)	*153272(153272)*	**153271(153271.1)**
84	ca-dblp-2012	164949	**164949(164949)**	**164949(164949)**	**164949(164949)**
85	sc-pwtk	207687	**207687(207693.4)**	207716(207719.9)	207716(207719.9)
86	soc-youtube-snap	276945	276945(276945.2)	**276945(276945)**	*276947(276947)*
87	socfb-B-anon	303048	**303048(303048)**	*303048(303048.9)*	**303048(303048)**
88	socfb-A-anon	375230	375231(375231)	*375231(375232.8)*	**375230(375230.9)**
89	sc-msdoor	381558	*381559(381559.3)*	**381558(381558.9)**	**381558(381558.9)**
90	web-it-2004	414671	414676(414676)	**414671(414676.3)**	*414688(414689.9)*
91	ca-coauthors-dblp	472179	**472179(472179)**	**472179(472179)**	**472179(472179)**
92	tech-as-skitter	525538	525630(525634)	*527185(527195.9)*	**525538(525542.7)**
93	inf-roadNet-PA	555220	*555341(555368.8)*	**555220(555243)**	555269(555324.6)
94	web-wikipedia2009	648313	**648313(648322.2)**	648317(648321.8)	*648318(648325.1)*
95	soc-pokec	843387	843398(843401.9)	*843422(843434.9)*	**843387(843390.4)**
96	sc-ldoor	856754	**856754(856754.4)**	856755(856757.4)	856755(856757.4)
97	ca-hollywood-2009	864052	**864052(864052)**	**864052(864052)**	**864052(864052)**
98	socfb-uci-uni	866766	**866766(866766)**	*866768(866768)*	**866766(866766)**

TABLE 2. The best, medium, and worst instance numbers for each algorithm.

Rank	MEAMVC	FastVC	FastVC2
# best	75	68	75
# medium	11	12	10
# worst	12	18	13

TABLE 3. The self-deviation of each algorithm.

Vertex Cover Size	(m, n)	MEAMVC	FastVC	FastVC2
≤500,000	(1, 91)	1.2	1.389011	0.78022
>500,000	(92, 98)	6.471429	7.714286	10.45714

Table. 3 shows the stability of MEAMVC, FastVC and FastVC2. The stability evaluates the ability that if one algorithm can obtain stable results in several independent runs. For example, the stability of MEAMVC of the first category is 1.2, which means the average deviation between $avg(MEAMVC, i)$ and $min(MEAMVC, i)$ is no more than 1.2 vertexes. The stability of algorithm A is measured by *self-deviation(A)* in (5).

$$self\text{-}deviation(A) = \sum_{i=m}^{n} \frac{avg(A, i) - min(A, i)}{n - m} \tag{5}$$

where $avg(A, i)$ denotes the averaged size of the covers of A on instance i, and $min(A, i)$ denotes the minimum size of the covers of A on instance i, and (m, n) is range of instances.

Obviously, the algorithm with smaller *self-deviation* is more stable than other algorithms. Therefore, as shown in Table. 3, MEAMVC is more stable than FastVC in both categories. When compared to FastVC2, FastVC2 is more stable in the first category, but MEAMVC is more stable in the second category. For FastVC and FastVC2, they are local search methods that means they can search deeper on a specific tentative solution. So, FastVC2 is more stable in small instances, but in large instances, it's harder for local search methods to escape from the local optimums in different independent runs. Therefore, the *self-deviation* of FastVC2 is smaller in small instances but larger in large instances.

Table. 4 shows the comparisons of the ability of MEAMVC, FastVC and FastVC2 to find the best solution. For example, the *global-deviation* of MEAMVC of the first category is 1.120879, which means the average deviation between $min(MEAMVC, i)$ and $best(i)$ is no more than 1.120879 vertexes. For algorithm A, the ability of A to find the best solution is measured by *global-deviation(A)* in (6), where $best(i)$ denotes the size of the smallest cover of all the compared algorithms on instance i.

$$global\text{-}deviation(A) = \sum_{i=m}^{n} \frac{min(A, i) - best(i)}{n - m} \tag{6}$$

The algorithm with a smaller *global-deviation* have more ability to found a smaller vertex cover. From Table. 4,

TABLE 4. The global-deviation of each algorithm.

Vertex Cover Size	(m, n)	MEAMVC	FastVC	FastVC2
≤500,000	(1, 91)	1.120879	6.813187	1.659341
>500,000	(92, 98)	32	241.2857	7.857143

MEAMVC and FastVC2 is more capable than FastVC in both category. For MEAMVC and FastVC2 in the 91 instances of the first category, the difference of *global-deviation*(MEAMVC) and *global-deviation*(FastVC2) is at least 0.5. That means MEAMVC precedes FastVC2 at least 0.5 vertexes in average for the 91 instances of the first category. But, for the 7 instances in second category, FastVC2 precedes MEAMVC at least 24 vertexes in average. In fact, for small instances, 1000s is enough for MEAMVC to find a good enough solution (a solution better than the solutions found by FastVC and FastVC2), so the *global-deviation* of MEAMVC is smaller than FastVC2. But for large instances, 1000s is not enough for MEAMVC to find a good enough solution, because MEAMVC is a width-based search method. Consequently, the *global-deviation* of MEAMVC in large instances is bigger than FastVC2. As for FastVC, the *global-deviation* of FastVC is always larger than FastVC2 and MEAMVC in both categories, so the intrinsic flaws of FastVC algorithm might be the reason.

To sum up, from the perspective of best and worst instance numbers, MEAMVC outperforms FastVC and slightly outperforms FastVC2. Moreover, from the perspective of stability and ability, MEAMVC outperforms FastVC, and MEAMVC is comparable to FastVC2.

Although, in the later researches, FastVC have been improved a little bit by using random walk [36], k-swap [37] on some instances (some other instances got a worse result), we here still use FastVC and its successor FastVC2 to represent the state of art. In our comparisons, all the algorithms didn't use the data preprocessing technology, although better results can be achieved by preprocessing the data [25].

VI. CONCLUSIONS AND FUTURE WORK

In this work, we introduce a new Membrane Evolutionary Algorithm Framework (MEAF) which is inspired from the structure and functioning of living cells. MEAF is totally different from membrane algorithms in many aspects. The membrane algorithms focus mostly on how to combine different independent heuristic algorithms together. But, MEAF is a heuristic algorithm. It has its own evolution operators. And better results can be obtained by using these operators to evolve membranes and the objects inside membranes.

By implementing MEAF, we devolop a solver named MEAMVC to solve the MVC problem. According to the experimental results, MEAMVC has a largest best instance number and a smallest worst instance number. In more detail, for over 3/4 of the benchmarks, MEAMVC found the best sized vertex cover. That shows the effectiveness of MEAF and MEAMVC. Besides, MEAMVC has many other characteristics, such as conceptual simplicity and ease of variable adjustment.

This work takes a first step towards membrane evolutionary algorithm in solving optimization problems, and provides key insights about how to use membrane evolutionary algorithm to solve problems. In the future, we would like to improve membrane evolutionary algorithm framework and apply membrane evolutionary algorithm framework to other problems.

REFERENCES

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

[2] C. Martín-Vide, G. Păun, J. Pazos, and A. Rodríguez-Patón, "Tissue P systems," *Theor. Comput. Sci.*, vol. 296, no. 2, pp. 295–326, 2003.

[3] M. Ionescu, G. Păun, and T. Yokomori, "Spiking neural P systems," *Fundam. Inf.*, vol. 71, no. 2, pp. 279–308, 2006.

[4] L. Pan, T. Wu, Y. Su, and A. V. Vasilakos, "Cell-like spiking neural P systems with request rules," *IEEE Trans. Nanobiosci.*, vol. 16, no. 6, pp. 513–522, Sep. 2017.

[5] B. Song, C. Zhang, and L. Pan, "Tissue-like P systems with evolutional symport/antiport rules," *Inf. Sci.*, vol. 378, pp. 177–193, Feb. 2017.

[6] H. Peng *et al.*, "Spiking neural P systems with multiple channels," *Neural Netw.*, vol. 95, pp. 66–71, Nov. 2017.

[7] L. Pan and C. Martín-Vide, "Solving multidimensional 0–1 knapsack problem by P systems with input and active membranes," *J. Parallel Distrib. Comput.*, vol. 65, no. 12, pp. 1578–1584, 2005.

[8] M. J. Pérez-Jiménez and A. Riscos-Núñez, "A linear-time solution to the knapsack problem using P systems with active membranes," in *Proc. Int. Workshop Membrane Comput.*, 2003, pp. 250–268.

[9] P. Guo, Y. Dai, and H. Chen, "A p system for Hamiltonian cycle problem," *Optik*, vol. 127, no. 20, pp. 8461–8468, 2016.

[10] X. Liu, J. Suo, S. C. H. Leung, J. Liu, and X. Zeng, "The power of time-free tissue P systems: Attacking NP-complete problems," *Neurocomputing*, vol. 159, pp. 151–156, Jul. 2015.

[11] G. Ping, Z. Jian, C. Haizhu, and Y. Ruilong, "A linear-time solution for All-SAT problem based on P system," *Chin. J. Electron.*, vol. 27, no. 2, pp. 367–373, Mar. 2018.

[12] D. Díaz-Pernil, M. A. Gutiérrez-Naranjo, and H. Peng, "Membrane computing and image processing: A short survey," *J. Membrane Comput.*, vol. 1, no. 1, pp. 58–73, Mar. 2019.

[13] T. Wang *et al.*, "Modeling fault propagation paths in power systems: A new framework based on event SNP systems with neurotransmitter concentration," *IEEE Access*, vol. 7, pp. 12798–12808, 2019.

[14] T. Y. Nishida, "Membrane algorithms," in *Proc. Int. Workshop Membrane Comput.*, 2005, pp. 55–66.

[15] 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.

[16] G. Zhang, M. Gheorghe, and C. Wu, "A quantum-inspired evolutionary algorithm based on P systems for knapsack problem," *Fundam. Informat.*, vol. 87, no. 1, pp. 93–116, 2008.

[17] 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, 2010.

[18] 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.

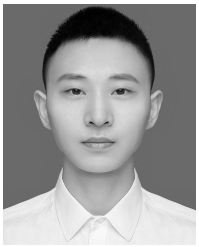
[19] G. Zhang *et al.*, "A novel membrane algorithm based on particle swarm optimization for solving broadcasting problems," *J. Universal Comput. Sci.*, vol. 18, no. 13, pp. 1821–1841, 2012.

[20] 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.

[21] G. Zhang, J. Cheng, M. Gheorghe, and Q. Meng, "A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems," *Appl. Soft Comput.*, vol. 13, no. 3, pp. 1528–1542, Mar. 2013.

[22] Z. Gexiang, R. Haina, C. Jixiang, and Q. Yanhui, "A population-membrane-system-inspired evolutionary algorithm for distribution network reconfiguration," *Chin. J. Electron.*, vol. 23, no. 3, pp. 437–441, Jul. 2014.
[23] X. Jianhua, H. Yufang, C. Zhen, H. Juanjuan, and N. Yunyun, "A hybrid membrane evolutionary algorithm for solving constrained optimization problems," *Optik*, vol. 125, no. 2, pp. 897–902, 2014.
[24] S. Cai, "Balance between complexity and quality: Local search for minimum vertex cover in massive graphs," in *Proc. Int. Conf. Artif. Intell.*, Jun. 2015, pp. 747–753.
[25] S. Cai, J. Lin, and C. Luo, "Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess," *J. Artif. Intell. Res.*, vol. 59, pp. 463–494, Jul. 2017.
[26] I. Dinur and S. Safra, "On the hardness of approximating minimum vertex cover," *Ann. Math.*, vol. 162, no. 1, pp. 439–485, 2005.
[27] J. Chen, I. A. Kanj, and G. Xia, "Improved upper bounds for vertex cover," *Theor. Comput. Sci.*, vol. 411, no. 40, pp. 3736–3756, 2010.
[28] T. Akiba and Y. Iwata, "Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover," *Theor. Comput. Sci.*, vol. 609, pp. 211–225, Jan. 2016.
[29] M. Xiao and H. Nagamochi, "Exact algorithms for maximum independent set," *Inf. Comput.*, vol. 255, pp. 126–146, Aug. 2017.
[30] S. J. Shyu, P.-Y. Yin, and B. M. T. Lin, "An ant colony optimization algorithm for the minimum weight vertex cover problem," *Ann. Oper. Res.*, vol. 131, nos. 1–4, pp. 283–304, 2004.
[31] R. Jovanovic and M. Tuba, "Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem," *Comput. Sci. Inf. Syst.*, vol. 10, no. 1, pp. 133–149, 2013.
[32] K. Kotecha and N. Gambhava, "A hybrid genetic algorithm for minimum vertex cover problem," in *Proc. IICAI*, 2003, pp. 904–913.
[33] H. Bhasin and G. Ahuja, "Harnessing genetic algorithm for vertex cover problem," *Int. J. Comput. Sci. Eng.*, vol. 4, no. 2, p. 6, 2012.
[34] S. Cai, K. Su, C. Luo, and A. Sattar, "NuMVC: An efficient local search algorithm for minimum vertex cover," *J. Artif. Intell. Res.*, vol. 46, no. 1, pp. 687–716, 2013.
[35] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 4292–4293.
[36] Z. Ma, Y. Fan, K. Su, C. Li, and A. Sattar, "Random walk in large real-world graphs for finding smaller vertex cover," in *Proc. IEEE Int. Conf. Tools Artif. Intell.*, Nov. 2017, pp. 686–690.
[37] C. Komusiewicz and M. Katzmann, "Systematic exploration of larger local search neighborhoods for the minimum vertex cover problem," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 846–852.
[38] Q. Zhou, X. Qin, and X. Xie, "Dimension incremental feature selection approach for vertex cover of hypergraph using rough sets," *IEEE Access*, vol. 6, pp. 50142–50153, 2018.
[39] Y. Zhou, C. Qiu, Y. Wang, M. Fan, and M. Yin, "An improved memetic algorithm for the partial vertex cover problem," *IEEE Access*, vol. 7, pp. 17389–17402, 2019.

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



CHANGSHENG QUAN was born in Dianjiang, Chongqing, China, in 1994. He received the bachelor's degree in computer science and technology from Chongqing University, China, in 2017, where he is currently pursuing the master's degree in computer science. His research interests include biological computing and heuristic algorithm.

HAIZHU CHEN was born in Liuzhou, Guangxi, China, in 1980. She received the Ph.D. degree in computer science from Chongqing University, China, in 2011. She is currently an Associate Professor with the Chongqing College of Electronic Engineering, China. Her current research interests include membrane computing and optimization algorithm.