Applying Genetic Algorithm to the Minimum Vertex Cover Problem

Keunhee Han † · Chansoo Kim ‡

ABSTRACT

Let \( G = (V, E) \) be a simple undirected graph. The Minimum Vertex Cover (MVC) problem is to find a minimum subset \( C \) of \( V \) such that for every edge, at least one of its endpoints should be included in \( C \). Like many other graph theoretic problems this problem is also known to be NP-hard. In this paper, we propose a genetic algorithm called LeafGA for MVC problem and show the performance of the proposed algorithm by applying it to several published benchmark graphs.

Keywords : Genetic Algorithm, Graph, Minimum Vertex Cover Problem

1. Introduction

Let \( G = (V, E) \) be a simple undirected graph, where \( V \) (or \( V(G) \)) is the set of vertices and \( E \) (or \( E(G) \)) is the set of edges. The Minimum Vertex Cover (MVC) problem is to find a minimum subset \( C \) of \( V \) such that for every edge, at least one of its endpoints should be included in \( C \). If we let \( n = |V| \) and \( C \) be a subset of \( V \), then MVC can be mathematically defined as follows:

\[
\text{Minimize: } \sum_{i=1}^{n} X_i
\]

Subject to \( \sum_{j=1}^{n} A_{ij} X_i \geq 1 \) for \( 1 \leq j \leq n \), and

\[
X_i \in \{0, 1\} \quad \text{if } v_i \in C
\]

\[
X_i \in \{0, otherwise\}
\]

Vertex cover problem has its applications in the area of VLSI design, scheduling, computer networking and bioinformatics. This problem is also closely related to the problems of maximum clique and independent set problems. However, since MVC problem is known to be NP-hard, researches have focused on developing approximation and heuristic algorithms.

Genetic algorithms (GAs), a part of evolutionary computing, were introduced by Holland in 1975 [1]. Since then GAs have been applied to a large variety of combinatorial optimization problems. GAs maintain a population, which consists of a large number of possible solutions (chromosomes) and search for good solutions among them. After generating the initial population selection operation selects solutions in order to reproduce the more promising solutions. These selection operations are based on the fitness values of the solutions. Chromosomes with higher fitness values will most likely be selected to reproduce, whereas, those with lower fitness values will be discarded. After the selection operation the chromosomes are subject to the operations of crossover.
and mutation. Crossover is the operation of combining the information of two chromosomes (called parents) so that the new chromosomes (called offspring) inherit the properties of both parents. Mutation is the process of changing the information of chromosome randomly in order to prevent converging the solutions into a local optimum.

GAs have been applied to MVC problem and can be found in [2, 3, 4]. In [2], the authors applied GA to MVC problem with infeasible chromosomes and applied their algorithms to random graphs. The most obvious encoding scheme for MVC is to use binary encoding scheme. However, in [3], the authors suggested a different encoding scheme called embedded binary decision diagram. In [4], the authors investigated the roles of the amounts of domain knowledge for solving MVC in GAs. They suggested three GAs with different amounts of embedded domain knowledge and compared their performance using BIL, R, E graphs (defined later).

In this paper, we develop a new genetic algorithm called LeafGA for MVC problem and analyze the performance of the proposed algorithm based on simulations. The basic idea of LeafGA is to find a vertex v adjacent to a leaf vertex. This vertex v is included in the partially built minimum vertex cover and removed from the current graph and continue these process in recursive manner until the edge set becomes empty. Note that LeafGA always maintains only feasible chromosomes.

The rest of this paper is organized as follows: In section 2, we prove some properties of MVC and show the strategies of the proposed genetic algorithm. In section 3, we apply LeafGA to the benchmark graphs developed by BHOSLIB [6] and two special classes of graphs. Finally, in section 4, we conclude our discussion.

2. Genetic Algorithm for MVC

Let VC and MVC denote a vertex cover and minimum vertex cover of a graph, respectively. The degree of a vertex v is denoted deg(v), and also as deg_G(v) whenever G needs to be distinguished from some other graph also under consideration. If deg(v) = 1, then we call v a leaf vertex. Let N(v) be the vertices adjacent to v in G.

2.1 Representation and initialization of chromosomes

In our GA, each chromosome, at all generations, represents a feasible vertex cover of a given input graph. For this purpose each chromosome is represented by a (0,1)-vector c of length n, where n = |V|. Therefore, for a chromosome c, if c[i] = 1 then it means vertex i is in VC; otherwise it is not in VC.

Finding a vertex cover of a graph G can be done by performing the following two simple steps repeatedly.

(step 1) Choose a vertex v with deg(v) ≠ 0 and add v to S.

(step 2) Remove v from G. Let G' be the resultant graph. If E(G') ≠ ∅ then go to step 1; otherwise stop.

Upon termination of the above procedure it is clear that the set S contains a VC of a given graph G. Therefore, in order to find a MVC, special attention must be paid to the selection of a vertex in step 1. Let v be a leaf vertex of G and w be the unique vertex adjacent to v. The edge (v, w) also must be covered by v or w or by both. However, the following theorem shows that covering the edge (v, w) by w alone is sufficient for any MVC.

Theorem 2.1. Let M be a MVC of a graph G = (V, E) such that M contains at least one leaf vertex v. Let w be the unique vertex adjacent to v in G, then M − \{v\} ∪ \{w\} is a MVC of G.

Proof: If w ∈ M, since M' = M − \{v\} is also a vertex cover of G, it leads to a contradiction that |M'| < |M|. Hence, w ∉ M. Since w ∉ M and w is the unique vertex adjacent to v in G, M − \{v\} ∪ \{w\} is also a MVC of G.

Theorem 2.1 implies that, in step 1 of the above procedure, we should not select a leaf vertex. Instead, the unique vertex adjacent to leaf vertex must be selected to form a better VC. However, if there exists no leaf vertices, we randomly choose a vertex v with deg(v) ≠ 0. Algorithm 2.1 shows the details of computing a VC of a graph.

Algorithm 2.1: Generation of a vertex cover

Procedure generate_VC(G, c)

remove_degree_one(G, c);

while E(G) ≠ ∅ do

randomly choose a vertex v such that deg(v) ≠ 0;
\text{c(v)} = 1;

if there exists a vertex v such that deg(v) = 1

remove_degree_one(G, c);

return c;

end-procedure

Procedure remove_degree_one(G, c)

let D = {v ∈ V | deg(v) = 1};

while D ≠ ∅ do

randomly choose a vertex v ∈ D and let w be the vertex adjacent to v;
\text{c(w)} = 1;

delete v and w from G;

update D, i.e., if deletion of v and w generates other leaf vertices, then add them to D;

end-procedure

In our GA, since we always maintain feasible chromosomes only, we are able to use simple objective function.
Let $VC(c)$ be the set of vertices represented in the chromosome $c$, i.e., $VC(c) = \{ i | c[i] = 1 \}$. Then the objective function is defined as follows:

$$f(c) = |VC(c)|.$$  

Therefore, our main objective is to minimize $f(c)$ over all possible chromosomes.

### 2.2 Genetic Operators

Let $p_1$ and $p_2$ be the two parent chromosomes chosen for the crossover operation. Since $p_1$ and $p_2$ are feasible vertex covers, the vertices contained in both parents should be inherited to the offspring. We remove these common vertices and all the edges adjacent to these vertices from $G$ and finally use the procedure $generate_{VC}$ in order to ensure the feasibility of the offspring. Algorithm 2.2 shows the details of the crossover operation.

**Algorithm 2.2.** Crossover Operation

```pseudo
procedure crossover(G, p1, p2, o)
    for $i = 0$ to $n - 1$ do
        if $p1[i] = p2[i] = 1$ then
            $o[i] = 1$
        else
            remove vertex $i$ from $G$;
        end-
    generate_{VC}(G, o);
end-
```

Let $c$ be a chromosome and $v \in VC(c)$ of an input graph $G = (V, E)$. Since $VC(c)$ is a vertex cover of $G$, clearly, $N(v) \subseteq VC(c)$. Therefore, if we add $v$ to $VC(c)$, then some vertices of $N(v)$ can be excluded from $VC(c)$ while maintaining the feasibility of $VC(c)$. Let $w \in N(v)$ and $W = (z \in N(w) | z \notin VC(c))$. Then, after we add $v$ to $VC(c)$, we can remove the vertex $w$ from $VC(c)$ if $W = \emptyset$. We use this idea as mutation operator. Algorithm 2.3 shows the details of the mutation operation.

**Algorithm 2.3.** Mutation Operation

```pseudo
procedure mutationOne(G, c, v) // $v \notin VC(c)$
    $c'[v] = 1$; $cnt = 0$; $c' = c$
    for each vertex $w$ adjacent to $v$ do
        let $W = (z \in N(w) | z \notin VC(c))$
        if $W = \emptyset$
            $cnt = cnt + 1$
            $c[w] = 0$
        if $cnt < 1$
            return $c'$;
        else
            return $c'$;
end-
```

In Algorithm 2.3, the variable $cnt$ counts the number of vertices that removed from $VC(c)$ after adding $v$ to $VC(c)$. At the end of this procedure if the value of $cnt$ is less than one then it means the mutation operation only degenerates the input chromosome. Therefore, in this case, we return the original chromosome, i.e., the mutation operation does not change the input chromosome $c$ in any way.

Since the operator $mutationOne$ prevents degenerating chromosomes, its capabilities as a mutation operator is somewhat limited. Therefore, in order to introduce new chromosomes into population, we use another mutation operator called $mutationTwo$ as shown in Algorithm 2.4.

**Algorithm 2.4.** Mutation Operation

```pseudo
procedure mutationTwo(G, c)
    for $i = 0$ to $n - 1$ do
        if $c[i] = 1$
            $c[i] = 0$
        else
            $c[i] = 1$;
            remove vertex $i$ from $G$;
        generate_{VC}(G, c);
end-
```

Note that $mutationTwo$ also only generates feasible vertex covers.

For the selection we use roulette wheel selection mechanism with slots sized proportionally to the fitness of the chromosomes. LeafGA also enforces eliticism which ensures the most highly fit chromosome of the current population is passed on to the next generation.

### 3. Experiments

In [4], Jun He et. al. adapted two classes of test graphs in order to measure the performances of their proposed genetic algorithms for MVC. The first class of test graphs contains an odd number of vertices where

$$V = \{ v_1, \ldots, v_n \}$$

$$E = \{ (v_i, v_{i+1}), (v_2, v_3), \ldots, (v_{n-1}, v_n), (v_n, v_1) \}.$$  

Since this graph is very similar to the cycle, it is not hard to see that MVC should contain the vertices $\{ v_2, v_4, \ldots, v_{n-2} \}$. Therefore, the value of optimum MVC is $(n - 1)/2$. However, as noted in [4], for any approximation algorithm, it is easy to fall into local optimum of the value $(n + 1)/2$. The definition of the second class of test graphs called $B(L, R, E)$ is as follows: The vertex set $L$ consists of $r$ vertices. The vertex $R$ is further subdivided into $r$ sets called $R_1, \ldots, R_r$. Each vertex in $R_i$ has an edge to $i$ vertices in $L$ and no two vertices in $R_i$ have a common neighbor in $L$. Note that $B(L, R, E)$ is a bipartite graph and it is easy to verify that the partite set $L$ is a MVC of $B(L, R, E)$. We also note that the graph $B(L, R, E)$, which is hard for deterministic greedy search algorithm, was originally defined in [5].

Even though these two classes of test graphs are expected to be hard instances for any heuristic algorithm for MVC problem, it is evident that the procedure
generate_VC can always find optimum solutions for these classes of graphs. For example, if we apply generate_VC to $B(L, R, E)$, since all the vertices in $R_1$ has degree one and each vertices in $L$ has unique neighbor in $R_1$, all the vertices in $L$ will be included in the chromosome. After that no vertices in the set $R_2 \sim R_r$ will be added to the chromosome since the edge set of $G$ is empty.

Since the two classes of test graphs used in [3] are trivial instances for LeafGA, in order to measure the performances of LeafGA we choose the benchmark graphs developed by BHOSLIB [6] as the test graphs. These graphs are available on the Internet.

Table 3.1 contains the results of executing LeafGA for each instance of the BHOSLIB test graphs. Even though the full set of BHOSLIB contains 40 graphs we selected only 8 representative graphs for the tests since the results of the other graphs are very similar. For each instance graph we executed LeafGA ten times and included the results of best, worst and average fitness values in Table 3.1. For these tests we used the population size of 40, crossover rate of 0.8. For the two mutation operators $mutationOne$ and $mutationTwo$ we used the mutation rates of 0.07 and 0.02, respectively. For the termination condition we used fixed number of iteration of 2,000.

The last column of Table 3.1 shows the results of CKACS algorithm developed for MVC by Gimour et al. [7]. It shows the average MCS values of ten executions of their algorithm on the same set of instance graphs. Note that CKACS is an Ant Colony Algorithm.

From Table 3.1 we can see that LeafGA can find MVC of each test graphs very close to their optimum values. It also shows that LeafGA is not sensitive to the size of the input graphs. In all instances, based on the average values, LeafGA shows much better performance than CKACS does. Also, the narrow gaps between best and worst fitness values indicate that the performance of LeafGA is very steady.

<table>
<thead>
<tr>
<th>Graphs</th>
<th>LeafGA</th>
<th>CKACS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>V</td>
<td>E</td>
</tr>
<tr>
<td>frb30-15-1</td>
<td>450</td>
<td>17827</td>
</tr>
<tr>
<td>frb35-17-1</td>
<td>595</td>
<td>27856</td>
</tr>
<tr>
<td>frb40-19-1</td>
<td>760</td>
<td>41314</td>
</tr>
<tr>
<td>frb45-21-1</td>
<td>945</td>
<td>59186</td>
</tr>
<tr>
<td>frb50-23-1</td>
<td>1150</td>
<td>80072</td>
</tr>
<tr>
<td>frb53-24-1</td>
<td>1272</td>
<td>94227</td>
</tr>
<tr>
<td>frb56-25-1</td>
<td>1400</td>
<td>109676</td>
</tr>
<tr>
<td>frb59-26-1</td>
<td>1534</td>
<td>126575</td>
</tr>
</tbody>
</table>

4. Conclusions

In this paper, we developed a new genetic algorithm called LeafGA for the minimum vertex cover problem. We proved that leaf vertices should not be included in MVC, and based on this property LeafGA selects the vertices adjacent to leaf vertices recursively to form a feasible vertex cover. Finally, we showed the performance of LeafGA by applying it to known benchmark graphs.

Reference