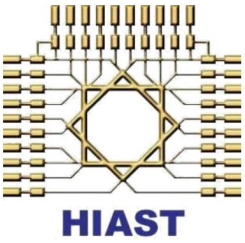
Syrian Arab Republic
Higher Institute for Applied Sciences and Technology
Informatics Department
Fourth Year



Using Genetic Algorithms to Find Approximations for the Minimum Vertex Cover Problem

Keywords: genetic algorithms, NP-complete, combinatorial optimization, non-deterministic algorithms, approximation algorithms, minimum vertex cover.

Author: *Farouk Hjabo*
Academic Supervisor: *Dr. Said Desouki*
General Supervisor: *Dr. Kadan Aljoumaa*
Langauge Supervisor: *Mr. Fahmi Alammareen*

February 25, 2018

Abstract

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

"It is not the strongest of the species that survives, nor the most intelligent, but the one most responsive to change."

— Charles Darwin

Contents

Cover Page **i**

Abstract **ii**

Contents **iii**

List of Figures iv

List of Tables iv

Abbreviations iv

1 Introduction **1**

2 Genetic Algorithms **1**

2.1 The Intuition Behind GAs 1

2.2 Typical GA Schema 2

3 The Minimum Vertex Cover Problem **4**

3.1 Related Algorithms 5

3.2 GA for the mvcp 7

4 Experimental Results **8**

4.1 results comments 11

5 Conclusion **13**

References **14**

List of Figures

1 General schema for genetic algorithms 3

2 The regular graph after Papadimitriou and Steiglitz 6

List of Tables

1 general schema for genetic algorithms 10

2 general schema for genetic algorithms 10

3 general schema for genetic algorithms 11

4 general schema for genetic algorithms 12

Abbreviations

GA	**G**enetic **A**lgorithm
mvc	**M**inimum **V**ertex **C**over
mvcp	**M**inimum **V**ertex **C**over **P**roblem

1 Introduction

Once the NP-hardness of a combinatorial optimization problem is established, the search for an optimal solution is abandoned. The goal then becomes one of finding a good heuristic, i.e. a polynomial running time algorithm that can find solutions close to the optimal. In most cases, traditional heuristics are problem dependent; a heuristic is tailored to the specific problem it is trying to solve.

In this paper, we present an alternative approach that uses genetic algorithms as a generalized heuristic for solving NP-hard combinatorial optimization problems. The application of a genetic algorithm is demonstrated here for the *minimum vertex cover* problem. These algorithms have been successfully applied to a broad range of problems. This wide range can be tackled by genetic algorithms mainly due to the fact that they work with an encoding of the domain rather than with the problem domain itself.

2 Genetic Algorithms

2.1 The Intuition Behind GAs

Genetic Algorithms (GAs) [1, 2] are population based search algorithms, where by repeated use of genetic operations, such as ***mutation***, ***selection***, ***crossover***, etc... Successive new generations of better populations in the direction of search objectives are created. They are inspired by Darwinian principles based on natural evolution. In other words, the main idea behind genetic algorithms is that only the ***fittest*** individuals will survive among generations of evolution.

Main advantage of genetic algorithms is that they ideally do not make any assumption about the underlying problem, hence they are suitable to tackle a wide range of diverse problems in engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, politics, chemistry, etc... [1, ch. 6]

So what is a GA? A typical GA consists of the following [1, ch. 1]:

1. a number, or ***population***, of candidate solutions to the problem.
2. a way of calculating how good or bad the individual solutions within the pop-

ulation are. i.e. how ***fit*** an individual is?

3. a method for mixing fragments of the better solutions to form new, on average even better solutions. Which permits the population to ***evolve*** naturally.
4. a ***mutation*** operator to avoid permanent loss of diversity within the solutions. This allows introducing new information in the population.

2.2 Typical GA Schema

The basic iteration cycle of a genetic algorithm proceeds on a population of individuals, each of which represents a search point in the space of potential solutions of a given optimization problem. In case of a canonical genetic algorithm, each individual is a binary vector $\vec{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ of fixed length n. The fitness function $f : \{0,1\}^n \to \mathbb{R}$ provides a quality measure which is used by the selection procedure to direct the search towards regions of the search space where the average fitness of the population increases. The recombination operator allows for the exchange of information between different individuals, and mutation introduces innovation into the population. This cycle is repeated until a termination criterion is fulfilled. In most cases, the algorithm is terminated after a certain number of iterations of the basic cycle or when a satisfactory fitness level has been reached.
Refer to figure 1 for a visual representation of this process, and refer to algorithm 1 for a more concise description.

Initialization The population size (denoted as u) depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be 'seeded' in areas where optimal solutions are likely to be found.

Selection Normally, selection in genetic algorithms is a probabilistic operator which uses the relative fitness $p_s(\vec{x_i}) = f(\vec{x_i}) / \sum_{j=1}^{u} f(\vec{x_j})$ to serve as selection probabilities (u denotes the population size). This selection operator is called proportional selection. It is commonly used; as intuitively the more fit a solution is the more we want it to affect the next generations.
If the problem under consideration is a minimization one, or if the fitness function can take negative values, then $f(\vec{x_i})$ has to be linearly transformed before calculating

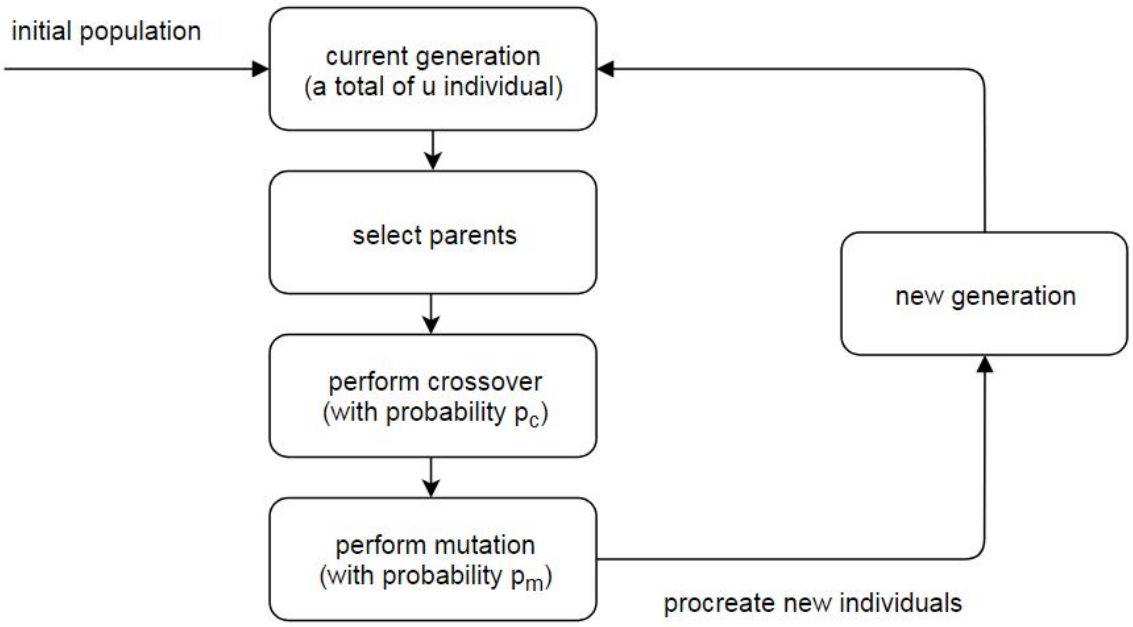


Figure 1: General schema for genetic algorithms

selection probabilities. This technique known as linear dynamic scaling is commonly used in genetic algorithms [2, pp. 123–124].

Crossover The recombination (crossover) operator allows for the exchange of information between different individuals. The original one-point crossover [3] works on two parent individuals by choosing a crossover point $\chi \in \{1, \ldots, n-1\}$ at random and exchanging all bits after the χ^{th} one between both individuals. Therefore, two new off-springs will be formed.
For example, if the two parents were encoded as $(111, 000)$ and we chose $\chi = 2$ then the new two off-springs will be $(110, 001)$. The crossover rate p_c (e.g. $p_c \approx 0.6$) determines the probability to undergo crossover (see algorithm 1 line 9), this allows the two parents to be present in the next generation; as crossover is not mandatory. Moreover, the value of p_c indicates the ratio of new off-springs in the next generation. The crossover operator can be extended to a generalized multi-point crossover or even to uniform crossover, where it is randomly decided for each bit whether to exchange it or not. The strong mixing effect introduced by uniform crossover is sometimes helpful to overcome local optima.

Mutation Innovation (new information) is introduced into the population by means of mutation, which works by inverting bits with a small probability p_m (e.g. $p_m \approx 0.001$) (see algorithm 1 lines 10 11 for clarity). Though mutation is often interpreted as a rather unimportant operator in genetic algorithms [3], recent theoretical work gives strong evidence for an appropriate choice of a mutation rate $p_m = 1/n$ on many problems tocite2, 12 misp.

Algorithm 1 GenticAlgorithm(n, u, p_c, p_m, f)

1: $P \leftarrow \{\}$ ▷ P contains the population
2: **loop** u times ▷ initialization
3: add a randomly generated binary vector of length n to P
4: **end loop**
5: **while** termination criterion is not fulfilled **do**
6: $Q \leftarrow \{\}$ ▷ Q will hold the new generation
7: **loop** $u/2$ times
8: select x, y from P using proportional selection ▷ selection
9: with probability p_c perform crossover between x, y ▷ crossover
10: for each bit in x flip it with probability p_m ▷ mutation
11: for each bit in y flip it with probability p_m ▷ mutation
12: add x, y to Q
13: **end loop**
14: $P \leftarrow Q$
15: **end while**

3 The Minimum Vertex Cover Problem

The problem of finding a minimum vertex cover is a classical optimization problem in computer science and is a typical example of an NP-complete optimization problem tocitewolfram that has approximation algorithms. The mvcp of an undirected graph $G = (V, E)$ where V is the set of vertices and E denotes the set of edges, consists in finding the smallest subset $V' \subseteq V$ such that $\forall \langle i, j \rangle \in E$, we have $i \in V'$ or $j \in V'$ (or both). V' is said to be a vertex cover of G.

The following is a formal definition of the mvcp in which we make use of Stinson's terminology for combinatorial optimization problems tocite14 mvcp:

Problem instance: A graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of vertices and $E \subseteq V \times V$ the set of edges. An edge between vertices i, j is denoted by

the pair $\langle i, j\rangle \in E$. We define the adjacency matrix e_{ij} according to:

$$e_{ij} = \begin{cases} 1 & \text{if } \langle i, j\rangle \in E \\ 0 & \text{otherwise} \end{cases}$$

On particular, we have $e_{ii} = 0$.

Feasible solution: A set $V' \subseteq V$ such that $\forall\langle i, j\rangle \in E, i \in V' \vee j \in V'$

Objective function: The size $|V'|$ of the vertex cover V'.

Optimal solution: a vertex cover V' that minimizes $|V'|$.

An interesting property of the mvcp is that by solving it, we also have a solution for two other graph problems: the *maximum independent set problem* (given $G = (V, E)$, find the largest subset $V' \subseteq V$ such that $\forall i, j \in V', \langle i, j\rangle \notin E$) and the *maximum clique problem* (given $G = (V, E)$, find the largest subset $V' \subseteq V$ such that $\forall i, j \in V', \langle i, j\rangle \in E$) The close relationship between these problems is characterized by the following lemma tocite6 misp.

Lemma 1

For any graph $G = (V, E)$ and $V' \subseteq V$, the following statements are equivalent:

- *V' is the minimum vertex cover of G.*
- *$V - V'$ is the maximum independent set in G.*
- *$V - V'$ is the maximum clique in $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{\langle i, j\rangle \in V \times V : \langle i, j\rangle \notin E\}$*

Consequently, one can obtain a solution of the maximum independent set problem by taking the complement of the solution to the minimum vertex cover problem. A solution to the maximum clique problem is obtained by taking the complement of the minimum vertex cover of $\bar{G}$.

3.1 Related Algorithms

Since we are interested in covering all the edges of the given graph by using as few nodes as possible, one might be tempted to use a greedy-based heuristic to tackle the

mvcp. The algorithm consists in repeatedly selecting a vertex of highest degree (the node that covers as many of the remaining edges as possible), and removing all of its incident edges. This is not a good strategy as was demonstrated by Papadimitriou and Steiglitz tocite10 p. 407 mvcp.

They considered regular graphs*, each of which consists of three levels. The first two levels have the same number of nodes while the third level has two nodes less than the number of nodes found on the previous two levels. More precisely, each graph consists of $n = 3k + 4$ ($k \geq 1$) nodes; $k + 2$ nodes on the first level, labeled $1, 2, \ldots, k + 2$; followed by $k + 2$ nodes on the second level, labeled $k + 3, \ldots, 2k + 4$; while the k nodes of the third level are labeled $2k + 5, \ldots, 3k + 4$. The regular graph for $k = 3$ can be found in figure 2. A minimum vertex cover is obtained by choosing all the nodes of the second level, but the greedy strategy would start with the nodes of the third level, since these have highest degree. Consequently, the greedy strategy finds a solution of size $2k + 2$ (since it has to select $k + 2$ nodes in addition to the nodes of the third level), while the optimal solution has size $k + 2$.

However this algorithm is not totally useless. It may be shown that it always achieves the ratio $\Theta(\lg n)^\dagger$, where n is the number of vertices. That is, if $C^*(n)$ is the size of the optimal solution and $C(n)$ is the size of the solution found by the greedy algorithm. Then we have $\frac{C(n)}{C^*(n)} = \Theta(\lg n)$ tocite9 p. 323 mvc.

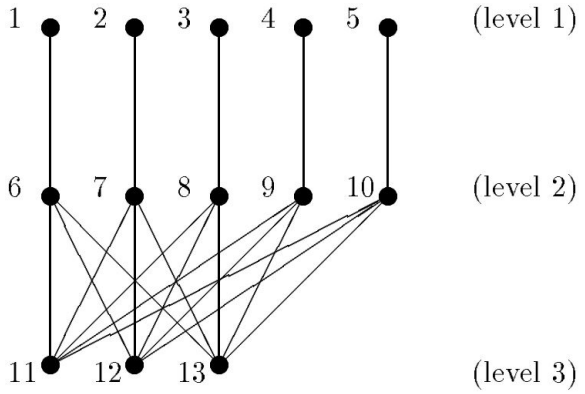


Figure 2: Construction of the regular graph after Papadimitriou and Steiglitz with $k = 3$.

Fortunately, a simpler and better algorithm exists. The following simple algorithm is surprisingly the best approximation algorithm known for the mvcp tocite9 p. 301

*by regular graphs we don't refer to the definition where each vertex has the same degree.

$^\dagger f(n) = \Theta(g(n)) \iff \exists \alpha, \beta, n_0 : \forall n \geq n_0,\ \alpha \cdot g(n) \leq f(n) \leq \beta \cdot g(n)$

mvcp.

Algorithm 2 vercov($G = (V, E)$)

1: $C \leftarrow \{\}$ ▷ C contains the vertex cover being constructed
2: $E' \leftarrow E$
3: **while** $E' \neq \phi$ **do**
4: randomly choose $\langle u, v\rangle \in E'$
5: $C \leftarrow C \cup \{u, v\}$
6: $E' \leftarrow E' - \{\langle x, y\rangle \in E' : x = u \vee y = v \vee x = v \vee y = u\}$
7: ▷ remove from E' every edge incident on either u or v
8: **end while**
9: **return** C

This is a *2-approximation algorithm*. Which means that the size of the vertex cover it returns is guaranteed to be no more than twice the size of an optimal vertex cover tocite[1], p. 968 mvcp. More precisely, if C^* is the size of the optimal solution and C is the size of the solution returned by algorithm 2 then the inequality $C \leq 2C^*$ must hold (notice that here the approximation doesn't depend on the number of vertices).

3.2 GA for the mvcp

In order to apply genetic algorithm to solve the mvcp, a cover V' is represented by a binary vector $\vec{x} = x_1 x_2 \dots x_n$ where $x_i = 1$ if the i^{th} node is in V', and $x_i = 0$ if it is not. Using this representation, genetic algorithm is applied to minimize the following fitness function [4]:

$$\begin{aligned} f(\vec{x}) &= \sum_{i=1}^{n} \left(x_i + n \cdot (1 - x_i) \cdot \sum_{j=i}^{n} (1 - x_j) e_{ij} \right) \\ &= \sum_{i=1}^{n} x_i + n \cdot \sum_{i=1}^{n} \sum_{j=i}^{n} (1 - x_i) \cdot (1 - x_j) \cdot e_{ij} \end{aligned} \tag{1}$$

The term $\sum_{i=1}^{n} x_i$ of $f(\vec{x})$ determines the size of the potential vertex cover represented by $\vec{x}$, while the term $n \cdot \sum_{i=1}^{n} \sum_{j=i}^{n} (1 - x_i) \cdot (1 - x_j) \cdot e_{ij}$ penalizes sets V' that are not covers by adding a penalty of magnitude n for each edge e_{ij} for which $i \notin V' \wedge j \notin V'$ (note that e_{ii} is previously defined to be zero). Consequently, any

infeasible solution will have a fitness greater than or equal to n. And for any feasible solution the second term will drop to zero, making the fitness value of the worst feasible solution at most n.

This fitness function was developed according to the following design principles that are important for a successful penalty function approach tocite7,11 mvcp:

- Infeasible binary vectors are guaranteed to yield fitness values which are inferior to fitness values of the worst feasible solutions.
- The penalty should be graded, i.e., fitness values should improve as solutions approach feasible regions of the search space. In other words, the more edges a solution covers, the fitter it is. notice that this is guaranteed by the second term in equation 1, where for each uncovered edge the fitness is penalized with the value of n.

The experiments reported in section 4 are performed by using a genetic algorithm with a population size of $u = 50$, mutation rate $p_m = 1/n$, crossover rate $p_c = 0.6$, proportional selection, and two-point crossover [4]. As reported in tocite2, 12 mvcp, the latter is expected to perform better than the traditional one-point crossover. A bound for the total number of fitness function evaluations is set as the termination criterion, see section 4 for details.

4 Experimental Results

For the experimental tests, the genetic algorithm software package GENEsYs is used tocite[1] misp. This implementation is based on the widely used GENESIS software by Grefenstette tocite3, pp. 374-377 misp, but allows for more flexibility concerning genetic operators and data monitoring. The parameter settings for the experiments are given in section 3.2.

Test Set 1: Multiple problem instances are used to demonstrate the applicability of GAs to tackle the mvcp. The results in table 1 and table 2 correspond to the first test set, which consists of five random graphs of size $n = 100$ with different edge densities d. Densities $d \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ are used respectively. An edge density of $d = 0.1$ means that an edge is placed between two nodes with a probability of 0.1. Moreover, we construct the graphs in a way that guarantees an optimum of

quality 55 [4]. i.e. we use algorithm 3 with $k = 55$ to generate random graphs. In addition, the regular graphs introduced by Papadimitriou and Steiglitz and described in section 3.1 are used. Recall that these graphs contain $n = 3k + 4$ ($k \geq 1$) nodes distributed on three levels. They can be scaled up by choosing high values for k. A graph instance of the regular graph of size $n = 100$ ($k = 32$) is chosen.

Algorithm 3 GenerateRandomGraph(n, d, k)

```
1: randomly select V' = {i_1, ..., i_k} ⊆ V = {1, ..., n}
2:         ▷ randomly select k different nodes, these nodes will hold a potential mvc
3: for i = 1 to n − 1 do
4:     for j = i + 1 to n do
5:         if (Random(0, 1) ≤ d) ∧ (i ∈ V' ∨ j ∈ V') then
6:             e_ij = 1
7:         else
8:             e_ij = 0
```

For each of these graphs, a total of $N = 100$ independent runs of the genetic algorithm is performed. The results are summarized in table 1 for the best fitness values that were encountered during the 100 runs. For each problem instance, the different fitness values that were obtained and their frequencies are recorded. Furthermore, the average fitness value $\bar{f}$ over all 100 runs is indicated at the bottom of the table. For example, the first column in table 1 indicates that the best obtained fitness value in 1 of the 100 runs was $f(\vec{x}) = 53$, the best obtained fitness value in 1 of the 100 runs was $f(\vec{x}) = 54$, the best obtained fitness value in 3 of the 100 runs was $f(\vec{x}) = 55$, and so on... And the average of these values is indicated by $\bar{f}$, which is the average value of the best obtained fitness values over the 100 runs.

As a termination criterion for the genetic algorithm, a bound for the total number of function evaluations is set. For example The total number of function evaluations per single run is chosen to be $2 \cdot 10^4$ in table 1. That is, the algorithm terminates when $2 \cdot 10^4$ evaluations of the fitness function had occurred. This bound is indicated as an index t in the notation $f_t(\vec{x})$. Consequently, only a small fraction of the search space is explored by the genetic algorithm. This fraction is about $2 \cdot 10^4/2^{100} \approx 1.6 \cdot 10^{-24}\%$ for the first test set. and it is about $4 \cdot 10^4/2^{200} \approx 2.5 \cdot 10^{-54}\%$ for the second test set.

For each of the problems in the first test set, a total of $N = 100$ independent runs of the vercov algorithm is performed. The results are summarized in table 2. Notice that for one problem instance, the results may differ in each run due to the non-determinism of the algorithm (see algorithm 2 line 4).

mvcp100-01		mvcp100-02		mvcp100-03		mvcp100-04		mvcp100-05		PS100	
$f_{2\cdot10^4}(\vec{x})$	N	$f_{2\cdot10^4}(\vec{x})$	N	$f_{2\cdot10^4}(\vec{x})$	N	$f_{2\cdot10^4}(\vec{x})$	N	$f_{2\cdot10^4}(\vec{x})$	N	$f_{2\cdot10^4}(\vec{x})$	N
53	1	55	34	55	77	55	96	55	99	34	65
54	1	57	3	56	1	67	1	83	1	66	35
55	3	59	2	59	6	68	1				
56	6	60	3	63	3	75	1				
57	4	61	10	64	3	90	1				
58	4	62	1	66	1						
59	9	63	8	67	1						
60	4	64	1	68	1						
61	6	65	1	70	1						
62	12	66	6	71	1						
63	5	67	6	74	1						
> 63	45	> 67	25	> 74	4						
$\bar{f} = 62.61$		$\bar{f} = 62.75$		$\bar{f} = 57.62$		$\bar{f} = 55.80$		$\bar{f} = 55.28$		$\bar{f} = 45.20$	

Table 1: general schema for genetic algorithms

mvcp100-01		mvcp100-02		mvcp100-03		mvcp100-04		mvcp100-05		PS100	
$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N
80	6	80		80		80	1	80		66	100
82	17	82	1	82		82		82	1		
84	23	84	2	84	1	84	1	84			
86	20	86	4	86	6	86		86	5		
88	17	88	18	88	12	88	5	88	7		
90	14	90	17	90	15	90	13	90	10		
92	1	92	31	92	18	92	24	92	21		
94	2	94	20	94	28	94	34	94	27		
96		96	4	96	16	96	19	96	17		
98		98	1	98	4	98	3	98	9		
100		100		100		100		100	3		
$\bar{f} = 86.46$		$\bar{f} = 89.22$		$\bar{f} = 92.22$		$\bar{f} = 92.96$		$\bar{f} = 93.12$			

Table 2: general schema for genetic algorithms

To test the behaviour of the genetic algorithm as well as the vercov algorithm for an even larger problem size, The same experiments were performed for larger graphs The results in table 3 and table 4 correspond to the second test set, which consists of five random graphs of size $n = 200$ with different edge densities d, $d \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$, are used. Moreover we construct the graphs in a way that guarantees an optimum of quality 110 (see algorithm 3). In addition to a graph instance of the regular graphs introduced by Papadimitriou and Steiglitzof of size $n = 220(k = 66)$. In this case, the genetic algorithm was allowed to run for $4 \cdot 10^4$ function evaluations. The corresponding results obtained by the genetic algorithm are shown in table 3, while the results from the vercov algorithm are presented in table 4. See the previous paragraphs for a better understanding of these results.

mvcp200-01		mvcp200-02		mvcp200-03		mvcp200-04		mvcp200-05		PS202	
$f_{4\cdot10^4}(\vec{x})$	N	$f_{4\cdot10^4}(\vec{x})$	N	$f_{4\cdot10^4}(\vec{x})$	N	$f_{4\cdot10^4}(\vec{x})$	N	$f_{4\cdot10^4}(\vec{x})$	N	$f_{4\cdot10^4}(\vec{x})$	N
110	2	110	55	110	95	110	99	110	100	68	60
113	1	120	10	128	1	140	1			134	40
116	4	121	7	129	6						
117	2	122	2	130	2						
119	3	123	7	134	1						
120	1	125	1								
121	2	126	1								
122	3	127	1								
124	3	129	1								
125	6	132	2								
126	2	134	1								
> 126	71	> 134	12								
$\bar{f} = 132.50$		$\bar{f} = 119.07$		$\bar{f} = 111.01$		$\bar{f} = 110.30$		$\bar{f} = 110.00$		$\bar{f} = 108.00$	

Table 3: general schema for genetic algorithms

4.1 results comments

From these tables, a number of interesting conclusions can be drawn. For the random graphs used to compare the genetic algorithm and the vercov algorithm, it is known by construction that an optimum of quality 55 exists in the first test set, (respectively, an optimum of quality 110 exists for the second test set). This optimum is likely to be the global optimum, And it is found by the genetic algorithm at least once within the $N = 100$ runs that were performed.

mvcp200-01		mvcp200-02		mvcp200-03		mvcp200-04		mvcp200-05		PS202	
$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N	$f(\vec{x})$	N
172	2	172		172		172		172		134	100
174	3	174		174	1	174		174			
176	13	176	1	176	1	176	1	176	1		
178	14	178	1	178	3	178	4	178	1		
180	23	180	5	180	5	180	1	180	5		
182	14	182	11	182	5	182	9	182	5		
184	15	184	18	184	17	184	8	184	5		
186	7	186	16	186	19	186	16	186	13		
188	5	188	24	188	16	188	16	188	11		
190	3	190	15	190	16	190	16	190	21		
192	1	192	4	192	9	192	15	192	19		
> 192	0	> 192	5	> 192	8	> 192	14	> 192	19		
$\bar{f} = 180.98$		$\bar{f} = 186.46$		$\bar{f} = 186.92$		$\bar{f} = 188.06$		$\bar{f} = 189.16$			

Table 4: general schema for genetic algorithms

Moreover, the randomly constructed problems quickly become simpler for the genetic algorithm when the edge density is increased. For an edge density of $d \in \{0.4, 0.5\}$, the genetic algorithm almost surely finds the global optimum of the problems, while vercov algorithm gives a relatively terrible results.

In case of the regular graph after Papadimitriou and Steiglitz, the genetic algorithm is able to find the global optimum in about 2/3 of all runs, while the remaining runs identify a solution of quality $2k + 2$. Whereas, vercov algorithm gives a solution of quality $2k + 2$ in all of the 100 runs as expected. Recall that the graphs are constructed so as to force the vercov algorithm into yielding solutions of quality $2k + 2$. The random choice of edges in line 4 of the algorithm implies that nodes from either layers one and two, or layers two and three are involved in the solution being constructed, which in turn implies that two layers have to be part of the solution found by the algorithm [10]mvcp.

For the randomly constructed graphs, the vercov algorithm performs poorly. It barely finds solutions of quality 80 (for graphs of size 100) and 172 (for graphs of size 200). The average performance of the vercov algorithm (surprisingly) decreases as the edge density is increased. i.e. the problems become even harder for the vercov algorithm while they become much simpler for the genetic algorithm when the edge

density increases. The average fitness found by the genetic algorithm is notably better than the one found by the vercov algorithm. One can demonstrate this by calculating the ratio of the absolute (or equivalently the relative) error. For example, for the first problem instance, let $\bar{f}_1, \bar{f}_2$ be the average found by the genetic algorithm, vercov algorithm respectively. We have $(\bar{f}_2 - 55)/(\bar{f}_1 - 55) = (86.46 - 55)/(62.61 - 55) \approx 4 = 400\%$ that is, the error was reduced by a factor of 4.

5 Conclusion

In this work, we have demonstrated that genetic algorithms can be used in a straightforward way t good approximate solutions of the minimum vertex cover problem. Moreover, the results found by the genetic algorithm are better than those obtained from the best known traditional heuristic, the vercov algorithm. These, in addition to other good results obtained on, highly constrained combinatorial optimization problems such as the subset sum [7], minimum tardy task [7], and multiple knapsack problems [8], give strong evidence that genetic algorithms can yield good solutions for a wide range of hard combinatorial optimization problems for which solutions can be represented by binary strings. [4, 2, p. 400, p.] [1]

References

[1] D.A. Coley. *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific, 1999.

[2] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Artificial Intelligence. Addison-Wesley Publishing Company, 1989.

[3] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[4] S. Khuri and Th. Bäck. "An evolutionary heuristic for the minimum vertex cover problem". In: *Genetic Algorithms within the Framework of Evolutionary Computation–Proc. of the KI-94 Workshop*. Available at: `https://www.researchgate.net/publication/2581134_An_Evolutionary_Heuristic_for_the_Minimum_Vertex_Cover_Problem/overview`. Saarbrücken, Germany. 1994, pp. 86–90.