A Hybrid Genetic Algorithm with Wisdom-of-Crowds Aggregation for Approximating Minimum Vertex Covers

Justin Pham
Computer Engineering and Computer Science
Speed School of Engineering
University of Louisville, USA
jhpham02@louisville.edu

Abstract—**This project was about an NP complete Vertex Cover problem and solving it using a Genetic Algorithm (GA) that is enhanced with a Wisdom of Crowds (WoC) aggregation approach. The goal of this project was to approximate minimal vertex covers by evolving a population of candidate solutions using natural selection, mutation and coverage. The WoC aspect combined the top performing individuals from each generation into a combined solution which would guide future generations. A GUI was developed in order to visualize graph states and control parameters such as mutation rate and population size. Graphs were generated using the Erdős Rényi model, and then the results were exported to a CSV. Testing involved different graph sizes, densities, and GA parameters which showed that the hybrid algorithm of the GA with WoC produced smaller covers and faster convergence compared to the standard GA.**

I. Introduction

A lot of important systems in the real world can be modeled as graphs. This can include communication networks, social networks, and more. Being able to analyze and optimize these graphs can be difficult and a lot of times require solving problems computationally. One problem that we are exploring today is the Vertex Cover problem which asks for the smallest set of vertices that all cover all the edges of a graph. Even though it sounds simple,, finding the minimal cover can be difficult computationally, since graphs can grow very large. This problem is known as an NP complete problem, although the optimization version could be considered to be an NP-hard problem.

Since exact solutions aren't usually practical, a lot of researchers rely on algorithms that approximate the solution as well as heuristic methods that would lead to finding good but not perfect solutions. With all of these different kinds of algorithms, one called the Genetic Algorithm or GA was used. Genetic Algorithms draw inspiration from evolution itself, where it would use a population of candidate solutions that evolve over generations through the processes of natural selection, crossover, and mutation. This algorithm is often used for NP hard problems since they can explore large spaces that need to be searched without requiring strict structure.

While GAs can find quality solutions, the performance can also vary with different parameter choices and the quality of the earlier populations. Researchers have then tried to find different methods with ideas from collective intelligence which was then called the Wisdom of Crowds or WoC. This approach found that combining independent solutions can lead to solutions that perform better than individuals. WoC can then guide evolutionary algorithms towards a higher quality solution by finding patterns that appear across the experts in each population.

In this project, we explore the hybrid algorithm that integrated both a GA with a WoC aggregation approach in order to approximate minimal vertex covers. The goal was to see whether combining the two would lead to better solution quality, faster convergence, and greater stability across many different graph types. We implemented the algorithm, generated graphs using the Erdos Renyi model, and created a GUI to visualize the behavior of the algorithm.

II. Prior Work

The Vertex Cover problem is a classic NP-hard problem when trying to optimize it. It has a lot of applications in network security and more. Since exact algorithms to solve this problem are exponential in the worst case, researchers have looked at approximation algorithms, heuristics, and evolutionary search algorithms. This section below will review researched work related to topics such as NP-completedness, metaheuristics, swarm intelligence, and wisdom of crowds.

2.1 NP-Hardness and Vertex Cover

The Vertex Cover problem is a classic NP-hard optimization problem. NP-complete problems are when problems in NP are linked in complexity to all the other problems in NP which means that finding a polynomial time solution in one would mean that all other problems could be solved in polynomial time, where P = NP (Guo and Wei, 2022). NP-hardness is also researched often in the context of decision problems but can also be about optimization problems such as what we did with the Vertex Cover problem which can still be challenging since its nature is combinatorial (Imanparast, 2019). There are also many other optimization problems which include the TSP, 0/1 Knapsack, and more, which are NP-hard and also need special approaches in order to find solutions for them(Alridha et al., 2021).

As I stated before, a vertex cover is a set of vertices in a graph where every edge is incident to at least one vertex. The minimum cover would then be a subset with the smallest cardinality according to Uy and Abregana, 2015. Also, they said that Karp's work in 1972 proved the problem was NP-complete even with strong graph restrictions. Approximation algorithms have also already been researched for the Vertex Cover problem. Chen et al. (2015) proposed an algorithm where they used Dijkstra's algorithm to use a max shortest path value as the selection criteria which led to a solution with time complexity of O(n^3). This method shows that there are other strategies to approximating this problem.

2.2 Genetic Algorithms

Genetic algorithms draw inspiration from natural selection and genetics itself, as it is in the name, where populations of candidate solutions would evolve over time or generations using operators like selection, crossover, and mutation. Holland (2000) formed this approach by introducing concepts such as building blocks and the schema theorem which described how subsets of the genes that were highly fit would evolve through generations. The work done here shows how the GA can actually search through very complex search spaces efficiently.

GAs have also been applied to many NP-hard optimization problems. One is the FSSP or Flow Shop Scheduling Problem. In this case, a job based and machine based GA variants were used which showed efficient and very good performance compared to other heuristics (Syarif et al., 2021).

Hybrid GAs have also been proposed to enhance the performance by integrating local optimization operators (Ugurlu 2013). They presented a hybrid GA for the Vertex Cover problem that combined it with local optimization procedures which removed unnecessary vertices. The algorithm they used also introduced a heuristic mutation operator which adjusted which vertices were mutated based on their frequency. This let the GA avoid premature convergence and be more consistent.

A factor that can be critical to the performance of the GA is parameter tuning. If poor choices to setting parameters is made it can affect the convergence greatly. Aitpov et al. (2024) proposed a lazy parameter control method. In this, the GA parameters would be chosen randomly from a power law distribution in each iteration. This results from this study suggested that dynamic parameters can be superior than static parameters for complex problems.

So, these studies show that both a standard GA and hybrid GAs are good for handling problems involving optimization.

2.3 Swarm Intelligence and Metaheuristics

ACO or Ant Colony Optimization was developed by Dorigo and Stutzle (2004). It simulated artificial ants that built solutions using pheromone trails which balanced exploration and exploitation. This collectiveness can be related to wisdom of crowds where the individual agent's actions would aggregate to form a high quality solution for NP complete problems like the vertex cover.

PSO or Particle Swarm Optimization was introduced by Kennedy and Eberhart (1995). In this, particles would navigate a solution space using both their own best positions and the swarm's collective best position. Similarly to ACO, this showed how distributed populations could lead to effectively exploring more complex search spaces.

Hybrid evolutionary approaches in itself also integrate many metaheuristics to take advantage of different strengths. Shuli Hu et al. (2023) showed that combining genetic algorithms with local search was able to improve the quality of the solution and also the reliability of convergence for combinatorial problems. These hybrid approaches can then combine with population based exploration which leads to them being very effective for a problem like the Vertex Cover problem.

2.4 Wisdom of Crowds and Collective Decision Methods

The concept of the WoC or Wisdom of Crowds shows the aggregating the solutions of the collective of individuals can lead to better results which can be better than relying on various individual experts. (Surowiecki, 2004). This principle also shows how collective intelligence can lead to better results in decision making which affirms that WoC can be used in computational optimization.

Grimes and Breen (2019) then demonstrated WoC in computational systems which showed that aggregating solutions could reduce predictive error since the diversity in the crowd would improve overall accuracy. This suggested that metaheuristic algorithms could actually benefit a lot from aggregation mechanisms.

Qi et al. (2021) then implemented collective intelligence evolution using AOC which was guided by neural networks. This showed that combining an agent based solution generation with aggregation can outperform the traditional ACO. The application of WoC to a combinatorial setting showed that a consensus could lead to better performance.

Ashby and Yampolskiy (2011) then introduced the Wisdom of Artificial Crowds which was a post processing method for genetic algorithms. In this, multiple independent solutions would be aggregated to form a final superior solution. This treated the intermediate solutions from the GA as a crowd of agents instead and the WoAC shows that implementing WoC principles in metaheuristic optimization can lead to much better performance in these algorithms.

Overall, these studies showed that WoC aggregation aspects can be integrated into many algorithms including evolutionary and other metaheuristics which can lead to better and more optimal solutions for NP-complete problems.

2.5 Gaps

While the Vertex Cover problem is well studied and has been approached with many algorithms, there is still limited research on it being solved with a GA + WoC approach. Also, there are still limitations with the solution quality and the reliability of the convergence of the problem. The work done here addresses that gap as we combined the GA with the WoC aggregation mechanisms in order to try and improve on both the quality and consistency of solutions for the Vertex Cover Problem.

III. Proposed Approach: GA+WoC Hybrid Algorithm for Minimum Vertex Cover

In this project, we present the hybrid algorithm that combines a standard Genetic Algorithm with a Wisdom of Crowds approach. The GA would perform evolutionary search and the WoC aspect would serve as a strategy to guide the generations with a solution combined from the best performing individuals of each population. The overall design of the algorithm is trying to improve the quality of the solution, stabilize convergence, and also avoid premature stagnation which are all common problems when using evolutionary approaches to NP hard problems.

The method can be split in 5 phases that include graph generation, population initialization, evolution, expert aggregation, and visualization. The algorithm is explained below.

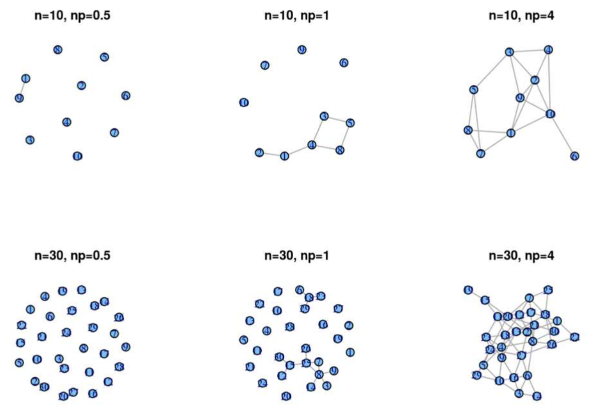
3.1 Graph Generation

We used the Erdos Renyi *G(n,p)* random graph model as the input. The parameters are below:

- n: number of vertices
- p: edge probability

With these parameters, the program samples every possible undirected edge independently using the probability p. This then generates a diverse graph with varying density and difficulty. The generation itself was integrated into the GUI which allows the user to change parameters as well as observe the behavior.

Erdos Renyi Model Sample Graphs:



3.2 Population Initialization

Each candidate vertex cover was represented as a binary vector

$$x = (x_1, x_2, \dots, x_n)$$

and 1 meant it was in the cover.

The initial population was also generated randomly uniform. This made sure that there was a high diversity in the early generations/stages of the search and made sure to avoid any structural bias.

3.3 Fitness Evaluation

The fitness function had 2 goals:

- Minimizing cover size
- Avoiding any uncovered edges

The function is defined as

$$fitness(x) = -\alpha * |x| - \beta * U(x)$$

where

- $|x|$ is the number of vertices selected
- $U(x)$ is the number of uncovered edges
- α and β are fixed weights where $\beta \gg \alpha$

This makes sure to prioritize feasibility and still biasing the search towards a minimal cover. Each generation all the individuals would have their fitness evaluated and then sorted.

3.4 Selection and Elitism

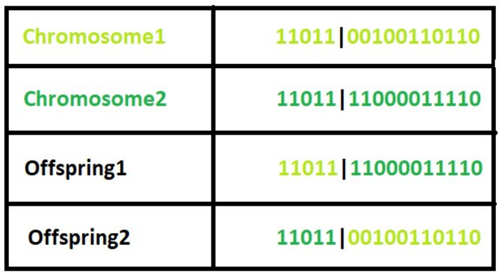
Next, we use elitism by copying the best performing individuals into the next generations. This makes sure that the population doesn't lose the top solutions. The rest of the pool is then created by selecting individuals with a probability proportional to the fitness rank, while also maintaining diversity.

3.5 Crossover

The reproduction happens using a single point crossover with the process below

- Two parents uniformly are chosen at random
- A crossover index *k* is randomly selected
- A child is formed by taking the first *k* bits from one parents and then *n – k* bits from the other parent

This makes it so that there can be a large scale recombination while maintaining the structure of the partial covers with potential.



Chromosome1	11011|00100110110
Chromosome2	11011|11000011110
Offspring1	11011|11000011110
Offspring2	11011|00100110110

Single Point Crossover

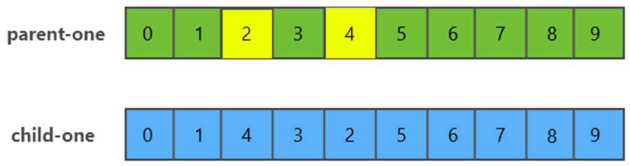
3.6 Mutation

In order to keep diversity but also prevent early convergence, we apply mutation with a controlled rate where it would flip a bit:

$$x_i \rightarrow 1 - x_i$$

The GUI also lets the user change the mutation rate parameter which lets users see the difference between exploration and exploitation.

Swap Mutation



3.7 Wisdom of Crowds Aggregation

When we enable WoC, there is a top fraction of individuals which can be changed by the user in the GUI that is used as the "expert" set. The experts are then aggregated through a consensus rule that is majority voted.

$$c_i = \begin{cases} 1 & \textit{if the majority of the experts have } x = 1 \\ 0 & \textit{if not} \end{cases}$$

c is a consensus vector which basically acts as the crowd aggregated solution which is meant to keep all the patterns from the experts.

Then, the consensus cover is put into the next generation to guide it towards a higher quality solution.

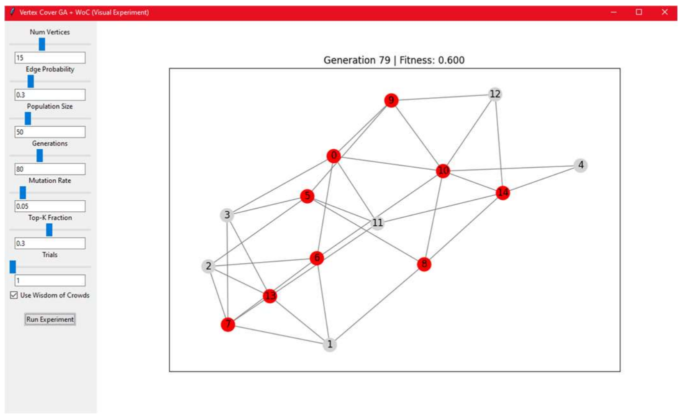
3.8 Runtime

The algorithm also allows the user to set a specific number of generations. This makes it so that the runtime can be predictable while still allowing the behavior of convergence to be analyzed.

3.9 Visualization

During execution, the GUI shows a live visualization of the graph. The vertices are highlighted in red if they are in the current best cover, or otherwise grey. The cover also evolves over time and it can be seen visually so that users can see the progression of the algorithm. All of the results are then saved into a CSV file. Also, multiple trials can be ran at the same time which allows for comparison between the hybrid approach and the standard GA and also different parameters using WoC.

Sample Run of GUI:



Vertex Cover GA + WoC (Visual Experiment)

Num Vertices: 15
Edge Probability: 0.3
Population Size: 50
Generations: 80
Mutation Rate: 0.05
Top-K Fraction: 0.3
Trials: 1
☑ Use Wisdom of Crowds

Generation 79 | Fitness: 0.600

CSV Sample Results:

```
False,15,0.3,50,80,0.05,0.3,1,10,0,0.6666666666666666,3.4
False,15,0.3,50,80,0.05,0.3,2,9,0,0.6,3.521
False,15,0.3,50,80,0.05,0.3,3,9,0,0.6,3.375
False,15,0.3,50,80,0.05,0.3,4,9,0,0.6,3.253
False,15,0.3,50,80,0.05,0.3,5,9,0,0.6,3.145
False,15,0.3,50,80,0.05,0.3,6,10,0,0.6666666666666666,3.5
False,15,0.3,50,80,0.05,0.3,7,9,0,0.6,3.222
```

```
False,15,0.3,50,80,0.05,0.3,8,8,0,0.5333333333333333,3.3
False,15,0.3,50,80,0.05,0.3,9,9,0,0.6,3.947
False,15,0.3,50,80,0.05,0.3,10,9,0,0.6,3.464
```

IV. Experimental Results

This section will show how well the standard Genetic Algorithm performed on the problem, and also how the GA with WoC approach compared to it. The performance was analyzed and assessed using randomly generated graphs using the Erdos Renyi model. Then, multiple controlled experiments were done with varying graph sizes, graph densities, and other parameters.

4.1 Data

All of the data used was generated rather than sourced from external sources. Each dataset corresponded to a random graph that was undirected and was created using the Erdos Renyi model. The parameters are below:

- *n*: number of vertices
- *p*: edge probability

For each pair of vertices, an edge was created with probability *p* which allowed control over the graph. The user was allowed to change both size and density. For example:

- Size (changed to 10,15,20,etc vertices)
- Density (p = 0.1 would lead to sparse graphs and p = 0.6 would lead to dense graphs)

So, a graph with n = 50 and p = 0.2 would have around 20% of all possible edges.

4.2 Test Design

15 controlled experiments were done for this project. In each experiment:

- Only one parameter was changed at one time
- All of the other parameters were kept as the baseline settings
- 10 trials were run per experiment
- Each trial would output to the CSV in the following format: (Use WoC, number of vertices, edge probability, population size, generations, mutation rate, top k fraction, trial number, cover size, number of uncovered edges, fitness, and finally runtime)

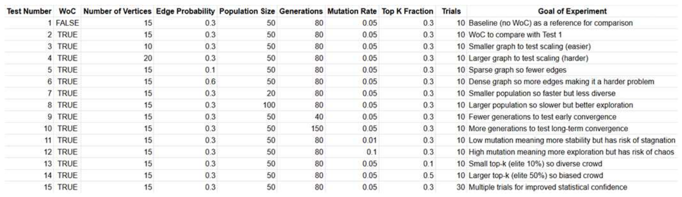
The baseline parameters are below:

- Number of Vertices: 15
- Edge Probability: 0.3
- Population Size: 50
- Number of Generations: 80
- Mutation Rate: 0.05
- Top K Fraction: 0.3
- Number of Trials: 10

Tests 1-15:

Test 1 - Set WoC to false. Goal was to have a baseline with just the standard GA for comparison
Test 2 - Set WoC to true. Goal was to have a baseline with WoC
Test 3 - Set number of vertices to 10. Goal was to test a smaller graph to see scaling
Test 4 – Set number of vertices to 20. Goal was to test a larger graph to see scaling
Test 5 – Set edge probability to 0.1. Goal was to test a sparse graph with fewer edges
Test 6 – Set edge probability to 0.6. Goal was to test a dense graph with more edges, making it a harder problem
Test 7 – Set population size to 20. Goal was to test a smaller population which means it would be faster but less diverse
Test 8 – Set population size to 100. Goal was to test a larger population which means it would be slower but with better exploration
Test 9 – Set number of generations to 40. Goal was to test early convergence
Test 10 – Set number of generations to 150. Goal was to test long term convergence
Test 11 – Set mutation rate to 0.01. Goal was to test a low mutation with more stability but with a risk of stagnation
Test 12 – Set mutation rate to 0.1. Goal was to test a high mutation which meant more exploration but could have a risk of chaos
Test 13 – Set top k fraction to 0.1. Goal was to test a small 10% elite group so testing a more diverse crowd
Test 14 – Set top k fraction to 0.5. Goal was to test a larger 50% elite group to test a more biased crowd
Test 15 – Set number of trials to 30. Goal was to test multiple trials to improve statistical confidence

Chart:



Test Number	WoC	Number of Vertices	Edge Probability	Population Size	Generations	Mutation Rate	Top K Fraction	Trials	Goal of Experiment
1	FALSE	15	0.3	50	80	0.05	0.3	10	Baseline (no WoC) as a reference for comparison
2	TRUE	15	0.3	50	80	0.05	0.3	10	WoC to compare with Test 1
3	TRUE	10	0.3	50	80	0.05	0.3	10	Smaller graph to test scaling (easier)
4	TRUE	20	0.3	50	80	0.05	0.3	10	Larger graph to test scaling (harder)
5	TRUE	15	0.1	50	80	0.05	0.3	10	Sparse graph so fewer edges
6	TRUE	15	0.6	50	80	0.05	0.3	10	Dense graph so more edges making it a harder problem
7	TRUE	15	0.3	20	80	0.05	0.3	10	Smaller population so faster but less diverse
8	TRUE	15	0.3	100	80	0.05	0.3	10	Larger population so slower but better exploration
9	TRUE	15	0.3	50	40	0.05	0.3	10	Fewer generations to test early convergence
10	TRUE	15	0.3	50	150	0.05	0.3	10	More generations to test long-term convergence
11	TRUE	15	0.3	50	80	0.01	0.3	10	Low mutation meaning more stability but has risk of stagnation
12	TRUE	15	0.3	50	80	0.1	0.3	10	High mutation meaning more exploration but has risk of chaos
13	TRUE	15	0.3	50	80	0.05	0.1	10	Small top-k (elite 10%) so diverse crowd
14	TRUE	15	0.3	50	80	0.05	0.5	10	Larger top-k (elite 50%) so biased crowd
15	TRUE	15	0.3	50	80	0.05	0.3	30	Multiple trials for improved statistical confidence

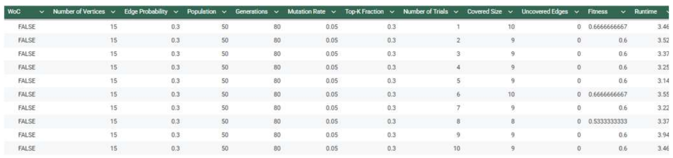
4.3 Results

4.3.1 Overall Performance

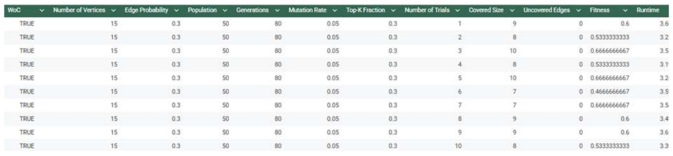
Through all the tests and experiments, both algorithms were able to find valid vertex covers with 0 uncovered edges pretty consistently. This showed that the fitness function and the penalty system worked well. However, the GA with the WoC approach was able to find smaller covers consistently compared to the standard GA which showed that crowd aggregation led to better quality in the convergence.

4.3.2 GA vs GA + WoC

Table for Test 1 and 2:



WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
FALSE	15	0.3	50	80	0.05	0.3	1	10	0	0.6666666667	3.46
FALSE	15	0.3	50	80	0.05	0.3	2	9	0	0.6	3.52
FALSE	15	0.3	50	80	0.05	0.3	3	9	0	0.6	3.37
FALSE	15	0.3	50	80	0.05	0.3	4	9	0	0.6	3.25
FALSE	15	0.3	50	80	0.05	0.3	5	9	0	0.6	3.14
FALSE	15	0.3	50	80	0.05	0.3	6	10	0	0.6666666667	3.55
FALSE	15	0.3	50	80	0.05	0.3	7	9	0	0.6	3.22
FALSE	15	0.3	50	80	0.05	0.3	8	8	0	0.5333333333	3.37
FALSE	15	0.3	50	80	0.05	0.3	9	9	0	0.6	3.94
FALSE	15	0.3	50	80	0.05	0.3	10	9	0	0.6	3.46



WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.3	50	80	0.05	0.3	1	9	0	0.6	3.6
TRUE	15	0.3	50	80	0.05	0.3	2	8	0	0.5333333333	3.2
TRUE	15	0.3	50	80	0.05	0.3	3	10	0	0.6666666667	3.5
TRUE	15	0.3	50	80	0.05	0.3	4	8	0	0.5333333333	3.1
TRUE	15	0.3	50	80	0.05	0.3	5	10	0	0.6666666667	3.2
TRUE	15	0.3	50	80	0.05	0.3	6	7	0	0.4666666667	3.5
TRUE	15	0.3	50	80	0.05	0.3	7	7	0	0.6666666667	3.5
TRUE	15	0.3	50	80	0.05	0.3	8	9	0	0.6	3.4
TRUE	15	0.3	50	80	0.05	0.3	9	9	0	0.6	3.6
TRUE	15	0.3	50	80	0.05	0.3	10	8	0	0.5333333333	3.3

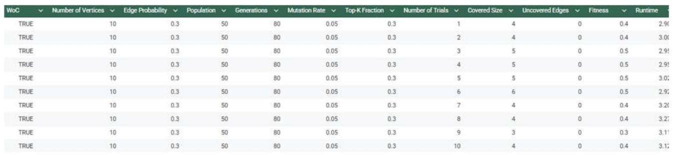
Comparing Test 1 and Test 2:
- The standard GA found cover sizes between 8 and 10
- The GA + WoC found cover sizes between 7 and 10
- The hybrid approach reduced the variance and improved the consistency of the cover sizes
- No tests had uncovered edges which means both worked
- Average runtime between both tests were very similar, which means adding the WoC didn't lead to a major computational cost

Average Cover Size:
- Standard GA: 9.1
- GA + WoC: 8.5

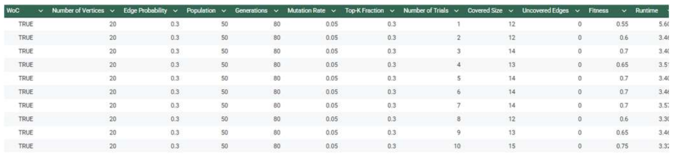
Best Cover Size:
- Standard GA: 8
- GA + WoC: 7

4.3.3 Effect of Graph Size

Table for Test 3 and 4:

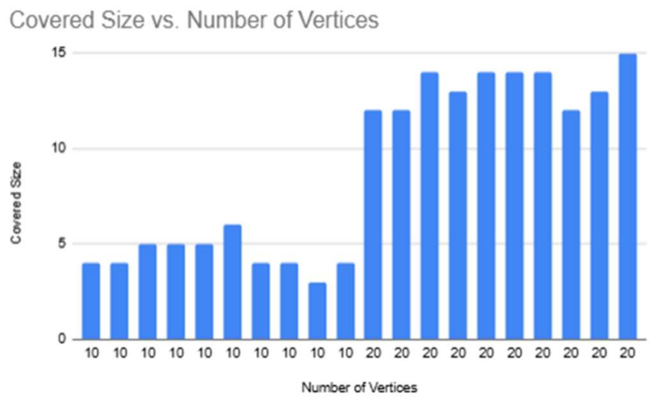


WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	10	0.3	50	80	0.05	0.3	1	4	0	0.4	2.9
TRUE	10	0.3	50	80	0.05	0.3	2	4	0	0.4	3.0
TRUE	10	0.3	50	80	0.05	0.3	3	5	0	0.5	2.9
TRUE	10	0.3	50	80	0.05	0.3	4	5	0	0.5	2.9
TRUE	10	0.3	50	80	0.05	0.3	5	5	0	0.5	3.0
TRUE	10	0.3	50	80	0.05	0.3	6	6	0	0.5	2.9
TRUE	10	0.3	50	80	0.05	0.3	7	4	0	0.4	3.2
TRUE	10	0.3	50	80	0.05	0.3	8	4	0	0.4	3.2
TRUE	10	0.3	50	80	0.05	0.3	9	3	0	0.3	3.1
TRUE	10	0.3	50	80	0.05	0.3	10	4	0	0.4	3.1



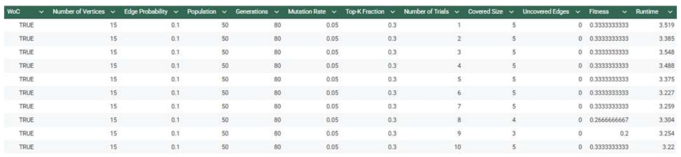
WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	20	0.3	50	80	0.05	0.3	1	12	0	0.55	5.6
TRUE	20	0.3	50	80	0.05	0.3	2	12	0	0.6	3.4
TRUE	20	0.3	50	80	0.05	0.3	3	14	0	0.7	3.4
TRUE	20	0.3	50	80	0.05	0.3	4	13	0	0.65	3.5
TRUE	20	0.3	50	80	0.05	0.3	5	14	0	0.7	3.4
TRUE	20	0.3	50	80	0.05	0.3	6	14	0	0.7	3.4
TRUE	20	0.3	50	80	0.05	0.3	7	14	0	0.7	3.5
TRUE	20	0.3	50	80	0.05	0.3	8	12	0	0.6	3.3
TRUE	20	0.3	50	80	0.05	0.3	9	13	0	0.65	3.4
TRUE	20	0.3	50	80	0.05	0.3	10	15	0	0.75	3.3

Comparing Test 3 and 4:
- These tests were very predictable as the larger graphs led to a larger average cover and a smaller graph would have a smaller average cover
- Runtime also slightly increased with the increase in graph size

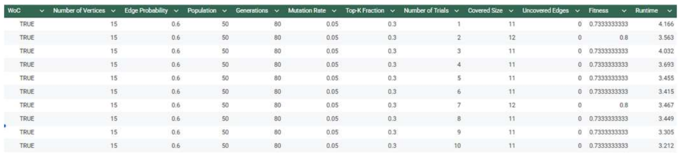


4.3.4 Effect of Graph Density

Table for Test 5 and 6:



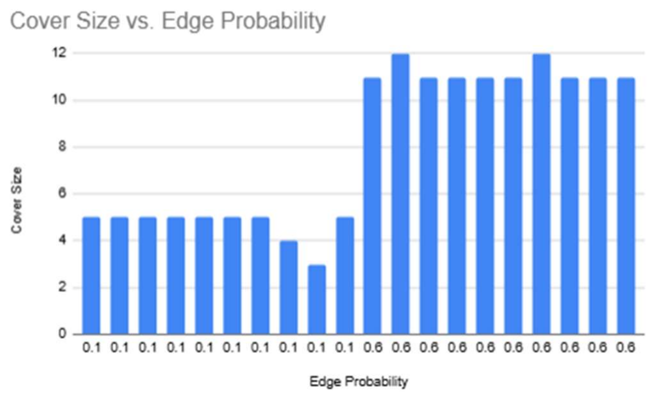
WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.1	50	80	0.05	0.3	1	5	0	0.3333333333	3.519
TRUE	15	0.1	50	80	0.05	0.3	2	5	0	0.3333333333	3.385
TRUE	15	0.1	50	80	0.05	0.3	3	5	0	0.3333333333	3.548
TRUE	15	0.1	50	80	0.05	0.3	4	5	0	0.3333333333	3.488
TRUE	15	0.1	50	80	0.05	0.3	5	5	0	0.3333333333	3.375
TRUE	15	0.1	50	80	0.05	0.3	6	5	0	0.3333333333	3.227
TRUE	15	0.1	50	80	0.05	0.3	7	5	0	0.3333333333	3.259
TRUE	15	0.1	50	80	0.05	0.3	8	4	0	0.2666666667	3.304
TRUE	15	0.1	50	80	0.05	0.3	9	3	0	0.2	3.254
TRUE	15	0.1	50	80	0.05	0.3	10	5	0	0.3333333333	3.22



WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.6	50	80	0.05	0.3	1	11	0	0.7333333333	4.166
TRUE	15	0.6	50	80	0.05	0.3	2	12	0	0.8	3.563
TRUE	15	0.6	50	80	0.05	0.3	3	11	0	0.7333333333	4.032
TRUE	15	0.6	50	80	0.05	0.3	4	11	0	0.7333333333	3.693
TRUE	15	0.6	50	80	0.05	0.3	5	11	0	0.7333333333	3.455
TRUE	15	0.6	50	80	0.05	0.3	6	11	0	0.7333333333	3.415
TRUE	15	0.6	50	80	0.05	0.3	7	12	0	0.8	3.467
TRUE	15	0.6	50	80	0.05	0.3	8	11	0	0.7333333333	3.449
TRUE	15	0.6	50	80	0.05	0.3	9	11	0	0.7333333333	3.305
TRUE	15	0.6	50	80	0.05	0.3	10	11	0	0.7333333333	3.212

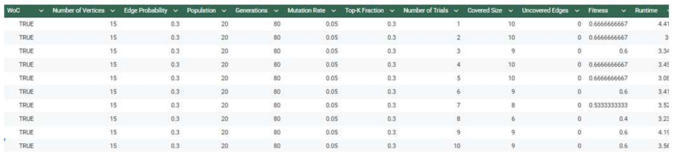
Comparing Test 5 and 6:

- This was also predictable as dense graphs made it so there were more vertices in the cover
- Sparse graphs led to a smaller cover size

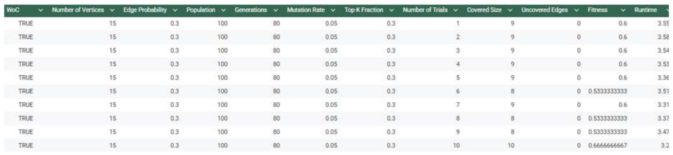


4.3.5 Effect of Population Parameter

Tables for Test 7 and 8:



WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.3	20	80	0.05	0.3	1	10	0	0.6666666667	4.41
TRUE	15	0.3	20	80	0.05	0.3	2	10	0	0.6666666667	3
TRUE	15	0.3	20	80	0.05	0.3	3	9	0	0.6	3.3
TRUE	15	0.3	20	80	0.05	0.3	4	10	0	0.6666666667	3.4
TRUE	15	0.3	20	80	0.05	0.3	5	10	0	0.6666666667	3.0
TRUE	15	0.3	20	80	0.05	0.3	6	9	0	0.6	3.41
TRUE	15	0.3	20	80	0.05	0.3	7	8	0	0.5333333333	3.5
TRUE	15	0.3	20	80	0.05	0.3	8	6	0	0.4	3.2
TRUE	15	0.3	20	80	0.05	0.3	9	9	0	0.6	4.1
TRUE	15	0.3	20	80	0.05	0.3	10	9	0	0.6	3.5



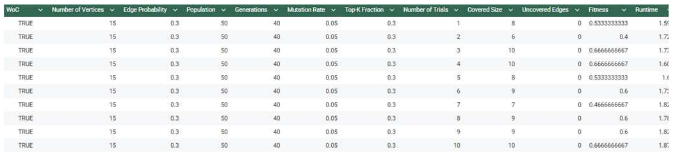
WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.3	100	80	0.05	0.3	1	9	0	0.6	3.55
TRUE	15	0.3	100	80	0.05	0.3	2	9	0	0.6	3.58
TRUE	15	0.3	100	80	0.05	0.3	3	9	0	0.6	3.54
TRUE	15	0.3	100	80	0.05	0.3	4	9	0	0.6	3.53
TRUE	15	0.3	100	80	0.05	0.3	5	9	0	0.6	3.36
TRUE	15	0.3	100	80	0.05	0.3	6	8	0	0.5333333333	3.51
TRUE	15	0.3	100	80	0.05	0.3	7	9	0	0.6	3.31
TRUE	15	0.3	100	80	0.05	0.3	8	8	0	0.5333333333	3.37
TRUE	15	0.3	100	80	0.05	0.3	9	8	0	0.5333333333	3.47
TRUE	15	0.3	100	80	0.05	0.3	10	10	0	0.6666666667	3.2

Comparing Test 7 and 8:

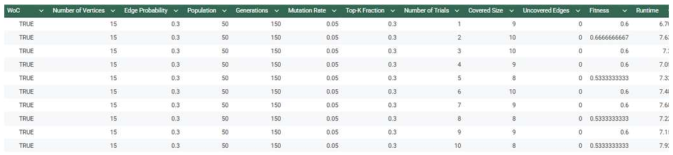
- A very small population led to more variability in the covers with a slower runtime
- A large population of 100 had a similar performance to the baseline of 50 but led to a higher runtime

4.3.6 Effect of Generations Parameter

Tables for Test 9 and 10:



WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.3	50	40	0.05	0.3	1	8	0	0.5333333333	1.5
TRUE	15	0.3	50	40	0.05	0.3	2	6	0	0.4	1.7
TRUE	15	0.3	50	40	0.05	0.3	3	10	0	0.6666666667	1.7
TRUE	15	0.3	50	40	0.05	0.3	4	10	0	0.6666666667	1.6
TRUE	15	0.3	50	40	0.05	0.3	5	8	0	0.5333333333	1.
TRUE	15	0.3	50	40	0.05	0.3	6	9	0	0.6	1.7
TRUE	15	0.3	50	40	0.05	0.3	7	7	0	0.4666666667	1.8
TRUE	15	0.3	50	40	0.05	0.3	8	9	0	0.6	1.7
TRUE	15	0.3	50	40	0.05	0.3	9	9	0	0.6	1.8
TRUE	15	0.3	50	40	0.05	0.3	10	10	0	0.6666666667	1.8



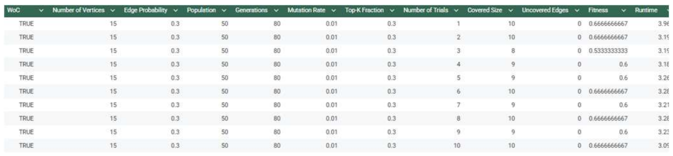
WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.3	50	150	0.05	0.3	1	9	0	0.6	6.7
TRUE	15	0.3	50	150	0.05	0.3	2	10	0	0.6666666667	7.6
TRUE	15	0.3	50	150	0.05	0.3	3	10	0	0.6	7.
TRUE	15	0.3	50	150	0.05	0.3	4	9	0	0.6	7.0
TRUE	15	0.3	50	150	0.05	0.3	5	8	0	0.5333333333	7.3
TRUE	15	0.3	50	150	0.05	0.3	6	10	0	0.6	7.4
TRUE	15	0.3	50	150	0.05	0.3	7	9	0	0.6	7.6
TRUE	15	0.3	50	150	0.05	0.3	8	8	0	0.5333333333	7.2
TRUE	15	0.3	50	150	0.05	0.3	9	9	0	0.6	7.1
TRUE	15	0.3	50	150	0.05	0.3	10	8	0	0.5333333333	7.9

Comparing Test 9 and 10:

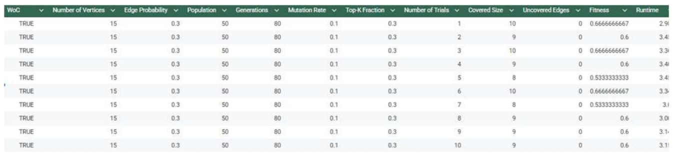
- With fewer generations, the population stagnated earlier which led to covers that were larger than it should be
- With more generations, this led to a higher runtime but was able to refine the cover more\

4.3.7 Effect of Mutation Rate

Tables for Test 11 and 12:



WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.3	50	80	0.01	0.3	1	10	0	0.6666666667	3.9
TRUE	15	0.3	50	80	0.01	0.3	2	10	0	0.6666666667	3.1
TRUE	15	0.3	50	80	0.01	0.3	3	8	0	0.5333333333	3.1
TRUE	15	0.3	50	80	0.01	0.3	4	9	0	0.6	3.1
TRUE	15	0.3	50	80	0.01	0.3	5	9	0	0.6	3.2
TRUE	15	0.3	50	80	0.01	0.3	6	10	0	0.6666666667	3.2
TRUE	15	0.3	50	80	0.01	0.3	7	9	0	0.6	3.21
TRUE	15	0.3	50	80	0.01	0.3	8	10	0	0.6666666667	3.2
TRUE	15	0.3	50	80	0.01	0.3	9	9	0	0.6	3.2
TRUE	15	0.3	50	80	0.01	0.3	10	10	0	0.6666666667	3.0



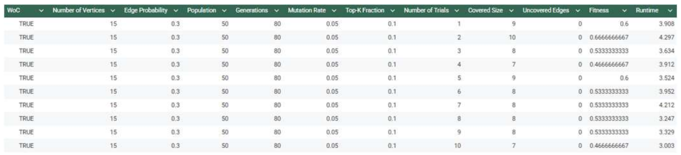
WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.3	50	80	0.1	0.3	1	10	0	0.6666666667	2.9
TRUE	15	0.3	50	80	0.1	0.3	2	9	0	0.6	3.4
TRUE	15	0.3	50	80	0.1	0.3	3	10	0	0.6666666667	3.3
TRUE	15	0.3	50	80	0.1	0.3	4	9	0	0.6	3.4
TRUE	15	0.3	50	80	0.1	0.3	5	8	0	0.5333333333	3.4
TRUE	15	0.3	50	80	0.1	0.3	6	10	0	0.6666666667	3.3
TRUE	15	0.3	50	80	0.1	0.3	7	8	0	0.5333333333	3.
TRUE	15	0.3	50	80	0.1	0.3	8	9	0	0.6	3.0
TRUE	15	0.3	50	80	0.1	0.3	9	9	0	0.6	3.1
TRUE	15	0.3	50	80	0.1	0.3	10	9	0	0.6	3.1

Comparing Test 11 and 12:

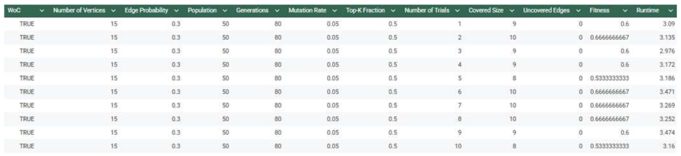
- A low mutation rate led to fast convergence but also had the risk of premature convergence
- Increasing the mutation rate led to greater diversity so test 12 had better final covers than test 11

4.3.8 Effect of Top K Fraction

Tables for Test 13 and 14:



WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.3	50	80	0.05	0.1	1	9	0	0.6	3.908
TRUE	15	0.3	50	80	0.05	0.1	2	10	0	0.6666666667	4.297
TRUE	15	0.3	50	80	0.05	0.1	3	8	0	0.5333333333	3.634
TRUE	15	0.3	50	80	0.05	0.1	4	7	0	0.4666666667	3.912
TRUE	15	0.3	50	80	0.05	0.1	5	9	0	0.6	3.524
TRUE	15	0.3	50	80	0.05	0.1	6	8	0	0.5333333333	3.952
TRUE	15	0.3	50	80	0.05	0.1	7	8	0	0.5333333333	4.212
TRUE	15	0.3	50	80	0.05	0.1	8	8	0	0.5333333333	3.247
TRUE	15	0.3	50	80	0.05	0.1	9	8	0	0.5333333333	3.329
TRUE	15	0.3	50	80	0.05	0.1	10	7	0	0.4666666667	3.003



WoC	Number of Vertices	Edge Probability	Population	Generations	Mutation Rate	Top-K Fraction	Number of Trials	Covered Size	Uncovered Edges	Fitness	Runtime
TRUE	15	0.3	50	80	0.05	0.5	1	9	0	0.6	3.09
TRUE	15	0.3	50	80	0.05	0.5	2	10	0	0.6666666667	3.135
TRUE	15	0.3	50	80	0.05	0.5	3	9	0	0.6	2.976
TRUE	15	0.3	50	80	0.05	0.5	4	9	0	0.6	3.172
TRUE	15	0.3	50	80	0.05	0.5	5	8	0	0.5333333333	3.186
TRUE	15	0.3	50	80	0.05	0.5	6	10	0	0.6666666667	3.471
TRUE	15	0.3	50	80	0.05	0.5	7	10	0	0.6666666667	3.269
TRUE	15	0.3	50	80	0.05	0.5	8	10	0	0.6666666667	3.252
TRUE	15	0.3	50	80	0.05	0.5	9	9	0	0.6	3.474
TRUE	15	0.3	50	80	0.05	0.5	10	8	0	0.5333333333	3.16

Comparing Test 13 and 14:

- A small top k fraction where the WoC relied on a smaller elite group led to a strong guidance earlier in the convergence but less selection led to the consensus being more sensitive
- A large top k fraction led to a more balanced and stable consensus. The cover also reflected the overall population more

4.3.9 Runtime

The runtime across all the tests:
- Ranged from 1.5 to 5.5 seconds
- Scaled with number of vertices, population size, and number of generations

V. Conclusion

We presented a hybrid algorithm combining a Genetic Algorithm with the Wisdom of Crowds approach to solve the Vertex Cover problem. This implementation added a consensus cover that would be reinserted into the population guide future evolutionary searches. Then we analyzed the method through varying graph sizes, densities, and other parameter settings. The results showed that the hybrid approach consistently performed better than the standard GA with it being more stable with its solution, and better convergence behavior.

To do further research, we would run more intense and complex problems to gain more data to analyze. However, our implementation is also able to be replicated so others can produce the same results. This hybrid method gives a good baseline towards analyzing evolutionary as well as metaheuristic approaches to an NP-hard problem such as the Vertex Cover problem.

REFERENCES

[1] Alridha, A., Salman, A. M., & Al-Jilawi, A. S. (2021). The applications of NP-hardness optimizations problem. Journal of Physics: Conference Series, 1818(1), 012179. https://doi.org/10.1088/1742-6596/1818/1/012179.

[2] Antipov, D., Buzdalov, M., & Doerr, B. (2023). Lazy parameter tuning and control: Choosing all parameters randomly from a power-law distribution. Algorithmica, 86(2), 442–484. https://doi.org/10.1007/s00453-023-01098-z.

[3] Ashby, L. H., & Yampolskiy, R. V. (2011). Genetic algorithm and wisdom of artificial crowds algorithm applied to light up. 2011 16th International Conference on Computer Games (CGAMES), 27–32. https://doi.org/10.1109/cgames.2011.6000341.

[4] Chen, J., Kou, L., & Cui, X. (2016). An approximation algorithm for the minimum vertex cover problem. Procedia Engineering, 137, 180–185. https://doi.org/10.1016/j.proeng.2016.01.248.

[5] Dorigo, M., & Stützle, T. (2004). Ant colony optimization. MIT Press..

[6] Grimes, S., & Breen, D. E. (2019). WOC-Bots: An agent-based approach to decision-making. Applied Sciences, 9(21), 4653. https://doi.org/10.3390/app9214653.

[7] Guo, J., & Wei, A. (2022). NP-Completedness..

[8] Holland, J. H. (2000). Building blocks, cohort genetic algorithms, and hyperplane-defined functions. Evolutionary Computation, 8(4), 373–391. https://doi.org/10.1162/106365600568220.

[9] Hu, S., Liu, H., Wu, X., Li, R., Zhou, J., & Wang, J. (2019). A hybrid framework combining genetic algorithm with iterated local search for the dominating tree problem. Mathematics, 7(4), 359. https://doi.org/10.3390/math7040359.

[10] Imparast, M. (2024). Challenges and Strategies for Tackling NP-Hard Problems..

[11] Kennedy, J., & Eberhart, R. (n.d.). Particle swarm optimization. Proceedings of ICNN'95 - International Conference on Neural Networks, 4, 1942–1948. https://doi.org/10.1109/icnn.1995.488968.

[12] Qi, X., Gan, Z., Liu, C., Xu, Z., Zhang, X., Li, W., & Ouyang, C. (2021). Collective intelligence evolution using ant colony optimization and Neural Networks. Neural Computing and Applications, 33(19), 12721–12735. https://doi.org/10.1007/s00521-021-05918-7.

[13] Surowiecki, J. (2005). The Wisdom of Crowds. Anchor Books..

[14] Syarif, A., Wamiliana, W., Lumbanraja, P., & Gen, M. (2020). Study on genetic algorithm (GA) approaches for solving flow shop scheduling problem (FSSP). IOP Conference Series: Materials Science and Engineering, 857(1), 012009. https://doi.org/10.1088/1757-899x/857/1/012009.

[15] Ugurlu, O. (2013). A NEW HYBRID GENETIC ALGORITHM FOR VERTEX COVER PROBLEM..

[16] Uy, J. A., & Abregana, V. P. (2015). Revisiting the vertex cover of graphs. Applied Mathematical Sciences, 9, 5707–5714. https://doi.org/10.12988/ams.2015.56455.