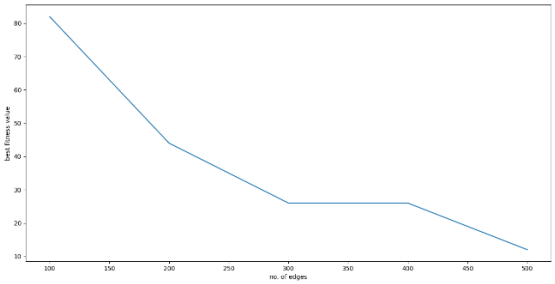
Artificial Intelligence (CS F407)

Aryan Gupta (2019A7PS0017G)

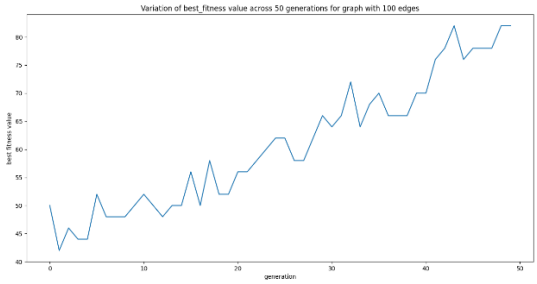
Assignment-1 – Vertex 3-Coloring using Genetic Algorithm

Part (a).

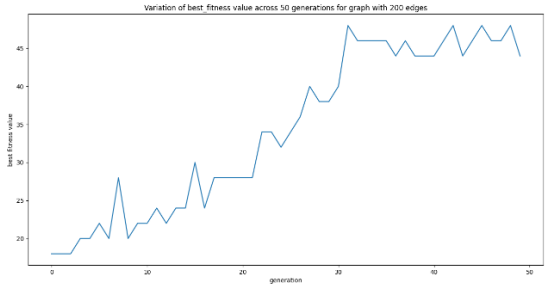
1. Best fitness value (%age of vertices) vs Number of Edges:



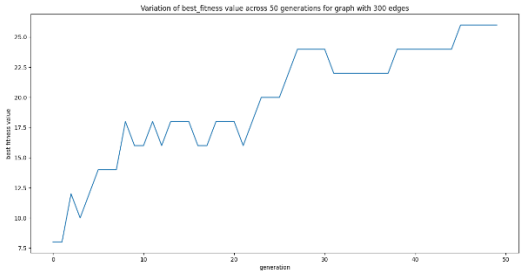
2. Variation of best_fitness value across 50 generations for graphs with different edge size set:



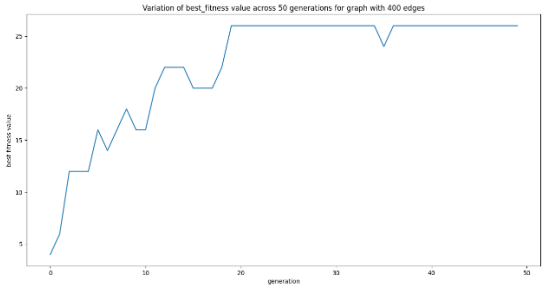
100 Edges



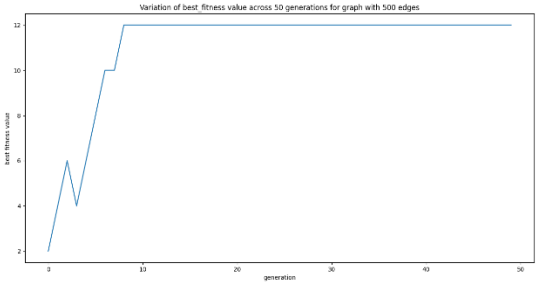
200 Edges



300 Edges



400 Edges



500 Edges

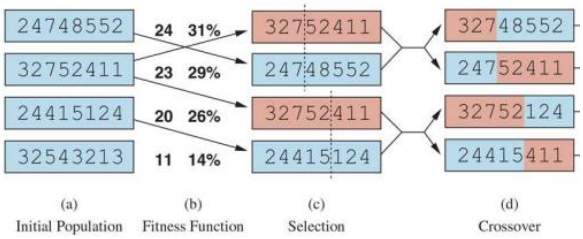
Part (b).

Appendix:

- num_edges: no. of edges in graph
- num_gen: max no. of generations for which the algo can run
- popln_sz: population size (no. of states in current generation)
- gens_before_restart: generations to wait for before randomly restarting from new state
- mutation_rate: probability with which mutation should be tried in given state

Improvements made to the textbook implementation:

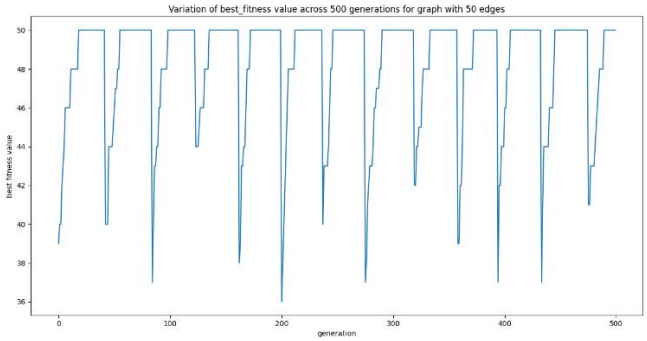
- **Elitism and Culling**
  - Selected *popln_sz* random pairs of parents, and each pair reproduced 2 children:



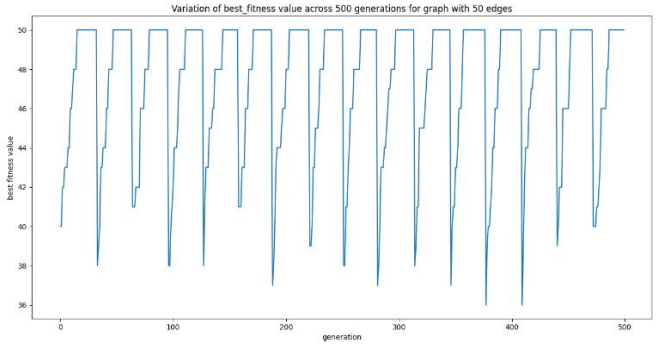
  - Among the *3*popln_sz* states (*popln_sz* states in current gen and *2*popln_sz* children), *popln_sz* number of states are chosen with the maximum fitness values.
- **Hyperparameter Tuning**
  - Number of generations: Since the graph converges in about 30-40 generations (for >= 200 edges), running the algo for an extravagantly high number of generations (say, 10000) does not help.
  - Population size:
    For a graph having **200 edge-size**:
    - A large population size, eg. >2000, takes a high amount of time to run for each generation.
    - The algorithm could run for only 44 generations in 45 secs for a population size of 5000 states, and just 10 generations for 10000 population size.
    - In the range of 100-1000, population size of <400 and >700 did not return consistent results; The fitness values obtained (experimenting over 20 graphs) ranged from 28 to 31 vertices.
    - Population size 500 proved to be a sweet spot with consistent results of 31 or 32 fitness value.
    - Graph having **300 edges:** For a population size of 1000, fitness value peaked at 25 vertices, but mode value remained 23 which is the same as that of 500 population size. Thus, not so helpful.
    - Graph having **500 edges:** For a population size of 2000, only about 40-50 generations could be run in 45 secs. The population randomly restarted just once, and the results obtained were poor.
- **Random Restarts**
  - If fitness value does not improve over *gens_before_restart*, a new population is generated with *popn_sz* random states.
    Note: In the implementation, the best fitness value per generation never decrements since elitism makes sure that the best parent is also retained in the new gen.
  - The optimal *gens_before_restart* value is found to be 20.
  - A low value like 10 or 15 would not give enough chance to a start state to prove its worth, and a high value such as 40 or 50 resulted in a waste of time and no. of generations for which the algo is run. (Also, as mentioned earlier, the graph converges in 40 generations.)
- **Mutation:**
  - Mutation is tried on the children reproduced with a probability of *mutation_rate*, set to 10%. A node is selected at random and is assigned a colour at random (chosen from all three options).
  - Thus, the overall probability that a state will have a mutated node is 6.66%.
  - Mutation actually improves the fitness values significantly. For 500 edge set size, introducing mutations increased the fitness value from 13 to 16 (consistently).

Results:

1. **50 Edges:** (Graph provided in CSV file)
   - **i.** Population size: 1000, Gens before restart: 30



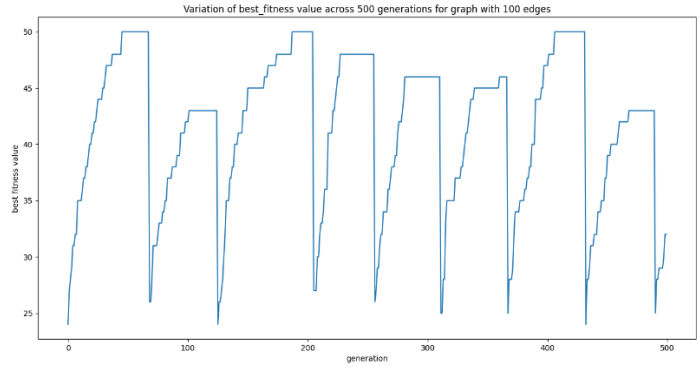
   - **ii.** Population size: 500, Gens before restart: 20



As can be seen, popln_sz = 500 consistently returns fitness value of 50 vertices.

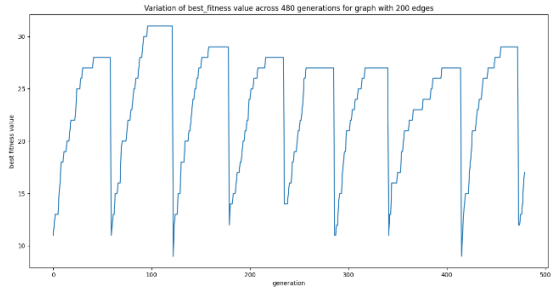
```
C:\Users\ARYAN GUPTA\Desktop\Codes\AI\Assignment1>2019A7PS0017G_ARYAN.py
Roll no.: 2019A7PS0017G
Number of edges: 50
Best state: [{0: 'r'}, {1: 'r'}, {2: 'r'}, {3: 'g'}, {4: 'g'}, {5: 'g'}, {6: 'r'}, {7: 'b'}, {8: 'g'}, {9: 'g'}, {10: 'r'}, {11: 'b'}, {12: 'r'}, {13: 'r'}, {14: 'r'}, {15: 'g'}
, {16: 'g'}, {17: 'r'}, {18: 'g'}, {19: 'g'}, {20: 'g'}, {21: 'g'}, {22: 'r'}, {23: 'r'}, {24: 'r'}, {25: 'r'}, {26: 'r'}, {27: 'b'}, {28: 'b'}, {29: 'b'}, {30: 'r'}, {31: 'g'},
 {32: 'g'}, {33: 'g'}, {34: 'g'}, {35: 'b'}, {36: 'b'}, {37: 'b'}, {38: 'b'}, {39: 'g'}, {40: 'g'}, {41: 'g'}, {42: 'g'}, {43: 'r'}, {44: 'b'}, {45: 'r'}, {46: 'g'}, {47: 'r'},
{48: 'b'}, {49: 'b'}]
Fitness value of best state: 50
Time taken: 13 secs
```

2. **100 Edges:** (Graph provided in CSV file)
   - **i.** Population size: 500, Gens before restart: 20

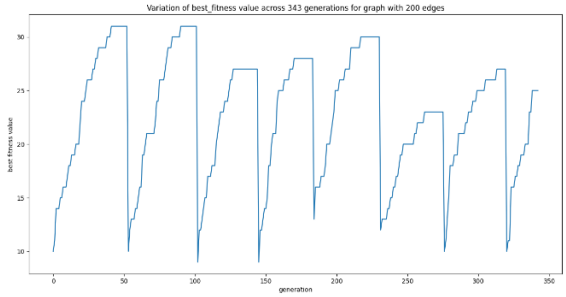



```
C:\Users\ARYAN GUPTA\Desktop\Codes\AI\Assignment1>2019A7PS0017G_ARYAN.py
Roll no.: 2019A7PS0017G
Number of edges: 100
Best state: [{0: 'r'}, {1: 'r'}, {2: 'g'}, {3: 'g'}, {4: 'b'}, {5: 'r'}, {6: 'b'}, {7: 'g'}, {8: 'r'}, {9: 'g'}, {10: 'r'}, {11: 'r'}, {12: 'r'}, {13: 'r'}, {14: 'r'}, {15: 'g'}
, {16: 'r'}, {17: 'b'}, {18: 'r'}, {19: 'g'}, {20: 'b'}, {21: 'g'}, {22: 'r'}, {23: 'b'}, {24: 'r'}, {25: 'r'}, {26: 'g'}, {27: 'r'}, {28: 'b'}, {29: 'g'}, {30: 'r'}, {31: 'b'},
 {32: 'g'}, {33: 'b'}, {34: 'r'}, {35: 'g'}, {36: 'b'}, {37: 'r'}, {38: 'b'}, {39: 'b'}, {40: 'b'}, {41: 'r'}, {42: 'b'}, {43: 'b'}, {44: 'r'}, {45: 'r'}, {46: 'g'}, {47: 'r'},
{48: 'r'}, {49: 'b'}]
Fitness value of best state: 50
Time taken: 41 secs
```

3. **200 Edges:** (Graph provided in CSV file)
    i. Population size: 1000

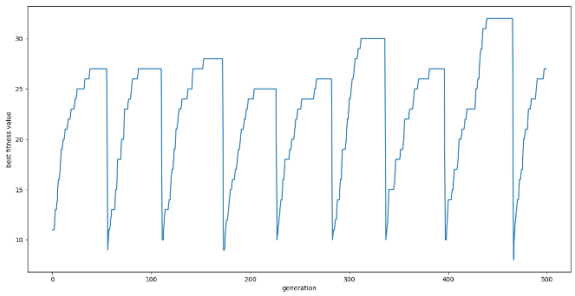


Gens before restart: 30



Gens before restart: 20

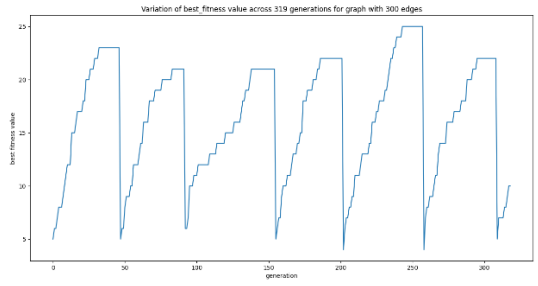
    ii. <u>Population size: 500</u>, Gens before restart: 20



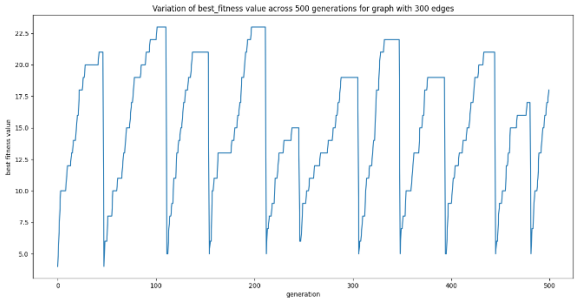
As can be seen, in this case, we frequently get the best fitness value as **31**, and it even peaks at **32**.

```
C:\Users\ARYAN GUPTA\Desktop\Codes\AI\Assignment1>2019A7PS0017G_ARYAN.py
Roll no.: 2019A7PS0017G
Number of edges: 200
Best state: [{0: 'b'}, {1: 'r'}, {2: 'b'}, {3: 'r'}, {4: 'r'}, {5: 'r'}, {6: 'g'}, {7: 'b'}, {8: 'b'}, {9: 'b'}, {10: 'b'}, {11: 'b'}, {12: 'b'}, {13: 'b'}, {14: 'g'}, {15: 'r'}
, {16: 'r'}, {17: 'g'}, {18: 'r'}, {19: 'r'}, {20: 'g'}, {21: 'r'}, {22: 'g'}, {23: 'g'}, {24: 'g'}, {25: 'g'}, {26: 'r'}, {27: 'b'}, {28: 'g'}, {29: 'b'}, {30: 'b'}, {31: 'b'},
 {32: 'g'}, {33: 'r'}, {34: 'b'}, {35: 'b'}, {36: 'b'}, {37: 'g'}, {38: 'g'}, {39: 'r'}, {40: 'b'}, {41: 'r'}, {42: 'r'}, {43: 'g'}, {44: 'r'}, {45: 'g'}, {46: 'b'}, {47: 'b'},
{48: 'b'}, {49: 'r'}]
Fitness value of best state: 32
Time taken: 22 secs
```

4. **300 Edges:** (Randomly generated graphs)
    i. Gens before restart: 20



Popln size: 1000
Max fitness: 25

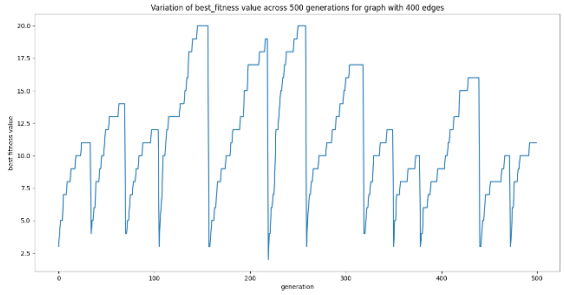


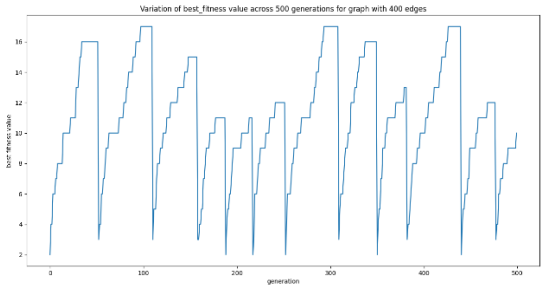
<u>Popln size: 500</u>
<u>Max fitness: 23</u>

For a population size of 1000, fitness value peaks at 25 (left image), but mode remains 23, the same as 500 population size (right image). So, choosing **500 popn size** is more consistent on average for all edge-set-sizes.

```
C:\Users\ARYAN GUPTA\Desktop\Codes\AI\Assignment1>2019A7PS0017G_ARYAN.py
Roll no.: 2019A7PS0017G
Number of edges: 300
Best state: [{0: 'r'}, {1: 'g'}, {2: 'r'}, {3: 'g'}, {4: 'r'}, {5: 'r'}, {6: 'b'}, {7: 'r'}, {8: 'g'}, {9: 'r'}, {10: 'r'}, {11: 'r'}, {12: 'b'}, {13: 'r'}, {14: 'r'}, {15: 'b'}
, {16: 'r'}, {17: 'g'}, {18: 'r'}, {19: 'b'}, {20: 'r'}, {21: 'r'}, {22: 'b'}, {23: 'g'}, {24: 'g'}, {25: 'r'}, {26: 'r'}, {27: 'r'}, {28: 'r'}, {29: 'g'}, {30: 'g'}, {31: 'r'},
 {32: 'r'}, {33: 'b'}, {34: 'b'}, {35: 'b'}, {36: 'b'}, {37: 'b'}, {38: 'r'}, {39: 'r'}, {40: 'g'}, {41: 'r'}, {42: 'r'}, {43: 'g'}, {44: 'r'}, {45: 'g'}, {46: 'b'}, {47: 'b'},
{48: 'r'}, {49: 'r'}]
Fitness value of best state: 23
Time taken: 26 secs
```

5. **400 Edges:** (Randomly generated graphs)
    i. Population size: 500, Gens before restart: 20





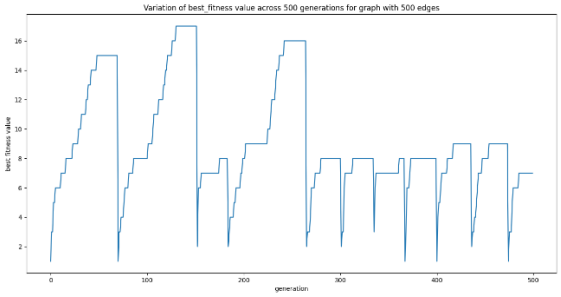
Fitness Value = 20

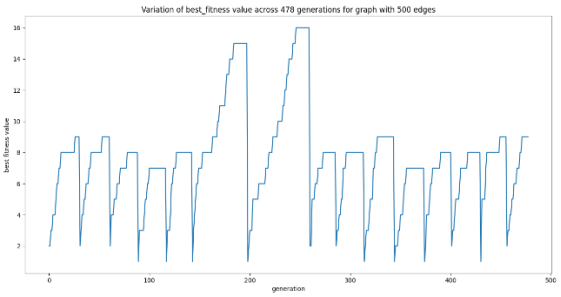
Fitness Value = 17

Even though, fitness values as high as 19 and 20 were obtained, **17 was the most frequent one**.

```
C:\Users\ARYAN GUPTA\Desktop\Codes\AI\Assignment1>2019A7PS0017G_ARYAN.py
Roll no.: 2019A7PS0017G
Number of edges: 400
Best state: [{0: 'r'}, {1: 'r'}, {2: 'r'}, {3: 'r'}, {4: 'r'}, {5: 'r'}, {6: 'r'}, {7: 'b'}, {8: 'b'}, {9: 'r'}, {10: 'b'}, {11: 'g'}, {12: 'r'}, {13: 'g'}, {14: 'r'}, {15: 'r'}
, {16: 'g'}, {17: 'r'}, {18: 'r'}, {19: 'r'}, {20: 'b'}, {21: 'r'}, {22: 'g'}, {23: 'r'}, {24: 'r'}, {25: 'b'}, {26: 'r'}, {27: 'r'}, {28: 'g'}, {29: 'r'}, {30: 'r'}, {31: 'r'},
 {32: 'r'}, {33: 'r'}, {34: 'r'}, {35: 'r'}, {36: 'r'}, {37: 'r'}, {38: 'g'}, {39: 'r'}, {40: 'b'}, {41: 'r'}, {42: 'b'}, {43: 'b'}, {44: 'r'}, {45: 'r'}, {46: 'r'}, {47: 'b'},
{48: 'b'}, {49: 'g'}]
Fitness value of best state: 17
Time taken: 32 secs
```

6. **500 Edges:** (Randomly generated graphs)
    i. Population size: 500





Fitness Value = 17

Gens before restart: 30

Fitness Value = 16

Gens before restart: 20

Even though, fitness values as high as 17 were obtained upon setting gens_before_restart to 30, there was no significant improvement since **16 was the most frequent one**, same as when the parameter was set to 20.

```
C:\Users\ARYAN GUPTA\Desktop\Codes\AI\Assignment1>2019A7PS0017G_ARYAN.py
Roll no.: 2019A7PS0017G
Number of edges: 500
Best state: [{0: 'r'}, {1: 'r'}, {2: 'r'}, {3: 'r'}, {4: 'r'}, {5: 'r'}, {6: 'r'}, {7: 'r'}, {8: 'r'}, {9: 'r'}, {10: 'r'}, {11: 'r'}, {12: 'r'}, {13: 'g'}, {14: 'r'}, {15: 'r'}
, {16: 'b'}, {17: 'g'}, {18: 'b'}, {19: 'g'}, {20: 'r'}, {21: 'r'}, {22: 'r'}, {23: 'r'}, {24: 'r'}, {25: 'r'}, {26: 'g'}, {27: 'b'}, {28: 'g'}, {29: 'b'}, {30: 'r'}, {31: 'r'},
 {32: 'r'}, {33: 'b'}, {34: 'r'}, {35: 'b'}, {36: 'r'}, {37: 'g'}, {38: 'r'}, {39: 'r'}, {40: 'b'}, {41: 'b'}, {42: 'g'}, {43: 'r'}, {44: 'r'}, {45: 'b'}, {46: 'r'}, {47: 'r'},
{48: 'r'}, {49: 'r'}]
Fitness value of best state: 16
Time taken: 45 secs
```

Conclusion:

- Population size of 500 is the sweet spot across all edge-set sizes
- Maximum number of generations: 500
- Generations to wait before restarting, 30 proves to be enough for all edge-set sizes
- Improvements by elitism, culling, mutation and random restarts have been significant!
  Even though individual graphs have not been shown, each improvement was carried out in a step-by-step manner as described on Page 2 of the report.