

Maximum Weighted Independent Set Problem

Independent set is set of nodes such that for each two node there is no edge connecting the two. Maximum Independent Set is an Independent Set where the number of nodes is the maximized. In Maximum Weighted Independent Set however we want to maximize total weight instead of number of nodes.

In this project we are required to solve Maximum Weighted Independent Set problem using three different optimization methods. These methods are Mixed Integer Program, Greedy Heuristic and Genetic Algorithm. But first we need to generate 20 graphs with given number of nodes and densities. For this I have used python and implemented the algorithm given in the assignment file. Graphs generated using this algorithm can be found in “Generated_Graphs” folder. Each graph file is named according to node and density parameters used to generate them.

Solving MWISP Using Gurobi MIP Optimizer

For Mixed Integer Program optimization, we used the Gurobi optimizer. Model of our Mixed Integer Program is given to us in the assignment file.

$$\max z = \sum_{v \in V} w_v x_v$$

subject to

$$\begin{aligned} x_u + x_v &\leq 1 \quad \forall (u, v) \in E \\ x_v &\in \{0, 1\} \quad \forall v \in V \end{aligned}$$

where x_v is the decision variable indicating whether the vertex v is included in the independent set or not, w_v is the weight of the vertex v and E is the all edges in the graph.

We set the time limit for each solution to 1 hour. Most of the time our program couldn't find the optimal solution within this time limit.

Nodes	Edges	Best Objective	Gap	
500	12439	34.777866	48.3166%	Time limit reached
500	37325	15.060515	66.2465%	Time limit reached
500	62237	9.5906060	69.0922%	Time limit reached
500	87716	6.1744160	0.0000%	Optimal solution found
500	112333	3.9407050	0.0000%	Optimal solution found
1000	50264	40.303603	157.7795%	Time limit reached
1000	149563	16.138836	290.9176%	Time limit reached
1000	248959	10.767728	282.2196%	Time limit reached
1000	349827	6.5127380	281.4924%	Time limit reached
1000	449884	4.7080190	0.0000%	Optimal solution found
1500	112675	46.335304	243.7139%	Time limit reached

1500	337286	17.464492	417.4484%	Time limit reached
1500	563266	11.084509	438.2662%	Time limit reached
1500	788176	7.4546860	461.3094%	Time limit reached
1500	1011457	4.3145110	286.5451%	Time limit reached
2000	199416	48.914668	303.0231%	Time limit reached
2000	599229	18.767179	522.0479%	Time limit reached
2000	1000200	11.137766	69.0922%	Time limit reached
2000	1399672	7.4575820	595.7892%	Time limit reached
2000	1799124	4.9601610	413.6877%	Time limit reached

More detailed reports for each solution can be found in the folder “MIP_Solutions”.

Solving MWISP Using Greedy Heuristic

For greedy heuristic optimization, I have implemented a simple greedy heuristic algorithm. I have sorted the weight's list in decending order. Picked the first one from the list and added it to the solution. Then I picked the next weight with biggest value and added it to the solution string if adding this weight affected the feasibility of the solution, I removed it from the solution and go to next biggest weight. I repeated this for all weight's in the list.

```

Greedy(G,W):
  S ← {0,0,0 ... ,0}
  W=sort_decending(W)
  for all w in W do
    set node with weight w, 1
    if solution is not feasible
      remove node with weight w
  end for
  Output S

```

Nodes	Edges	Objective	Completed In
500	12439	28.034406	13s
500	37325	11.162040	12s
500	62237	8.3497349	11s
500	87716	5.1425030	11s
500	112333	2.7155970	13s
1000	50264	34.412266	65s
1000	149563	14.386226	119s
1000	248959	7.4494070	116s
1000	349827	5.6736509	66s

1000	449884	3.2820900	119s
1500	112675	42.022271	204s
1500	337286	16.484963	220s
1500	563266	8.8922360	184s
1500	788176	5.4002690	438s
1500	1011457	4.0362390	404s
2000	199416	43.565015	474s
2000	599229	17.320467	650s
2000	1000200	9.3660869	660s
2000	1399672	5.5208180	479s
2000	1799124	3.5468290	646s

While this greedy heuristic solution didn't provide us better solutions than the MIP solver, it can still be useful due to its extremely small running time compared to MIP solver.

Solving MWISP Using Genetic Algorithm

In the Genetic Algorithm, as requested in the assignment, I have used binary tournament selection, uniform crossover with probability of 0.5, bitwise mutation with probability of $1/n$ (n : number of nodes), and implemented my own "generate initial population" and "repair" functions.

```

Genetic(G,W):
  P ← Generate_Initial_Population(G,W) // P: population
  for 0 to number of generations do
    PP ← Binary_Tournament_Selection(P,G,W) // PP: Parent population
    PO ← Uniform_Crossover(PP,G,W) // PO: Offsprings
    PM ← MUTATE(PO,G,W) //PM: Mutated population
    P ← Repair(PM,G,W)
  end for

```

We are also required to run genetic algorithm with different population sizes (50,100) and generation numbers (50,100,150) for each graph. In total I have run 120 different scenarios using genetic algorithm.

When generating the initial population I used the following procedure. For each initial solution I picked one node with biggest weight that is not already picked by any other initial solution and set that node in the solution to 1 and rest to 0. This way I have generated p (p : population size) solutions, each containing one of the p largest weighted nodes.

For repair function I randomly removed nodes from the unfeasible solution until it becomes feasible, then for each removed node, I added a new node with biggest value that doesn't break feasibility. I am not exactly sure about this repair function, it produces decent solutions that doesn't reduce the standard deviation but can be improved further.

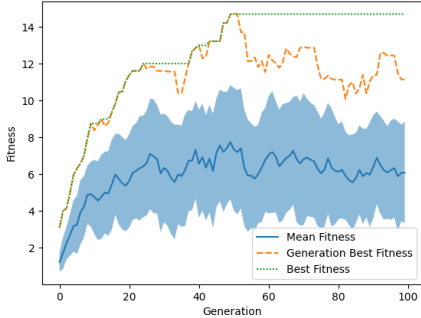
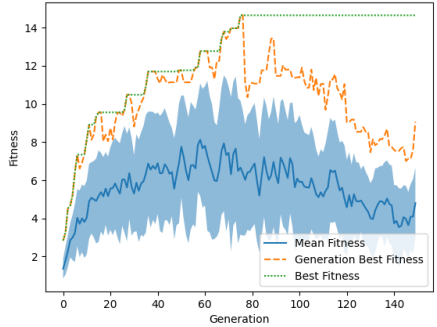
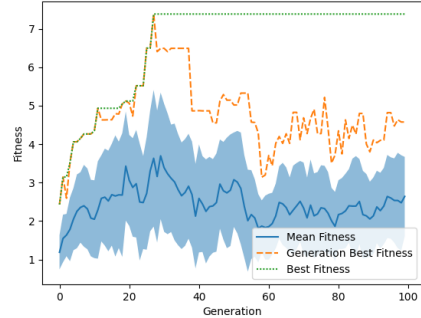
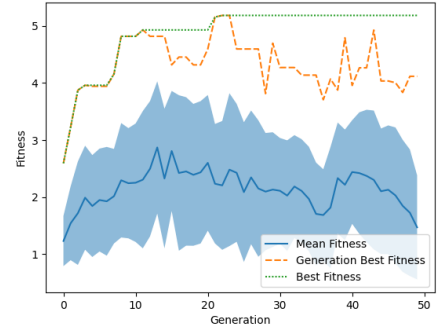
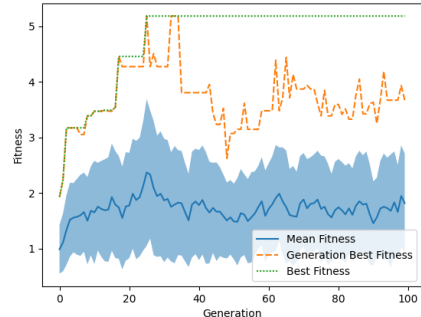
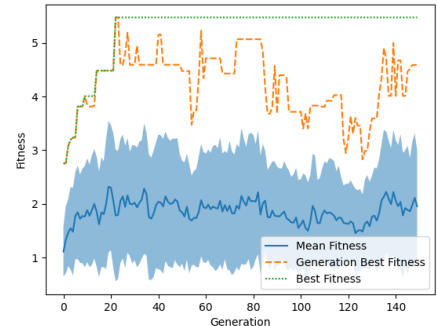
Generating Initial Population	Repair Function
Generate_Initial_Population(G,W): P ← initial population with all nodes, 0 N ← indexes of largest n nodes for i = 0 to n do P[N[i]] = 1 end for Output initial population, P	Repair(S,G,W): I ← indexes of all nodes with value of 1 in the solution c ← 0 // removed node counter while S is not feasible do Randomly pick a node from I and remove it from S c++ end while for 0 to c do Greedy pick the node with biggest possible W Add this node to the S end for Output feasible S

Table of best fitnesses from each scenario

Nodes	Edges	Fitness (Best fitnesses are marked with yellow)					
		Population # 50			Population # 100		
		Generatio n # 50	Generatio n # 100	Generatio n # 150	Generatio n # 50	Generation # 100	Generation # 150
500	12439	12.020945	12.507060	12.335749	11.278055	14.712618	12.111327
500	37325	6.9109159	7.3847770	6.7440200	6.7312160	6.5528879	7.2534710
500	62237	4.3815800	4.3350570	5.0539460	4.8055819	5.1846680	4.6614230
500	87716	2.9656190	4.3878260	3.6826410	3.5981320	4.1727060	3.5816990
500	112333	2.4223120	2.7716680	2.6935660	2.7213050	2.6961820	2.7672990
1000	50264	12.677905	11.369600	12.552221	12.394047	15.254859	14.359198
1000	149563	5.9029600	5.8275590	7.0129570	6.9528300	7.0267950	6.5351609
1000	248959	3.7385210	4.9125700	5.8207749	4.4466390	5.0175050	4.4766780
1000	349827	3.0535670	2.9094680	3.5778330	3.6682170	3.4691970	3.6809360
1000	449884	2.7768840	2.4341550	2.7950790	2.6417700	2.8156340	3.1283630
1500	112675	12.911926	13.589002	14.646395	11.994280	14.226126	12.653166
1500	337286	17.208511	6.5728670	6.4956070	5.9901280	7.4538379	7.8168950
1500	563266	4.4929600	4.5663420	4.3336260	5.3713850	4.6329600	5.4728270
1500	788176	3.2652840	3.3788489	3.5404330	3.3496230	3.4623100	3.7505270
1500	1011457	2.0098790	2.9481950	2.6415740	2.9282830	3.2749970	2.8590940
2000	199416	14.787908	11.623753	12.395629	12.488440	15.297935	13.442325
2000	599229	6.3985940	6.0879470	6.6417560	6.1930580	7.1796630	6.9281049

2000	1000200	5.1923179	4.7417880	4.3703330	4.6978800	5.0664550	5.6932459
2000	1399672	3.5150930	3.2935570	3.4636240	3.6749490	3.4753090	3.8965420
2000	1799124	2.5614390	2.5392960	2.5129420	2.7085130	2.5697530	2.9236810

Table of generation-fitness plots for best solution of each graph. Green line represents best fitness up until that generation, orange line represents best fitness in the generation, blue line represents mean fitness the generation and blue area around mean represents the standard deviation. Detailed graphs and reports for each scenario can be found in “Genetic_Solutions” folder.

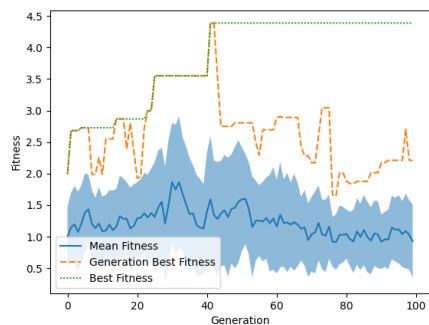
Generation – Fitness plots for best solutions of each graph			
Nodes: 500 Edges: 12439 Best: 14.712618 Pop: 100		Nodes: 1500 Edges: 112675 Best: 14.646395 Pop: 50	
Nodes: 500 Edges: 37325 Best: 7.3847770 Pop: 50		Nodes: 1500 Edges: 337286 Best: 17.208511 Pop: 50	
Nodes: 500 Edges: 62237 Best: 5.1846680 Pop: 100		Nodes: 1500 Edges: 563266 Best: 5.4728270 Pop: 100	

Nodes:
500

Edges:
87716

Best:
4.3878260

Pop: 50

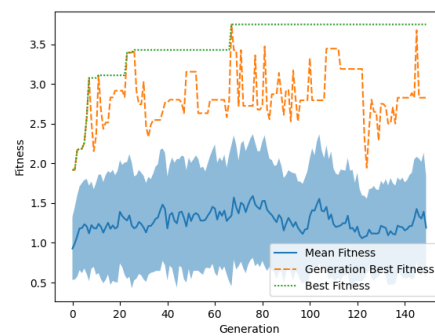


Nodes:
1500

Edges:
788176

Best:
3.7505270

Pop: 100

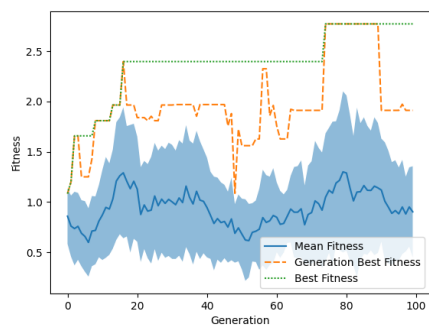


Nodes:
500

Edges:
112333

Best:
2.7716680

Pop: 50

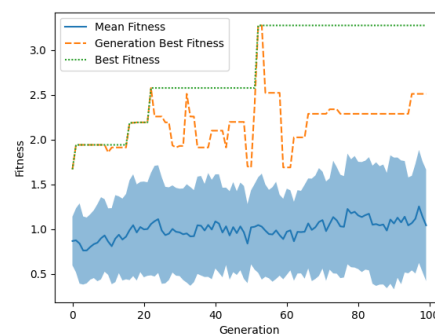


Nodes:
1500

Edges:
1011457

Best:
3.2749970

Pop: 100

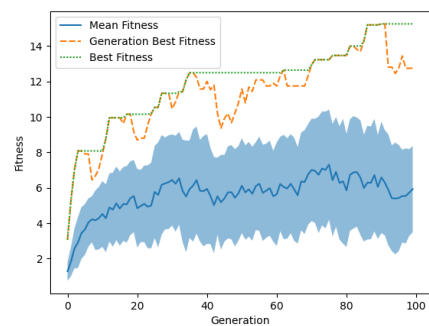


Nodes:
1000

Edges:
50264

Best:
15.254859

Pop: 100

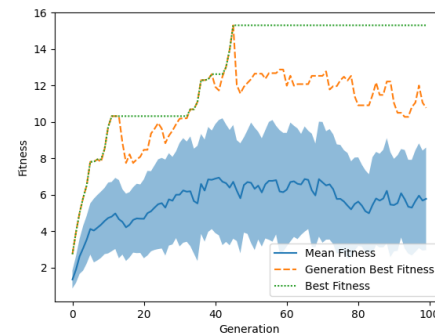


Nodes:
2000

Edges:
199416

Best:
15.297935

Pop: 100

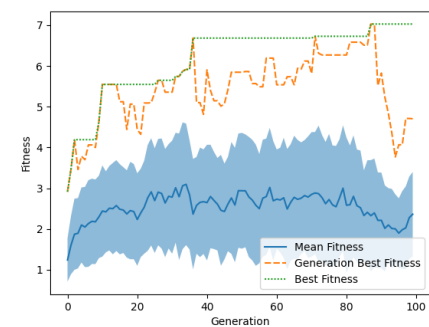


Nodes:
1000

Edges:
149563

Best:
7.0267950

Pop: 100

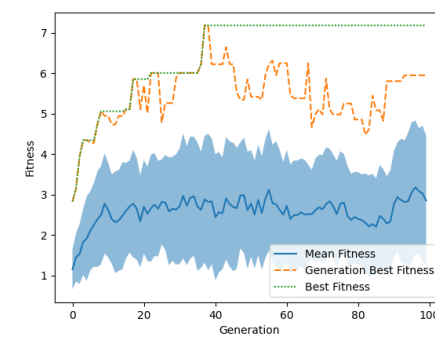


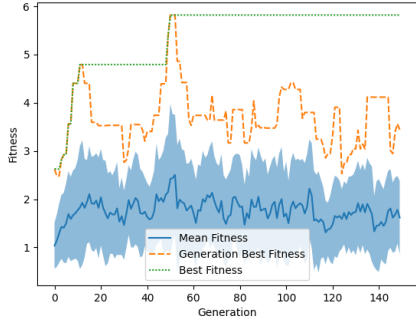
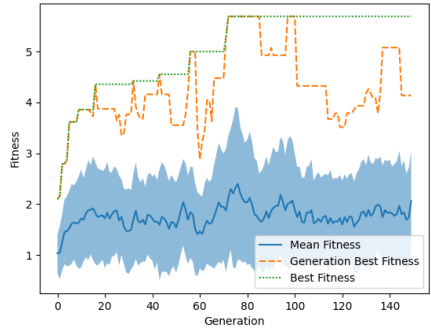
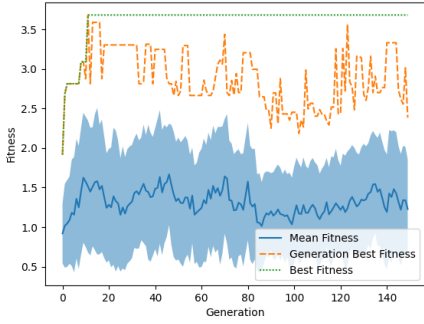
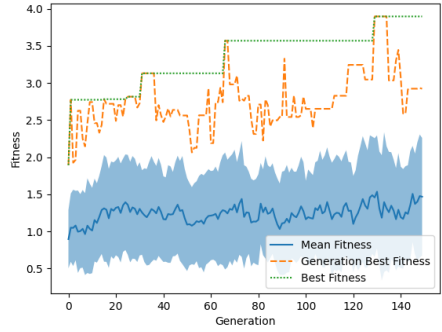
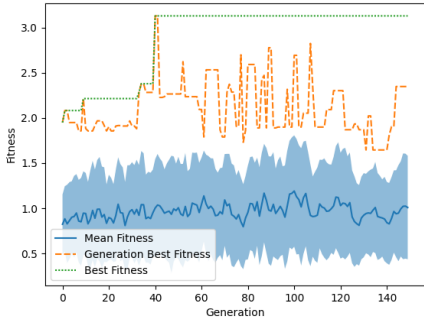
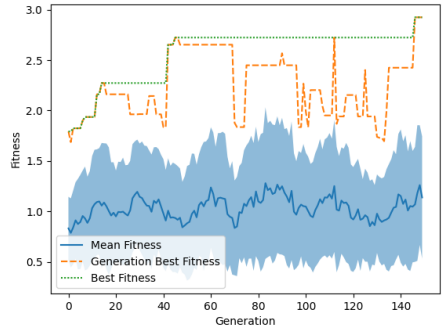
Nodes:
2000

Edges:
599229

Best:
7.1796630

Pop: 100



Nodes: 1000 Edges: 248959 Best: 5.8207749 Pop: 50		Nodes: 2000 Edges: 1000200 Best: 5.6932459 Pop: 100	
Nodes: 1000 Edges: 349827 Best: 3.6809360 Pop: 100		Nodes: 2000 Edges: 1399672 Best: 3.8965420 Pop: 100	
Nodes: 1000 Edges: 449884 Best: 3.1283630 Pop: 100		Nodes: 2000 Edges: 1799124 Best: 2.9236810 Pop: 100	

Conclusion

Despite not being able to complete within given time limit, Mixed Integer Program solver performed the best for each graph. Meanwhile Greedy Algorithm usually performed better than genetic algorithm and achieved objective values that are relatively close to the MIP solution. Genetic algorithm performed worse than I initially expected, but I believe these fitness (objective) values can be increased by increasing population size and number of generations.

Below there is a table compares best objectives of three optimizers. Yellow highlight represents optimizer with best solution and orange highlight represents the second best.

Nodes	Edges	MIP Best Objective	Greedy Best Objective	Genetic Best Objective
500	12439	34.777866	28.034406	14.712618
500	37325	15.060515	11.162040	7.3847770
500	62237	9.5906060	8.3497349	5.1846680
500	87716	6.1744160	5.1425030	4.3878260
500	112333	3.9407050	2.7155970	2.7716680
1000	50264	40.303603	34.412266	15.254859
1000	149563	16.138836	14.386226	7.0267950
1000	248959	10.767728	7.4494070	5.8207749
1000	349827	6.5127380	5.6736509	3.6809360
1000	449884	4.7080190	3.2820900	3.1283630
1500	112675	46.335304	42.022271	14.646395
1500	337286	17.464492	16.484963	17.208511
1500	563266	11.084509	8.8922360	5.4728270
1500	788176	7.4546860	5.4002690	3.7505270
1500	1011457	4.3145110	4.0362390	3.2749970
2000	199416	48.914668	43.565015	15.297935
2000	599229	18.767179	17.320467	7.1796630
2000	1000200	11.137766	9.3660869	5.6932459
2000	1399672	7.4575820	5.5208180	3.8965420
2000	1799124	4.9601610	3.5468290	2.9236810

Code, used to run all these solutions can be found in “Optimization-Term-Project” folder.