**Sabancı University**

Faculty of Engineering and Natural Sciences

Fall 2020

**CS301 Algorithms**

Group 18 Project Report

Group Members:

Abdullah Cevizli

Eyüp Burak Keskintaş

Emre Muslu

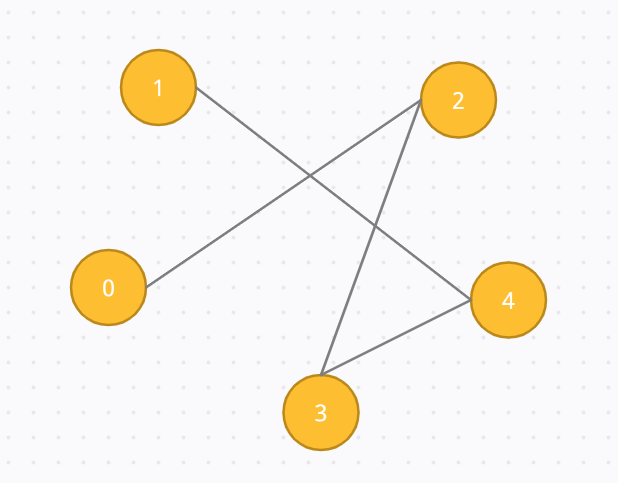
Doruk Ertoğlu

Instructor: Hüsnü Yenigün

**Maximum Independent Set Problem**

1. **The Problem**
2. **Definition of the Problem**

Given a simple undirected graph G = (V, E), independent set S of graph G is set of vertices such that no pair of vertices in set S is connected by an edge ∈ E. That is, set S only contains non-adjacent vertices. Maximum independent set MIS of graph G is independent set that have largest possible cardinality. Namely, there is no vertex v in G /MIS such that v ∪ MIS is also maximum independent set. There can be more than one possible unique maximum independent set of graph G. An example of Maximum independent set:



Given a graph G = (V, E), V = {1,2,3,4,5}, E = {(1,4), (0,2), (2,3), (1,4), (4,3)}, Maximum independent set MIS = {1,3,0}.

Maximum independent set has many applications in real life. For instance, Ağralı *et al.* demonstrates that scheduling employees in service industry with employee requirements is a maximum independent set problem [1]. Given an employee, requirement for this employee is represented as a node in undirected graph. So, it is observed that any assignment of a employee to a requirement requires independent set. Thus, optimal solution for this requires finding maximum independent set.

1. **Proving the hardness of the Problem**

**Independent Set is NP Complete**

Proof:

1. Independent Set is NP
2. Independent Set is NP-Hard

A. Independent Set is NP

If a problem is NP, then given a solution to the problem and an instance of the problem, a graph G and a positive integer k in our case. We will be able to check the solution is correct or not in polynomial time.

The solution is a subset V’ of the vertices, which includes the vertices having a place with the independent set. We can approve this solution by checking that each pair of vertices have a place in the solution are non-neighboring, by basically confirming that they share an edge to one another. This should be possible in polynomial time, that is O(V+E) utilizing the graph G(V+E):

**check=true**

**foreach pair {u, v} in subset V’:**

**if edge between them:**

**check=false**

**break**

**if check = true:**

**Correct solution**

**else:**

**incorrect solution**

B. Independent Set is NP-Hard

To demonstrate that Independent Set problem is NP-Hard, we will reduce from a known NP-Hard problem to this Independent Set problem. We will do a reduction from which the Clique Problem can be reduced to the Independent Set problem.

Each instance of the Clique problem depends on the graph G(V, E) and an integer k can be changed over to the necessary graph G’(V’, E’) and k’ of the Independent Set problem. V’ is all the vertices of G and E’ is the complement of edges E. Time required to compute the graph G’ is O(V+E) because we need to traverse over all the vertices and edges.

Now we will prove that computing Independent Set in reality reduces to the computation of the Clique. Two propositions are given in for reduction:

1. Allow us to expect there is a clique of size k in graph G. The presence of clique suggests that there are k vertices in G, where every one of the vertices is associated by an edge with the residual vertices. This highlights that since these edges are in G, hence they can’t be in G’. Thus, these k vertices do not neighbor each other in G’ and consequently form an Independent Set of size k.
2. We expect that the graph G’ has an independent set of vertices of size k’ in addition, none of these vertices shares an edge with others. When complement G’ in order to get graph G, these k vertices will share edges thus, be neighbor to other. Hence, the graph G will have a clique of size k.

Lastly, if there is a clique of size k in graph G we can say that there is an independent set of size k in graph G’. Being both NP and NP-Hard makes the Independent Set problem NP-Complete [3].

G Clique size 3 -> A, C, D

G’ Independent Set size 3 -> A, C, D

1. **The algorithm**

**Algorithm Description**

As we know there is no polynomial algorithm for any hard problem. For this, approximate and heuristic algorithms are developing indeed. In this project we used an algorithm to find the maximum independent set in an undirected graph presented by Gainanov *et al* [2].

Definition 1:

Given a graph G = (V, E), a vertex v ∈ V is a k-vertex, if neighborhood of this vertex has size of k for an integer k and this neighborhood is a complete subgraph.

Definition 2:

Given a graph G = (V, E), a vertex v ∈ V is (k, m)-vertex if neighborhood of this vertex has size of k for an integer k and this neighborhood requires m edges to become complete subgraph.

Algorithm Features:

1. Input of the algorithm is the graph G = (V, E)-> Adjacency matrix
2. Each iteration changes the matrix, when a vertex is in S and removed from with adjacent vertices.
3. Inside the arrays k and m of length n, for related vertex values of parameters k and m stored (updated in iterations).
4. The vertex with minimum value of m and maximum value of k is returned by MinMaxParam() ( CheckKvertex() in implentation)
5. Vertex with minimum m and maximum k is selected into set.

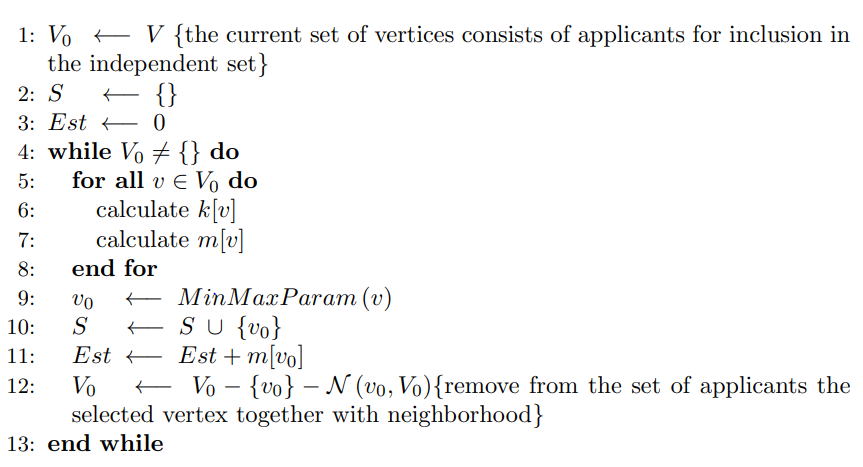
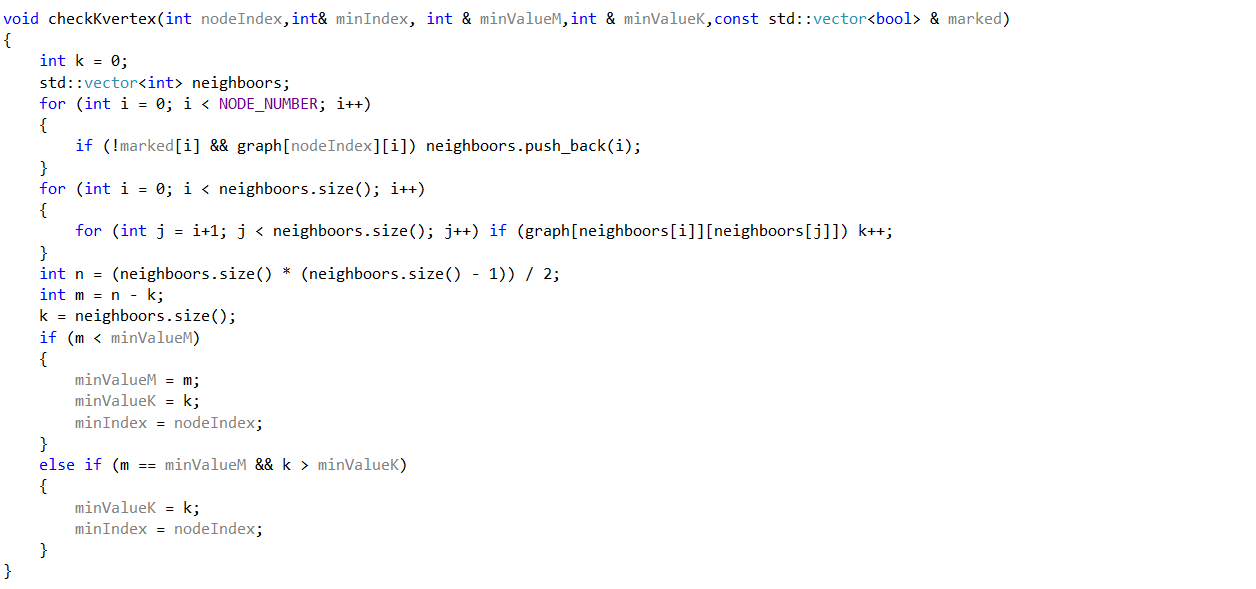


Figure 1: pseudo code for Algorithm [2]

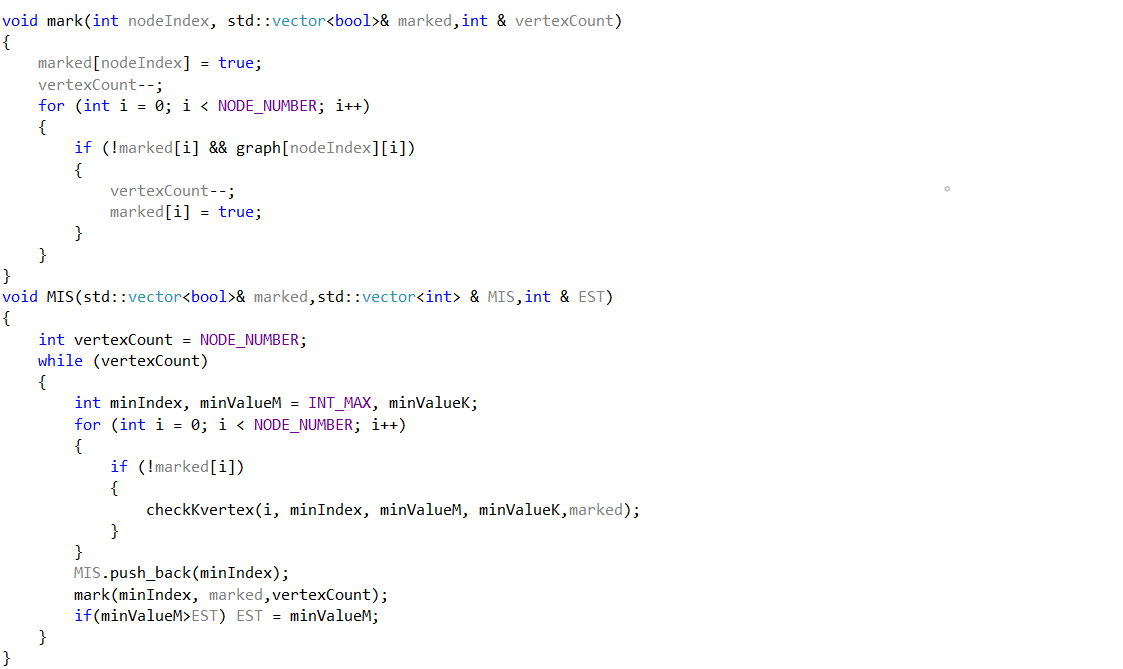
**Algorithm Analysis**

Worst case asymptotic running time of this algorithm:



O(k1²+k2²+…+ks²)

O(V)



O(V)

O(V³/S)

O(V³)

O(V)

Number of vertices = V, Size of the output set = S

CheckKvertex: By amortized analysis, if we call the neighborhood size is “k”, we assign amortized cost as and CheckKvertex() has a worst case time complexity of O(V) + O(k1²+ k2²+ …+ ks²).

Here we can say O(²+ ²+ …+ ) = O((+ + ...+ ) × ) where denotes mean of k values. Also, O(+ + ...+ ) = O(V) since all neighborhoods of added vertices cannot exceed V. Thus, = O(V / S) and O((+ + ...+ ) × ) = O(V² / S). Result is O(V) + O(V² / S). Since S can’t be bigger than “V”, (V² / S) dominates “V”. Therefore CheckKvertex = O(V² / S).

MIS: CheckKvertex is O(V² / S), it is iterated “V” times by the for loop. Inside the while loop we have O(V³/S) and Mark function which is O(V). While loop is iterated “S” times therefore we have O(S\*((V³/S) + V)). (V³/S) dominates “V” so the worst-case time complexity of MIS algorithm is

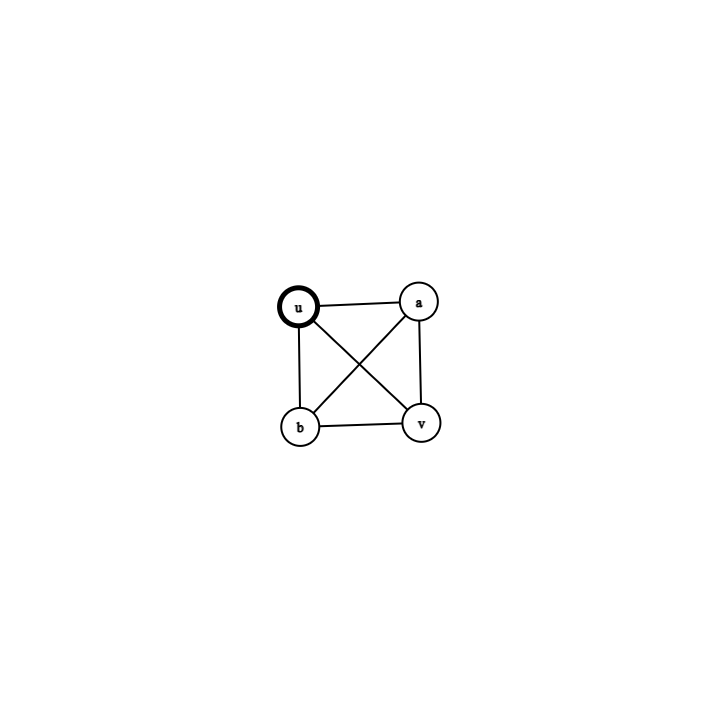
O(S\*( V³/S)) = O(V³).

**Proving correctness of the algorithm**

2.1 – Base theorem of heuristic algorithm

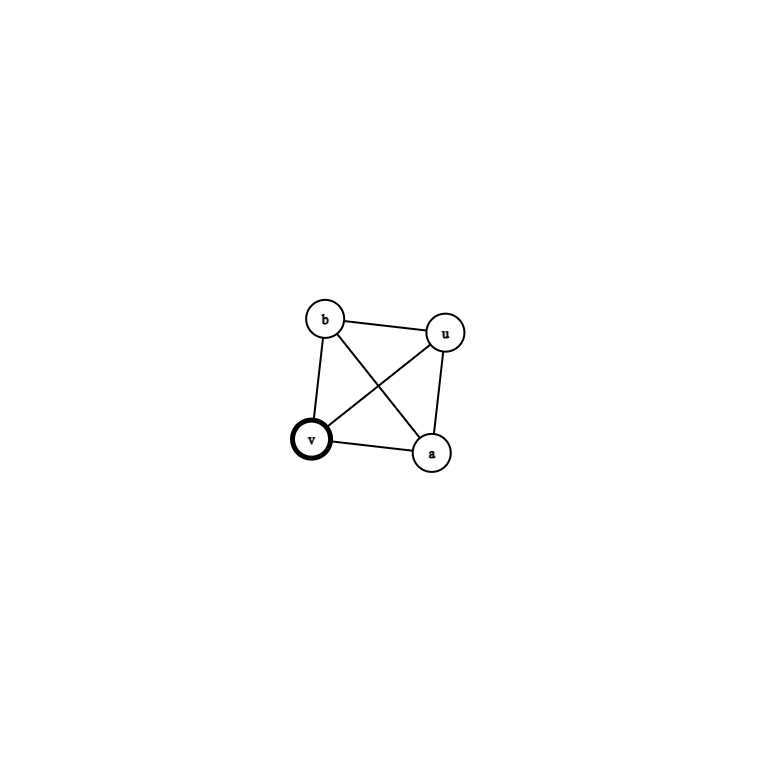
Our algorithm is heuristic algorithm. Hence, it is not giving optimal solution for all cases. However, it always provides independent set.

Base Theorem of heuristic algorithm:



Given a graph G = (V, E), let say v ∈ V is a k-vertex. Then there is exist a maximum independent set S of graph G.

Proof of theorem: Given a maximum independent set S and k - vertex v ∈ V and set of all neighbors of this vertex v, this maximum independent set S must include this vertex v or one of the neighbors of v. Indeed, let’s assume S does not include then there is a set S’ with S U (u) where u denotes vertex v or one of the neighbors of v. Thus, this set has more cardinality then set S which contradicts S is a maximum independent set[2].

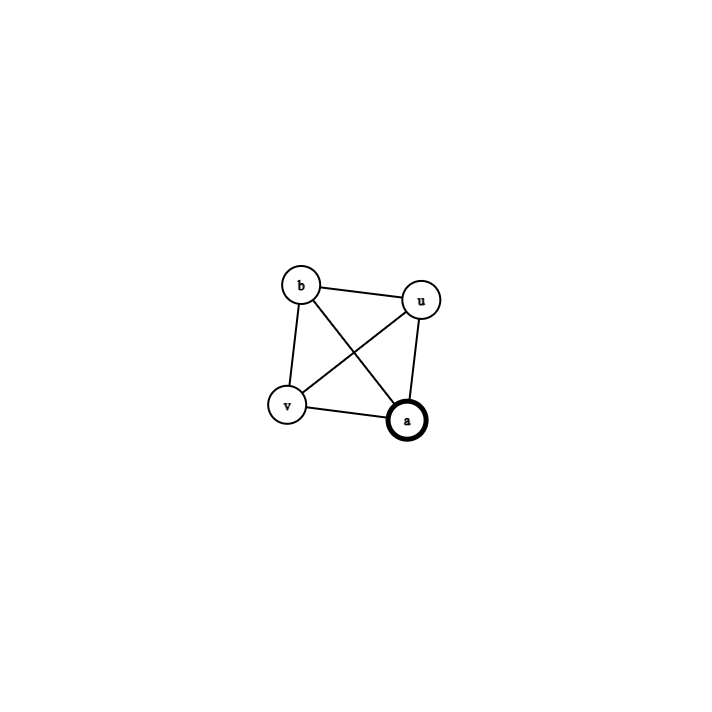
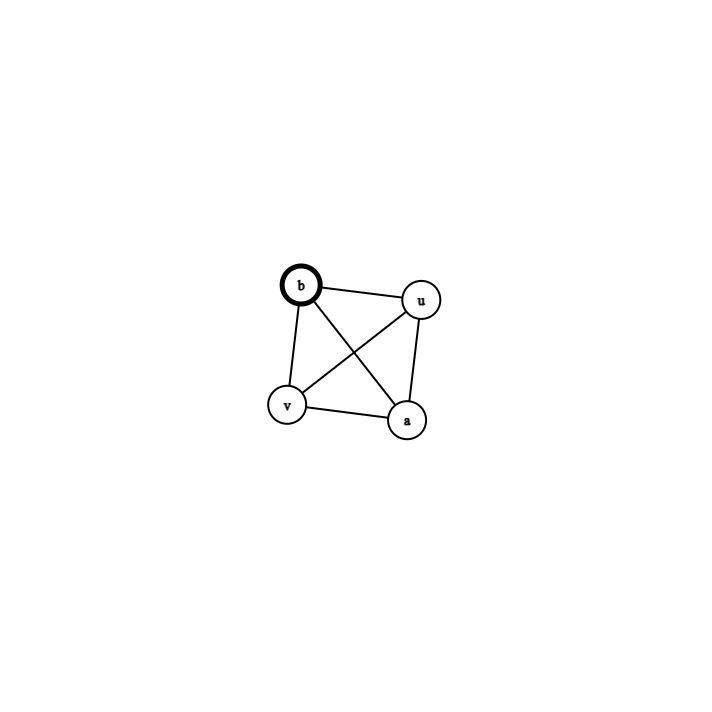
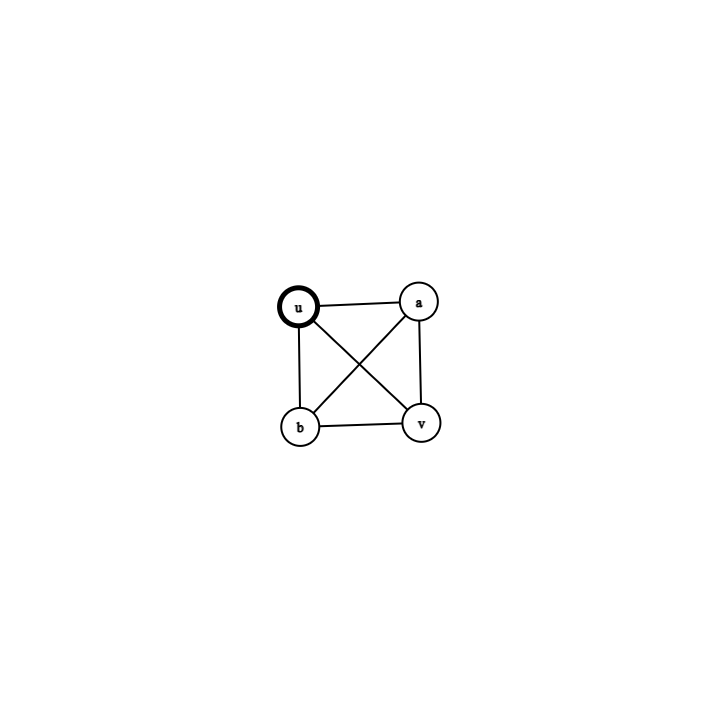


Two Possible cases of that situation

1- If v is in the S which is set of maximum independent set.

Then, there exists maximum independent set.

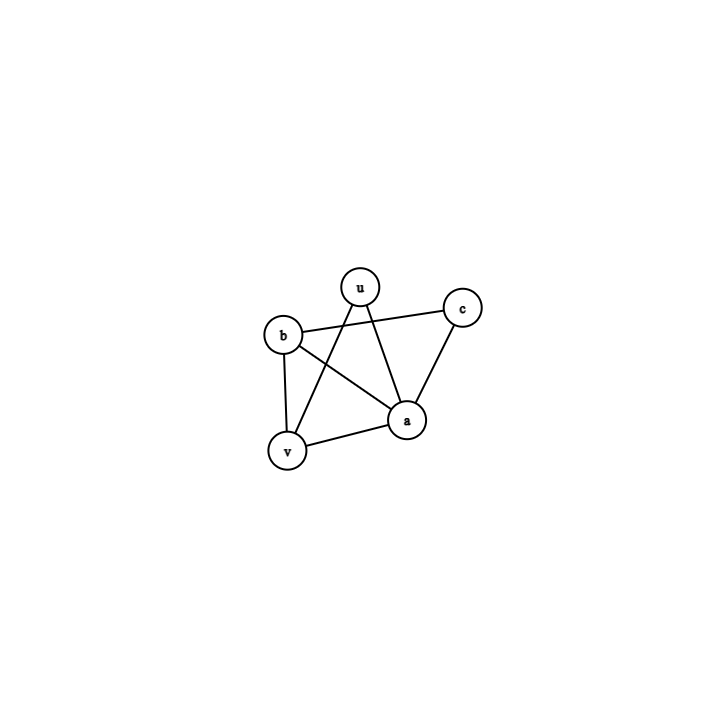
If v is not in S but u is in S which is node neighborhood of v.



That u can u, b, a in that case. So, we can find S’ which is consist of S – {u} (selected neighborhood node) U {v}. Hence, v is in S’ and cardinality of S’ is not bigger than cardinality of S. They have equal size so S’ is also maximum independent set if S is maximum independent set. Therefore, we can say that v is in S[2].

2.2 Including a vertex according to m values.

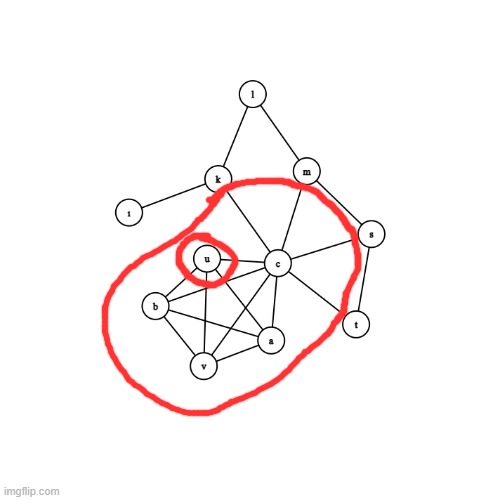
In every common iteration of algorithm, it includes into set a vertex with minimum m value among uncovered vertices. As proven in base algorithm whenever there is a vertex with zero m value a maximum independent set must include it. Thus, this way algorithm maximizes set size.



In this graph vertex a is (4,3) – vertex, v is (3,1) vertex, b is (3,1)- vertex, u is (2,0) vertex, c is (2, 0) vertex, since inclusion of vertex with minimum m value maximizes, in this case algorithm will include vertex u.

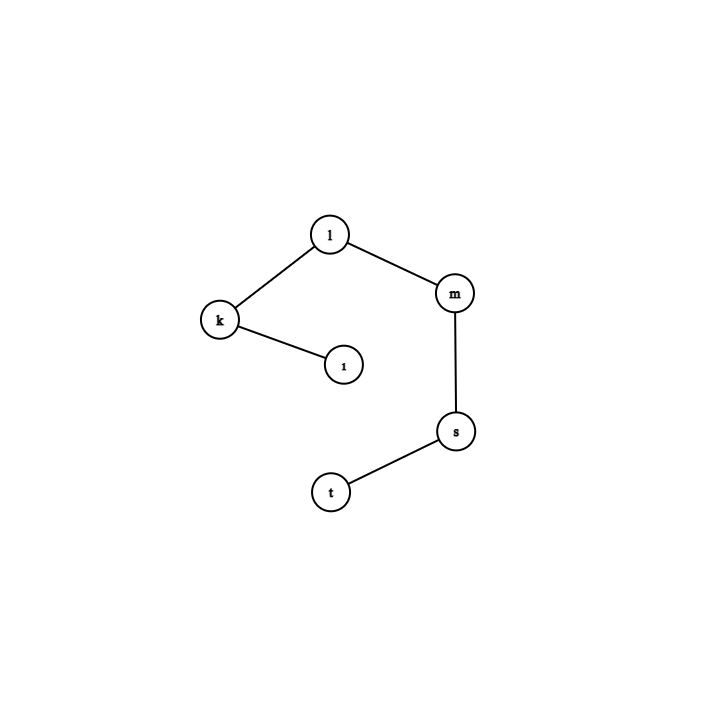
2.3 Prevent collisions

When algorithm includes vertex u to maximum independent set S, it will cover u and its neighborhood. For example, if node u is chosen it will not be considered in next iteration since in every iteration algorithm iterates over uncovered vertices. All remain uncovered vertices are not connected with vertex u.



When node u selected shown in left picture.

We will remove it and its neighborhood.



When chosen node and its neighborhood are not remain list. So, we prevented collisions because there is not chance choose node from S which is maximum independent set.

2.4 Est and optimal solution

Whenever a vertex included into set, it compares m value of this vertex with current maximum included m value .it calculates EST value which is shows max m value of the vertices in the set. As explained in proposition one, vertex v has zero m value, then a maximum independent set includes it. If EST is zero, that means algorithm creates optimal solution since all vertices in the set has m value zero. As a heuristic algorithm it is expected EST value is not always zero.

**Ratio Bound**

Following propositions and proofs are work of the Gainanov *et al* [2].

Given a graph G = (V, E) let S(G) be all independent sets of G and let (G) be all maximal independent sets of G and let denote size of maximal independent set S ∈ (G) maxS(G).

Proposition 1.

Given two graph G1 = (V, E1) and G2 = (V, E2) where E1 ⊆ E2, (G2) ⊆ S(G1).

Proof: let S ∈ S(G2) for each pair of vertices u, v ∈ V, if u, v ∈ S then (u, v) is not element of E2 by definition of independent set. Since E1 ⊆ E2 (u, v) is not element of E1 as well. Thus S ∈ S(G1) and S(G2) ⊆ S(G1). Also, (G2) ⊆ S(G2) thus proposition holds.

Proposition 2.

Given two graph G1 = (V, E1) and G2 = (V, E2) where E1 ⊆ E2, maxS(G1) ≥ maxS(G2).

Proof: for every S ∈ (G2), S ∈ S(G1) by proposition 1 and there exist S’ ∈ (G1) such that |S’| ≥ |S|. thus maxS(G1) = |S’| | ≥ |S| = maxS(G2).

Proposition 3.

Given two graph G1 = (V, E1) where u, v ∈ V, (u, v) is not element of E1, G2 = (V, E2) where E2 = E1 ⊔ {(u, v)}. maxS(G1) ≥ maxS(G2) ≥ maxS(G1) – 1

Proof: first inequality holds by proposition 2. Let S ∈ (G1). If u, v is not element of S or u ∈ S v is not, or v ∈ S u is not, then S ∈ (G2) thus maxS(G2) = |S| ≥ maxS(G1) – 1.

If u, v ∈ S then there exist S’ such that S’ = S – {v} or S’ = S – {u}.

thus |S’| = S – 1 ≥ maxS(G1) – 1.

Proposition 4.

Given graph G = (V, E) Let v ∈ G be (k, m) vertex and v ∈ S where S ∈ S(G),

|S| ≥ maxS(G) -m

Proof: Let {, . . . , } be missing edges in neighborhood of v and let G’ = (V,E’) where

E’ = E ⊔ {, . . . , } then by applying proposition 3 m times we get |S| ≥ maxS(G) -m.

Lemma 1.

Let S\* be optimal maximum independent set and let S be set that algorithm creates.

|S\*| - |S| ≤ M where M = max{ | is the m value of vertex ∈ S, is a (k, m) vertex}

Proof of lemma 1: given graph G = (V, E), |S\*| = maxS(G) and let S includes a (k, m) vertex v ∈ V. By proposition 4, |S\*| - |S| ≤ m. By considering vertex in S with maximum value m, we guarantee that inequality |S\*| - |S| ≤ m always holds. Thus, we can define ratio bound p(n) as following

|S\*| / |S| - |S| / |S| ≤ M / |S|

|S\*| / |S| - 1 ≤ M / |S|

|S\*| / |S| ≤ M/|S| + 1

Let p(n) = M / |S| + 1 where n denotes m value of vertices of set S.

|S\*| / |S| ≤ p(n)

1. **Implementation of the Algorithm**

**Experimental Analysis of the Algorithm**

Standard deviation, standard error, sample mean and confidence level intervals are calculated for analyze the algorithm.

1. **Standard Deviation**

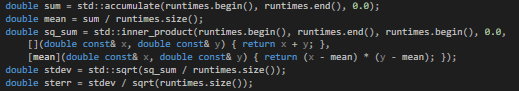
Standard deviation calculated with given formula

**σ**= Population standard deviation

**N=** Run times

**Σ**= sum of elements

**μ**= population mean



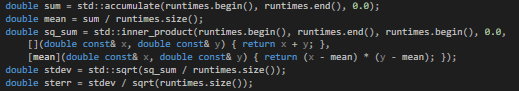
1. Standard Error



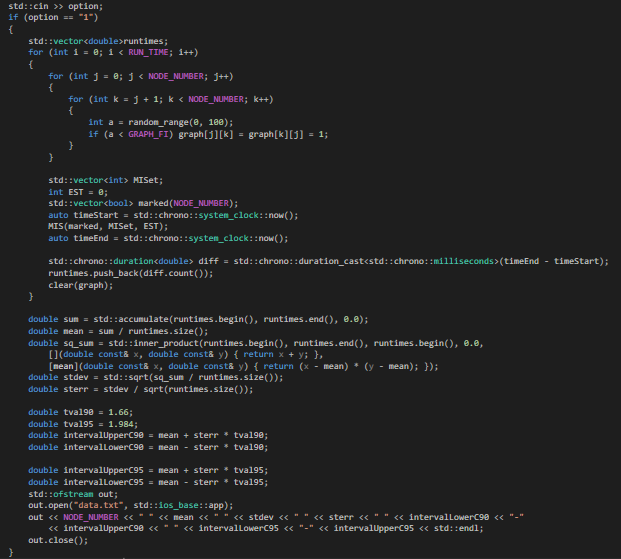
Standard Error calculated with given formula

**σ**= Population standard deviation

**N=** Run times



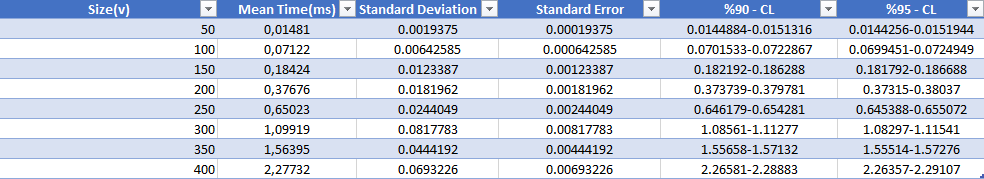
3) Running Time Algorithm



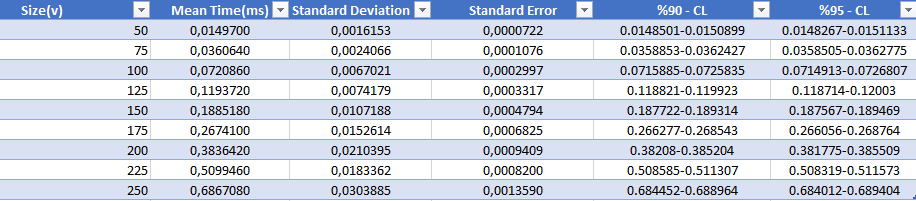
Experimental analyzes handled under predetermined conditions such as t-values. There are two different Confidence level with specific t values. Confidence level is %90 and t value for 1.66. Another confidence level is %95 and t-value for 1.984. Two different experiment applied on Algorithm using standard deviation, standard error, t-values, and confidence levels. One of the experiments is understanding relation between size and mean time when vertex number increasing, and the edge density is fixed. Second one is understanding relation between size and mean time when edge density of graph changing, and vertex number is fixed. Algorithm worst complexity is O(V^3). Also, algorithm depends on edge density rate because when vertex number is same and edge number increased, it changes m values which is important point of algorithm.

Keep Number of edge density (ED = %50)

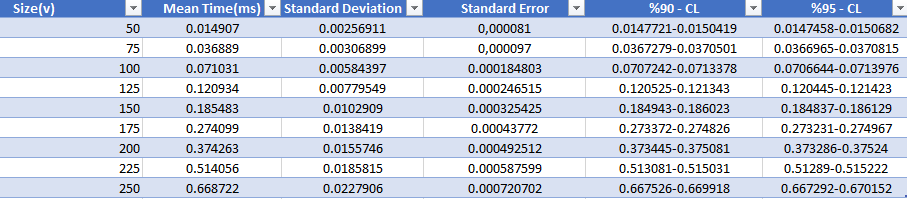
100 Number of Iterations per each Input Size



500 Number of Iterations per each Input Size



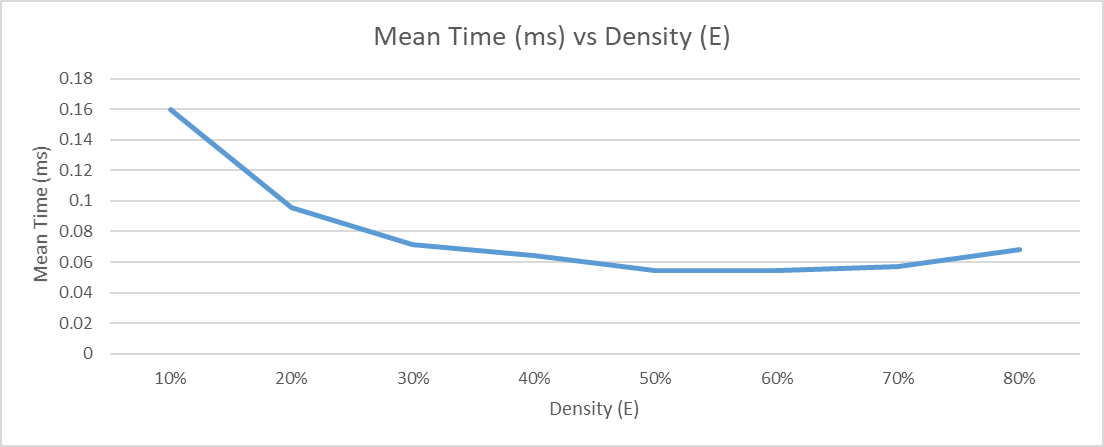
1000 Number of Iterations per each Input Size



**Keep Number of Vertices Constant (V = 100)**

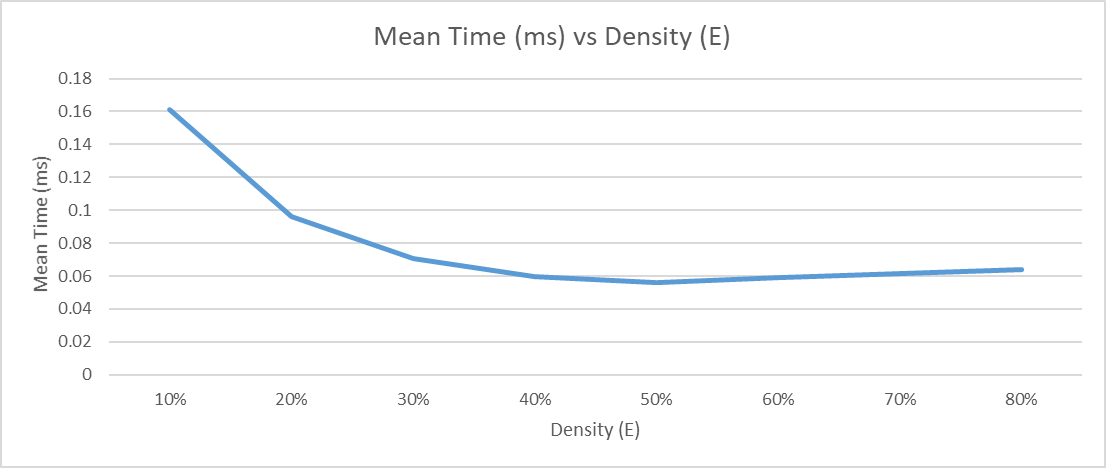
100 Number of Iterations per each Density



****

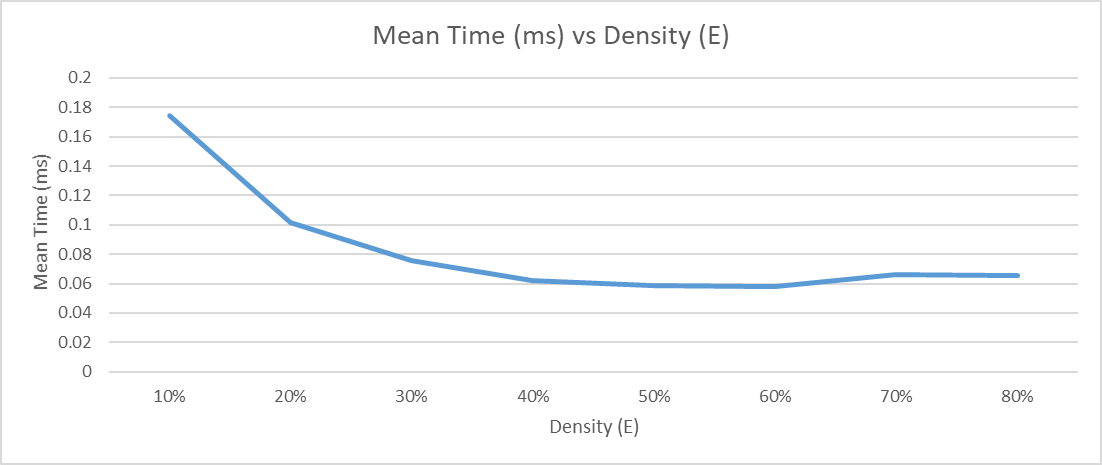
300 Number of Iterations per each Density



****

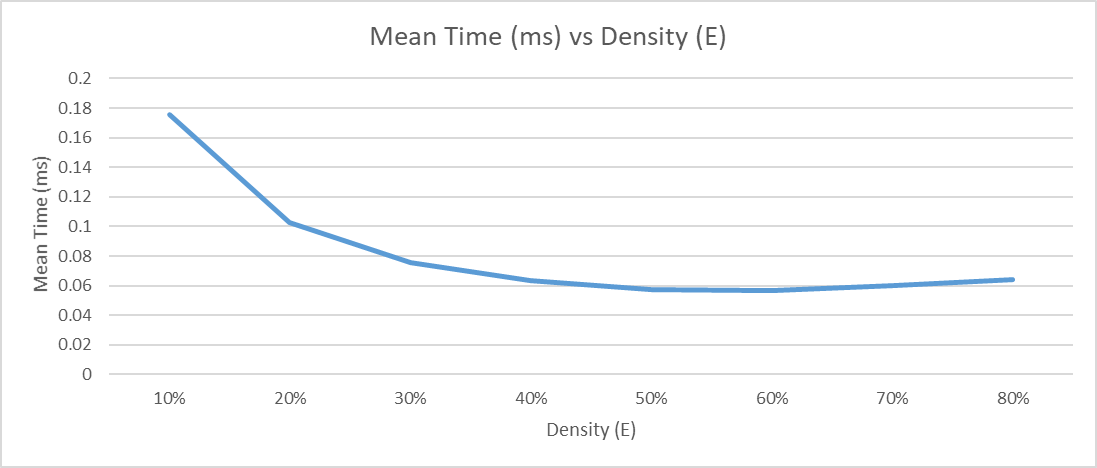
600 Number of Iterations per each Density



****

1000 Number of Iterations per each Density

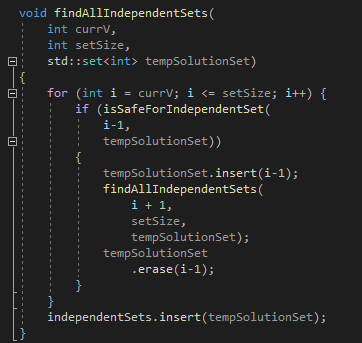


****

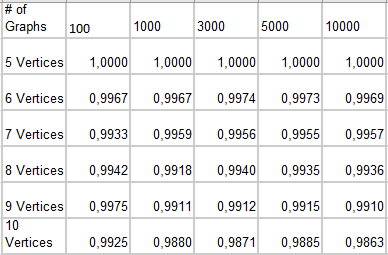
**Analysis of Ratio Bound**

Black box testing will be used with smaller graph sizes in order to check ratio bound. We defined the quality of the heuristic algorithm as following:

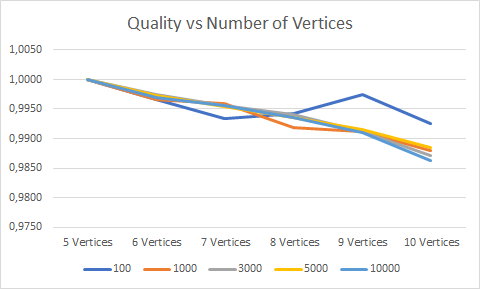
Quality = MISet.size() \* 1.0 / sizeMaximalIndependentSets() = 1 / Ratio



Above code in the given figure, is a brute force algorithm for finding all independent sets by iterating all possible combinations of vertices. This brute force algorithm is developed for the purpose of checking quality for smaller graphs.

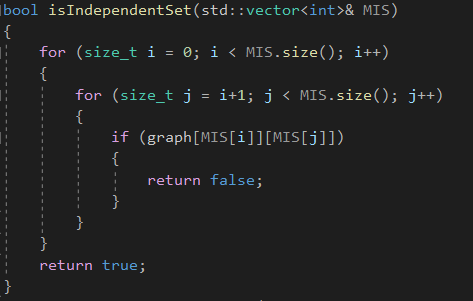
Quality vs Number of Vertices

When the number of vertices increases this tends to a decrease in the quality of Independent Set heuristic algorithms, as seen on the chart and the above image.

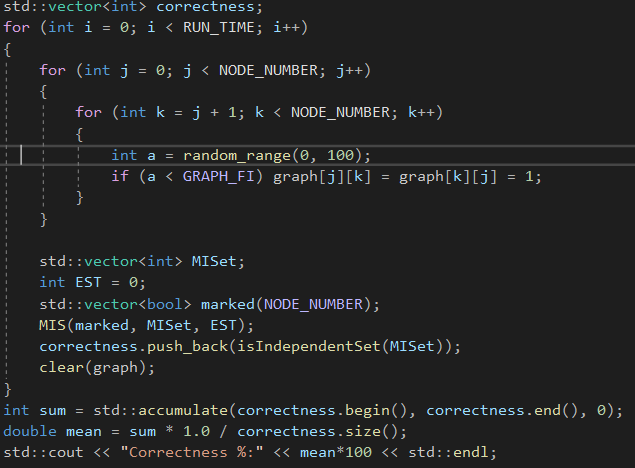


**Correctness and Testing of Heuristic Algorithm**

The heuristic algorithm that we implemented does not always provide maximum independent set but as proven in correctness of algorithm section it always provides an independent set since it covers all neighborhood of chosen vertex. Following algorithm is used for checking given set is independent or not. It traverses all vertices and check if there is edge between them.



To analyze if algorithm creates independent set or not following experiment conducted with 500 times run, 250 vertex node and edge density around fifty percent.



Given result:



Therefore, algorithm creates always independent set.

To test if algorithm gives maximum independent set or not black box testing method applied to algorithm along with a brute force algorithm. Since brute force algorithm works in exponential time, we were only able to test with small graphs. Given a random graph experiment compare heuristic and brute force algorithms. To get graph visualization from adjacency matrix an open-source service is used and it is available on graphonline.ru/en/

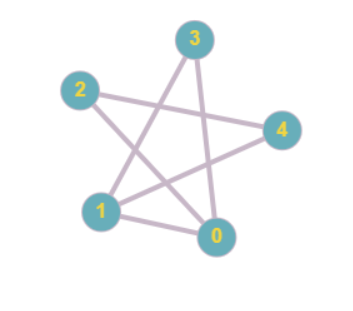
Experiment 1:

Figure 1: Experiment 1 graph

Given this graph, heuristic algorithm gives set {3, 2} and brute force algorithm gives all maximal independent sets {0, 4}, {1, 2}, {2, 3}, {3, 4} thus result is correct. Also, Estimation value given as 0.

Experiment 2:

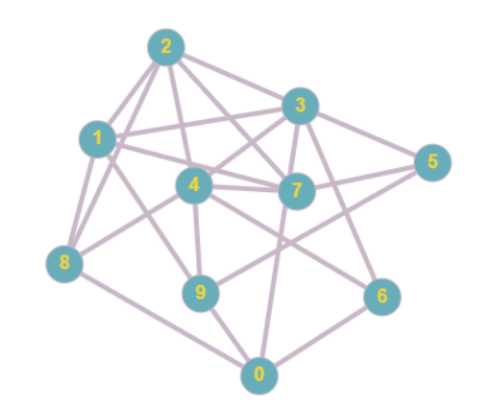


Figure 2: Experiment 2 graph

Given this graph, heuristic algorithm gives set {6, 8, 7, 9} and brute force algorithm gives all maximal independent sets {0, 1, 4, 5}, {3, 7, 8, 9}, {6, 7, 8, 9}. Thus, result is correct and estimation value is 1.

Experiment 3:

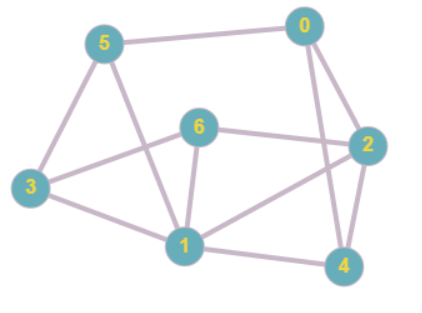


Figure 3: Experiment 3 graph

Given this graph, heuristic algorithm gives set {3, 0} and brute force algorithm gives maximal independent set {4, 5, 6}. Thus, result is incorrect and estimation value is 1. ratio bound holds.

Experiment 4:

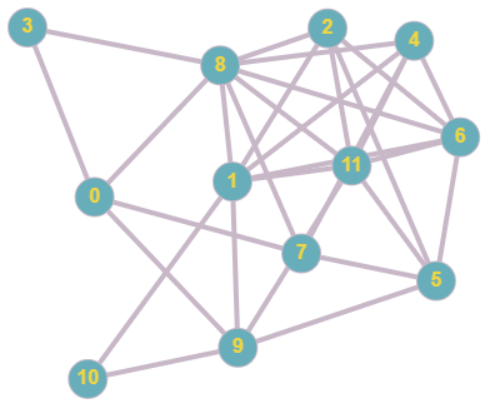


Figure 4: Experiment 4 graph

Given this graph, heuristic algorithm gives set {3, 10, 2, 4} and brute force algorithm gives all maximal independent sets {0, 2, 4, 10}, {0, 4, 5, 10}, {2, 3, 4, 9}, {2, 3, 4, 10}, {2, 3, 7, 9}, {2, 3, 7, 10}, {3, 4, 5, 10}, {3, 6, 7, 9}, {3, 6, 7, 10}, {3, 7, 10, 11}. Thus, result is correct and estimation value is 0.

Experiment 5:

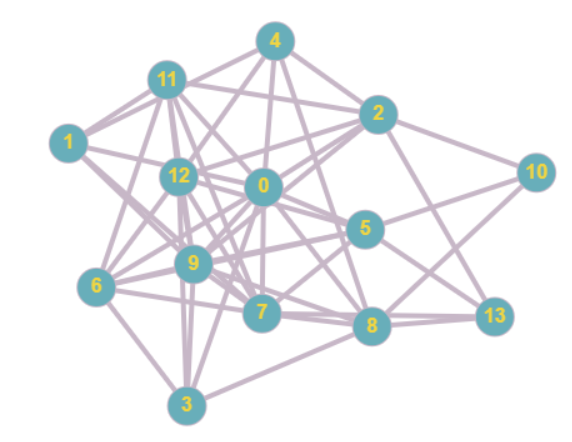


Figure 5: Experiment 5 graph

Given this graph, heuristic algorithm gives set {1, 6, 2, 8} and brute force algorithm gives all maximal independent sets {3, 4, 10, 11, 13}. Thus, result is incorrect and estimation value is 3. Ratio bound holds.

Experiment 6:



Figure 6: Experiment 6 graph

Given this graph, heuristic algorithm gives set {0, 7, 11, 12, 13, 15, 10, 1, 3, 6, 8, 16, 18} and brute force algorithm gives all maximal independent sets {0, 1, 3, 5, 6, 7, 8, 10, 12, 13, 15, 16, 18}, {0, 1, 3, 5, 6, 7, 8, 10, 12, 15, 16, 18, 19}, {0, 1, 3, 6, 7, 8, 10, 11, 12, 13, 15, 16, 18}, {0, 1, 3, 6, 7, 8, 10, 11, 12, 15, 16, 18, 19}, {0, 1, 3, 6, 7, 8, 10, 11, 13, 14, 15, 16, 18}{0, 1, 3, 6, 7, 8, 10, 11, 14, 15, 16, 18, 19}, {1, 2, 3, 5, 6, 7, 8, 10, 12, 13, 15, 16, 18}, {1, 2, 3, 5, 6, 7, 8, 10, 12, 15, 16, 18, 19}, {1, 2, 3, 6, 7, 8, 10, 11, 12, 13, 15, 16, 18}, {1, 2, 3, 6, 7, 8, 10, 11, 12, 15, 16, 18, 19}, {1, 2, 3, 6, 7, 8, 10, 11, 13, 14, 15, 16, 18}, {1, 2, 3, 6, 7, 8, 10, 11, 14, 15, 16, 18, 19}. Thus, result is correct and estimation value is 0.

Experiment 7:



Figure 7: Experiment 7 graph

Given this graph with no edges, heuristic algorithm gives set {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}and brute force algorithm gives all maximal independent sets {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}. Thus, result is correct and estimation value is 0.

Experiment 8:

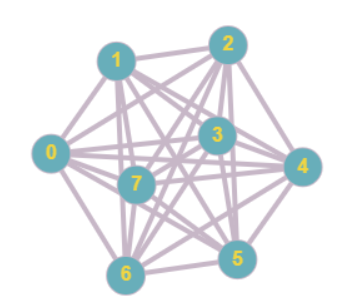


Figure 8: Experiment 8 graph

Given this complete graph, heuristic algorithm gives set {0} and brute force algorithm gives all maximal independent sets { 0 }, { 1 }, { 2 }, { 3 }, { 4 }, { 5 }, { 6 }, { 7 }. Thus, result is correct and estimation value is 0.

As given in ratio bound analysis section, algorithm works good with small graphs. For a last experiment, a graph with 80 vertex count around fifty percent edge density is conducted with long run time. Algorithm gives a set with 8 cardinality and brute force gives sets with 10 cardinality and estimation value is 7.

**Conclusion**

As a conclusion, Maximum independent set is a NP-Hard problem which has many usages in real life application. It is proven that decision version of this problem is NP-Complete that is reduced from maximum Clique problem. Thus, there is no known algorithm that solves this problem in polynomial time. However, there are heuristic algorithms that solves the problem that trade-offs optimal solution for runtime. In this project an approximation algorithm is implemented and analyzed. This algorithm gives an independent set with no guarantee of being maximum independent set with worst time complexity no more than O(). Moreover, it is shown that it has ratio bound that relies on output size. Thus, as cardinality of optimal solution increase, algorithm deviates from optimal solution.

Algorithm is experimentally analyzed in next section. Performance runtime is analyzed with respect to vertices number. It is seen that in practice algorithm has better runtime than O(). Additionally, runtime is analyzed with respect to edge density of graph and it is seen increase in edge density reduces runtime since cardinality of optimal solution must decrease with increasing edge density. Even in algorithm section it is proven that algorithm always provides independent set, correctness is experimentally analyzed since there can be coding errors. It is shown that algorithm gives independent set always. However, since it is a heuristic algorithm it does not guarantee giving maximal independent set. Therefore, algorithm output is analyzed on various graph and it is demonstrated that there are cases it does not give optimal solution. As graph size increase algorithm result tend to stray away from optimal solution. For ratio bound, experiment shows that for small number of vertices, algorithm works near perfectly with 98 percent correctness which can be used to say ratio bound is loose bound.

**References**

[1] S. Ağralı, Z. Taşkın, A. Ünal. (2017). “Employee scheduling in service industries with flexible employee availability and demand.” *Omega*. [online]. 66, pp. 159–169. doi:10.1016/j.omega.2016.03.001

[2] N. Gainanov, N. Mladenovic, V. Rasskazova, D. Urosevic. (2018). “Heuristic Algorithm for Finding the Maximum Independent Set with Absolute Estimate of the Accuracy”. *CEUR-WS.* [online]. 2098, pp. 141–149. Available: <http://ceur-ws.org/Vol-2098/paper12.pdf> [Jan. 10, 2021].

[3] Anonymous , “Proof that Independent Set in Graph theory is NP Complete”. Internet: <https://www.geeksforgeeks.org/proof-that-independent-set-in-graph-theory-is-np-complete/>, Jun. 26, 2020 [Jan. 25,2021].