CS301

2024-2025 Fall

Project Final Report

Group 22

Alper Çamlı-30858

Dilara Alkanalka-30758

1. Problem Description

1.1 Overview:

The Independent Set Problem is a problem in graph theory, which focuses on finding a subset of vertices in a graph in a way that no two vertices in the subset are adjacent. The subset in this case is known as an "independent set." The main goal of the problem is to find whether an independent set of a certain given size exists in a graph and also to find the largest possible independent set within a certain graph which has various applications in many real life domains. There occurs a great challenge for the problem especially as the graphs get larger and more complex.

1.2 Decision Problem:

In the decision problem form of independent set problem, the aim is to find if there exists an independent set of size *k*. The decision problem is formulated as a yes-or-no question, it asks: Given a graph G=(V,E) is there an independent set in the graph with at least *k* vertices?

1.3 Optimization Problem:

The optimization task of the problem aims to maximize the size of the independent set within the given graph, with the goal to find the largest possible subset of vertices that no two vertices are adjacent, also known as the maximum independent set.

1.4 Example Instances:

Consider the image below where various types of graphs and their independent sets are shown. The Wheel Graph W8 consists of an outer cycle connected with a central vertex, here consecutive cycle vertices and the central vertex are avoided to get the independent set. Utility Graph K3,3 (Bipartite Graph) consists of two sets of vertices where each vertex is connected to every vertex in the other set, to get the independent set vertices from one of these partitions are chosen. Petersen Graph is a symmetric graph that has a unique connectivity between vertices and the vertices to get the independent set are chosen carefully based on these connections. Frucht Graph is another unique graph that has no particular symmetry or a structure also

explained as no automorphism other than identity, and the vertices are selected based on avoiding any direct connections to get the independent set.

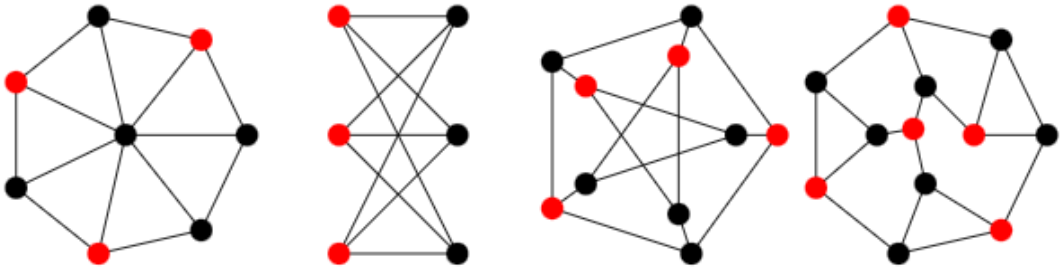


Figure 1: Independent Set by Mathworld

The image below shows a graph of a cube with eight vertices and six independent sets with the condition of not having any two vertices that connect directly. Here the theory of maximal independent set can be examined, in graph theory, a maximal independent set (MIS) or maximal stable set is an independent set that is not a subset of any other independent set.(wikipedia) For the image below, the two of the independent sets are maximal independent set.

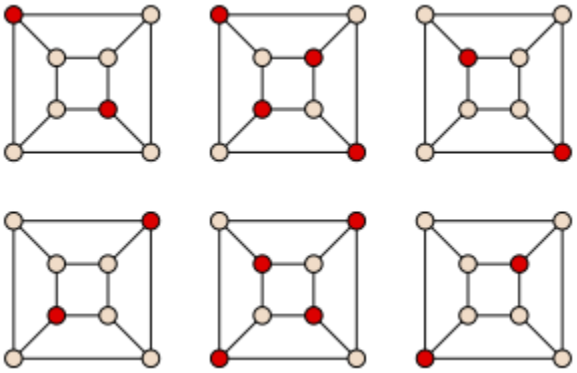


Figure 2: Maximal Independent Set by Wikipedia

1.5 Real World Applications:

The Independent Set Problem has many applications adapted to use in real life practices especially in computer science, resource allocation in cloud computing and wireless network optimization can be given as examples to these real life practices.

1.5.1 Resource Allocation in Cloud Computing: In cloud computing, different tasks or virtual machines may require different resources, however assigning the resources can be hard to execute as it may cause conflict in memory, bandwidth etc. In this case each task or virtual machine is considered as a vertex and the edges exist for allocating the resources. An independent set is the tasks or virtual machines that can run concurrently without causing any conflicts. The problem is used to optimize the resource allocation in cloud services to allow service more efficiently and to more clients without outdoing the certain limits.

1.5.2 Wireless Network Optimization: In wireless networks, each device or transmitter needs to use a different channel or frequency to avoid intervention and conflict between the channels. In this context each transmitter can be represented as a vertex of a graph and an edge exists if the two transmitters are close enough that the frequencies can interfere and cause a conflict between each other. The Independent Set Problem is used so that the frequency allocations are done efficiently with the specific limits in many areas such as telecommunications and internet networks.

1.6 Hardness of the Problem:

The Independent Set problem is NP- complete. As the problem is both NP and NP-hard, it also follows that the problem is NP-complete.

Proving NP: To prove if the problem is in NP class, we need to be able to verify that there is a potential solution in polynomial time given a subset of vertices for the case that no two vertices have a shared edge. Based on the proof available on GeeksforGeeks, we can check the solution by simply checking the vertices belonging to the independent set based on the given condition, which can be done in polynomial time, which is O(V+E) on the graph G(V, E).(GeeksforGeeks, n.d.).

Proving NP-hard: In order to demonstrate the NP-hardness of the Independent Set Problem, a reduction from a well-known NP-hard problem can be done. The execution for this solution is done by choosing the Clique Problem which is demonstrated on GeeksforGeeks[3]. For every instance of the Clique Problem consisting of graph G(V, E) and integer k, it can be transformed to the demanded graph format as G' (V', E') and k' for the Independent Set Problem. The time complexity of traversing all the vertices and edges to compute the graph G' is O(V+E). If it is assumed that graph G has a clique with k size, which suggests that each vertex is connected by an edge with the remaining vertices, which also suggests that these edges are in graph G and can not be in G'. This suggests that k vertices are not connected to each other and form an independent set with size k. It is assumed that complementary graph G' also has an independent set of vertices with size k'. when the graph is complemented to obtain G, these vertices will be adjacent to each other and G will have a clique of size k. Therefore it is proved that if there is a clique of size k in G', there is an independent set of size k in graph G. "Therefore any instance of the independent set problem can be reduced to an instance of the clique problem.Thus, the independent set is NP-Hard." (GeeksforGeeks, n.d.).

Proving NP-complete: As demonstrated, The Independent Set Problem is established as a member of NP class and also NP-hardness, since the problem's solution can be verified with polynomial time, also by reducing the graph to Clique Problem it is proven that the problem is NP-hard. Thus the Independent Set Problem is also defined as NP-complete.

2. Algorithm Description

2.1 Brute Force Algorithm

2.1.1 Overview:

For the Independent Set Problem, the brute-force approach involves generating all possible subsets of vertices and checking each subset to see if it qualifies as an independent set. An independent set, in this context, is a subset of vertices in which no two vertices are adjacent. This means that for every pair of vertices in the subset, there is no edge connecting them in the graph.

The brute-force method is known for its high computational cost, which makes it impractical for large graphs. Specifically, if the graph has n vertices, then there are 2^n possible subsets of vertices. Thus, the brute-force algorithm has an exponential time complexity, making it inefficient for large instances of the problem.

2.1.2 Pseudocode:

Loop Over All Subsets: The for each subset S of vertices in G loop conceptually generates each subset of vertices in G. Since there are 2^n possible subsets of n vertices mentioned in the paper of A.Vince[4], this loop will iterate 2^n times.

Check for Independence: For each subset, the **IsIndependentSet** function is called. This function checks all pairs of vertices in S to ensure that no two vertices are adjacent. In the worst case, this check takes O(n^2) time if S includes all n vertices.

Update the Maximum Independent Set: If the current subset S is independent and its size is larger than **max_size**, we update **max_size** and **max_set** to reflect this new larger independent set.

Return the Result: After all subsets have been checked, the function returns **max_set**, which contains the vertices of the largest independent set found.

```
function IsIndependentSet(graph G, subset S):
    for each pair of vertices (u, v) in subset S:
        if (u, v) is an edge in G:
            return false
    return true

function MaxIndependentSet(graph G):
    max_size = 0
    max_set = {}
    for each subset S of vertices in G:
        if IsIndependentSet(G, S):
            if size(S) > max_size:
                max_size = size(S)
                max_set = S
```

```
    return max_set
```

2.2 Heuristic Algorithm

The heuristic algorithm for the Independent Set Problem uses a greedy approach to build an independent set efficiently in polynomial time. Starting with an empty set, the algorithm iteratively selects a vertex with the fewest neighbors (to minimize conflicts) and adds it to the independent set. After each selection, the chosen vertex and its neighbors are removed from the graph to ensure the independence property[6]. This process continues until no vertices remain.

The greedy algorithm operates based on the principle of making locally optimal choices at each step to approximate a global solution. Although it does not guarantee the maximum independent set, it is effective for large graphs, with a time complexity of O(n + m), where n is the number of vertices and m is the number of edges. By choosing vertices with the fewest neighbors, this specific heuristic minimizes conflicts and typically performs well in sparse graphs.

2.2.2 Pseudocode:

The **`GreedyIndependentSet`** function begins with an empty set and iteratively selects vertices to add to the independent set. At each step, the function **`SelectVertexWithFewestNeighbors`** identifies a vertex with the smallest degree in the graph. This choice minimizes the number of conflicts, as fewer neighbors mean fewer vertices need to be removed from the graph.

Once a vertex is selected, the function **`RemoveVertexAndNeighbors`** ensures that no edges exist between vertices in the independent set by removing the selected vertex and all its neighbors from the graph. This guarantees that the independence property is maintained as the algorithm progresses. The algorithm continues this process until no vertices remain in the graph.

```
function GreedyIndependentSet(graph G):

    independent_set = {}
```

```
    while G is not empty:
        vertex = SelectVertexWithFewestNeighbors(G)
        independent_set.add(vertex)
        RemoveVertexAndNeighbors(G, vertex)
    return independent_set

function SelectVertexWithFewestNeighbors(graph G):
    min_neighbors = ∞
    selected_vertex = null
    for each vertex v in G:
        if degree(v) < min_neighbors:
            min_neighbors = degree(v)
            selected_vertex = v
    return selected_vertex

function RemoveVertexAndNeighbors(graph G, vertex):
    for each neighbor u of vertex:
        RemoveVertex(G, u)
    RemoveVertex(G, vertex)
```

3. Algorithm Analysis

3.1 Brute Force Algorithm

3.1.1 Correctness Analysis:

Theorem:

The brute-force algorithm finds the maximum independent set in any given graph G.

Proof:

By construction, the algorithm checks all possible subsets of vertices in G and verifies the independence property (where no edges exist between any two vertices in a subset). It keeps track of the largest independent set found, guaranteeing that it returns the maximum. Since no possible independent set is missed, and no larger independent set is possible.

3.1.2 Time Complexity:

Tight Upper Bound for Time Complexity

To analyze the time complexity tightly, let's look more closely at the two main operations the algorithm performs:

1. Generating All Subsets of Vertices
    - Given n vertices in the graph, there are 2^n possible subsets (since each vertex can either be included in or excluded from a subset).
    - Therefore, the algorithm iterates through all 2^n subsets.
2. Checking Independence for Each Subset
    - For each subset, the algorithm checks whether the subset is independent, i.e., it checks that no two vertices in the subset are adjacent.
    - To check independence, the algorithm iterates over each edge in the subset.
    - In the worst case, each subset may contain up to n vertices, and the maximum number of edges among these vertices (in a complete graph) is the combination of n by which equals $n(n-1)/2$, which is also $O(n^2)$.

- So, for each subset, the independence check takes O(n^2) time.

Combining the Operations

Since there are 2^n subsets and each subset requires O(n^2) time to check for independence, the total time complexity T(n) of the algorithm is:

$$T(n) = O(2^n) \cdot O(n^2) = O(n^2 \cdot 2^n)$$

Proof of Tight Upper Bound

1. Counting the Subsets — We know there are exactly 2^n subsets for a set of n vertices, so the subset generation step is bounded by O(2^n).
2. Bounding the Independence Check — For each subset, the algorithm needs to verify independence by iterating over pairs of vertices in the subset. In the worst case, a subset might contain all n vertices, resulting in O(n^2) comparisons for that subset.
3. Total Work Done — Since each subset check takes O(n^2) time and there are 2^n subsets, we multiply these to get the total upper bound where c and k are the constant number of operations which are redundant to calculate in asymptotic notation:

$$T(n) = c(2^n) \cdot k(n^2)$$

This bound is tight because the brute-force approach cannot skip any subset or avoid the independence check for each subset without potentially missing the largest independent set. Therefore, any faster solution would miss potential configurations, making $\Theta(n^2 \cdot 2^n)$ the tight upper bound.

Conclusion

Thus, the tight upper bound for the time complexity of this brute-force algorithm is:

$$\Theta(2^n)$$

This exponential growth indicates why the brute-force approach is computationally infeasible for large graphs, as the running time increases very quickly with even small increases in n.

3.1.3 Space Complexity:

The space complexity of the algorithm is primarily due to the storage of subsets and especially the largest independent set found.

1. Subset Storage: At any given time, the algorithm stores information about the subset it's currently evaluating, which takes **O(n)** space, as each subset can have up to n vertices.
2. Maximum Set Storage: The algorithm keeps track of the largest independent set found so far, which also requires **O(n)** space.

Therefore, the total space complexity is: ***O(n)***

This is relatively low compared to the time complexity, as the algorithm does not need to store all subsets simultaneously.

3.2 Heuristic Algorithm

3.2.1 Correctness Analysis:

Theorem

The heuristic algorithm for the Independent Set Problem is correct in the sense that it always produces a valid independent set for the input graph G.

Proof

To demonstrate correctness, we must show that the output of the heuristic algorithm satisfies the independence property, meaning no two vertices in the output set are adjacent. While the algorithm may not guarantee the maximum independent set, it ensures that the result is always a valid independent set.

1. **Base Case**: Initially, the independent set is empty, which is trivially a valid independent set.
2. **Inductive Step**: At each iteration, the algorithm selects a vertex v from the graph and adds it to the independent set. After adding v, it removes v and all its neighbors from the graph. This guarantees that no other vertices in the independent set are adjacent to v, as all conflicts are resolved by the removal process. Consequently, the independence property is maintained.
3. **Termination**: The algorithm continues until no vertices remain in the graph. At this point, the independent set contains only vertices that were added without adjacency conflicts.

Since the algorithm explicitly checks and removes all potential conflicts (adjacent vertices) at each step, it is guaranteed to produce a valid independent set. The result satisfies the definition of the problem, even if it is not guaranteed to be the largest independent set.

3.2.2 Time Complexity:

To derive the tight worst-case time complexity for the heuristic algorithm, we analyze each component of the algorithm.

Key Operations

1. **Vertex Selection**:

The `SelectVertexWithFewestNeighbors` function iterates over all remaining vertices in the graph to find the vertex with the fewest neighbors. For a graph with n vertices and m edges:

- Iterating through all vertices takes O(n).
- Calculating the degree of each vertex, which involves iterating through its neighbors, takes O(m) in total across all iterations of the algorithm.

Thus, the time complexity of vertex selection is O(n) for each iteration, and this operation is repeated for all remaining vertices.

2. **Vertex and Neighbor Removal**:

The **`RemoveVertexAndNeighbors`** function removes the selected vertex and its neighbors from the graph. Each removal involves updating the adjacency lists of the affected vertices.

Since each edge is processed at most once during the entire algorithm (either to remove a vertex or to check its degree), the total time spent on this operation is O(m) across all iterations.

Total Time Complexity

- The algorithm performs n iterations, as a vertex is selected and removed at each step.
- In each iteration, vertex selection takes O(n), and edge removal is bounded by O(m) across all iterations.

Combining these:

$$\mathbf{T(n,m) = O(n \cdot n)+O(m) = O(n^2 + m)}$$

In terms of a tight bound using **Θ**:

$$\mathbf{T(n, m) = \Theta(n^2 + m)}$$

Tight Bound Justification

The worst case for the heuristic algorithm occurs when the structure of the graph leads to inefficient choices during vertex selection. This inefficiency arises when:

1. **The heuristic repeatedly selects suboptimal vertices** (vertices that lead to a small independent set). In sparse graphs ($m \approx n$), the **Θ**(n^2) term dominates because vertex selection occurs n times, each taking O(n) steps.
2. **The graph has a dense structure**, where most vertices have high degrees, leading to frequent large-scale removals of vertices and their neighbors. In dense graphs ($m \approx n^2$), the **Θ**(m) term dominates as the graph update steps process every edge.

Thus, the algorithm's time complexity is **Θ(n^2 + m)**, and this bound is tight since every vertex and edge is processed exactly once during the algorithm.

3.2.3 Space Complexity:

The space complexity of the heuristic algorithm depends on the graph representation, the independent set storage, and temporary variables.

- **Graph Representation**: Using an adjacency list, the graph requires O(n + m) space, where n is the number of vertices and mmm is the number of edges.
- **Independent Set Storage**: The independent set requires O(α(G)), where α(G) is the size of the largest independent set, which is at most n.
- **Temporary Variables**: Fixed O(1) space is used for operations like vertex selection and edge removal.

Overall, the total space complexity is:

$$\mathbf{O(n + m + \alpha(G))}$$

For sparse graphs m = O(n), this simplifies to O(n). For dense graphs m = O(n^2), it becomes O(n^2), dominated by the graph representation.

4. **Sample Generation (Random Instance Generator)**

The Independent Set Problem is handled on a graph, with the aim to find a set of vertices that no two vertices in the subset are adjacent. For creation of the random instances for the problem, random generation of graphs with specified number of vertices and edges is needed. The generation algorithm needed for this case depends on parameters such as number of vertices (n) and probability of an edge that occurs between two vertices for the graph density (p). The idea for the graph generation algorithm was taken from GeeksforGeeks' article on Erdős–Rényi model[8].

The pseudo-code for the algorithm is as follows:

Input: n (number of vertices), p (edge probability)

Output: Graph G = (V, E)

1. Initialize an empty graph G with n vertices: V = {v1, v2, ..., vn}, E = ∅

2. For each pair of vertices (u, v) where u ≠ v:

a. Generate a random number r in [0, 1].

b. If r ≤ p: Add the edge (u, v) to E.

3. Return the graph G = (V, E)

The python code for the algorithm is as follows:

```
import random

def generate_random_graph(n, p):
    graph = {i: [] for i in range(n)}  # initializing list
    for i in range(n):
        for j in range(i + 1, n):  # checking all pairs (i, j)
            if random.random() <= p:  # adding edge with probability p
                graph[i].append(j)
                graph[j].append(i)
    return graph

# Example:
n = 5  # number of vertices
p = 0.5  # edge probability
random_graph = generate_random_graph(n, p)
print(random_graph)
```

The sample output for this code is:

```
{0: [2, 4], 1: [2, 4], 2: [0, 1, 4], 3: [4], 4: [0, 1, 2, 3]}
```

5. Algorithm Implementations

5.1 Brute Force Algorithm

5.1.1 Algorithm and Sample Runs:

```
import random
```

```
from itertools import combinations



# Function to generate a random graph

def generate_random_graph(n, p):

    graph = {i: [] for i in range(n)}  # Initialize adjacency list

    for i in range(n):

        for j in range(i + 1, n):  # Iterate over all pairs (i, j)

            if random.random() <= p:  # Add edge with probability p

                graph[i].append(j)

                graph[j].append(i)

    return graph

# Function to check if a subset is an independent set

def is_independent_set(graph, subset):

    for i, j in combinations(subset, 2):  # Check all pairs in the subset

        if j in graph[i]:  # If edge exists, not independent

            return False

    return True

# Brute force to find the maximum independent set

def find_max_independent_set(graph):

    n = len(graph)

    max_set = []

    for r in range(n + 1):  # Check subsets of all sizes

        for subset in combinations(range(n), r):
```

```
                if is_independent_set(graph, subset) and len(subset) >
len(max_set):
                    max_set = subset
    return max_set
def main():
    # Get user input for number of vertices and edge probability
    n = int(input("Enter the number of vertices (n): "))
    p = float(input("Enter the edge probability (p between 0 and 1): "))
    # Generate the random graph
    random_graph = generate_random_graph(n, p)
    print("Generated Random Graph (Adjacency List):", random_graph)

    max_set = find_max_independent_set(random_graph)
    print("Maximum Independent Set:", max_set)
if __name__ == "__main__":
    main()
```

Sample runs and outputs (15 samples):

-Enter the number of vertices (n): 5

Enter the edge probability (p between 0 and 1): 0.5

Generated Random Graph (Adjacency List): {0: [2], 1: [], 2: [0, 3, 4], 3: [2, 4], 4: [2, 3]}

Maximum Independent Set: **(0, 1, 3)**

-Enter the number of vertices (n): 5

Enter the edge probability (p between 0 and 1): 0.1

Generated Random Graph (Adjacency List): {0: [], 1: [], 2: [], 3: [], 4: []}

Maximum Independent Set: **(0, 1, 2, 3, 4)**

-Enter the number of vertices (n): 5

Enter the edge probability (p between 0 and 1): 0.9

Generated Random Graph (Adjacency List): {0: [1, 2, 3, 4], 1: [0, 3, 4], 2: [0, 3, 4], 3: [0, 1, 2, 4], 4: [0, 1, 2, 3]}

Maximum Independent Set: **(1, 2)**

-Enter the number of vertices (n): 8

Enter the edge probability (p between 0 and 1): 0.4

Generated Random Graph (Adjacency List): {0: [2, 3, 4, 5, 7], 1: [3], 2: [0, 3, 5, 7], 3: [0, 1, 2, 4], 4: [0, 3, 5, 6, 7], 5: [0, 2, 4, 6, 7], 6: [4, 5, 7], 7: [0, 2, 4, 5, 6]}

Maximum Independent Set: **(0, 1, 6)**

-Enter the number of vertices (n): 9

Enter the edge probability (p between 0 and 1): 0.5

Generated Random Graph (Adjacency List): {0: [1, 2, 3, 4, 7, 8], 1: [0, 2, 6], 2: [0, 1, 3, 4, 6], 3: [0, 2, 4, 5, 7, 8], 4: [0, 2, 3, 6], 5: [3, 8], 6: [1, 2, 4], 7: [0, 3], 8: [0, 3, 5]}

Maximum Independent Set: **(1, 4, 5, 7)**

-Enter the number of vertices (n): 10

Enter the edge probability (p between 0 and 1): 0.3

Generated Random Graph (Adjacency List): {0: [7, 9], 1: [2, 7, 9], 2: [1, 9], 3: [], 4: [8, 9], 5: [], 6: [9], 7: [0, 1], 8: [4, 9], 9: [0, 1, 2, 4, 6, 8]}

Maximum Independent Set: **(0, 1, 3, 4, 5, 6)**

-Enter the number of vertices (n): 12

Enter the edge probability (p between 0 and 1): 0.8

Generated Random Graph (Adjacency List): {0: [1, 2, 3, 4, 6, 7, 8, 9, 10], 1: [0, 2, 4, 5, 6, 7, 8, 9, 10, 11], 2: [0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11], 3: [0, 2, 4, 5, 6, 7, 8, 9, 10, 11], 4: [0, 1, 2, 3, 5, 6, 7, 8, 11], 5: [1, 2, 3, 4, 6, 8, 10, 11], 6: [0, 1, 2, 3, 4, 5, 7, 10, 11], 7: [0, 1, 2, 3, 4, 6, 9, 11], 8: [0, 1, 2, 3, 4, 5, 10, 11], 9: [0, 1, 2, 3, 7, 10, 11], 10: [0, 1, 2, 3, 5, 6, 8, 9, 11], 11: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}

Maximum Independent Set: **(6, 8, 9)**

-Enter the number of vertices (n): 13

Enter the edge probability (p between 0 and 1): 0.2

Generated Random Graph (Adjacency List): {0: [3, 7], 1: [2, 5], 2: [1, 11, 12], 3: [0, 8], 4: [7, 10, 11], 5: [1, 9, 11], 6: [], 7: [0, 4, 10], 8: [3, 12], 9: [5], 10: [4, 7], 11: [2, 4, 5], 12: [2, 8]}

Maximum Independent Set: **(0, 1, 6, 8, 9, 10, 11)**

-Enter the number of vertices (n): 14

Enter the edge probability (p between 0 and 1): 0.5

Generated Random Graph (Adjacency List): {0: [2, 4, 5, 7, 9], 1: [2, 3, 4, 5, 8, 9, 10, 11, 12], 2: [0, 1, 4, 5, 6, 7, 8, 9, 10, 12], 3: [1, 9, 13], 4: [0, 1, 2, 8, 9, 11, 12, 13], 5: [0, 1, 2, 7, 10, 11], 6: [2, 7, 10, 11, 12], 7: [0, 2, 5, 6, 8, 9, 10, 11, 12, 13], 8: [1, 2, 4, 7, 10, 11], 9: [0, 1, 2, 3, 4, 7, 10, 11], 10: [1, 2, 5, 6, 7,

8, 9, 11, 13], 11: [1, 4, 5, 6, 7, 8, 9, 10, 13], 12: [1, 2, 4, 6, 7, 13], 13: [3, 4, 7, 10, 11, 12]}

Maximum Independent Set: **(5, 6, 8, 9, 13)**

-Enter the number of vertices (n): 15

Enter the edge probability (p between 0 and 1): 0.2

Generated Random Graph (Adjacency List): {0: [1, 10], 1: [0, 4, 8, 9, 13], 2: [5, 7, 9, 11, 14], 3: [], 4: [1, 8, 11, 13], 5: [2, 7, 8], 6: [11], 7: [2, 5, 13], 8: [1, 4, 5, 12], 9: [1, 2, 10, 14], 10: [0, 9], 11: [2, 4, 6], 12: [8], 13: [1, 4, 7], 14: [2, 9]}

Maximum Independent Set: **(0, 3, 4, 5, 6, 9, 12)**

-Enter the number of vertices (n): 15

Enter the edge probability (p between 0 and 1): 0.9

Generated Random Graph (Adjacency List): {0: [1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14], 1: [0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14], 2: [0, 3, 4, 5, 6, 7, 8, 9, 12, 13], 3: [2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14], 4: [0, 1, 2, 3, 5, 6, 7, 8, 9, 11, 12, 13, 14], 5: [0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 13, 14], 6: [0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14], 7: [0, 1, 2, 3, 4, 5, 6, 8, 10, 11, 12, 13, 14], 8: [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14], 9: [0, 1, 2, 3, 4, 5, 6, 8, 10, 11, 13, 14], 10: [0, 1, 3, 5, 6, 7, 8, 9, 11, 12, 13, 14], 11: [0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14], 12: [0, 1, 2, 3, 4, 6, 7, 8, 10, 11, 14], 13: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14], 14: [0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]}

Maximum Independent Set: **(0, 3)**

-Enter the number of vertices (n): 16

Enter the edge probability (p between 0 and 1): 0.4

Generated Random Graph (Adjacency List): {0: [1, 4, 7, 9, 10, 11, 13, 14], 1: [0, 2, 5, 7, 8, 14], 2: [1, 7, 10, 14], 3: [5, 6, 10, 12], 4: [0, 10], 5: [1, 3, 6, 8, 11, 12], 6: [3, 5, 9, 10], 7: [0, 1, 2, 10], 8: [1, 5, 10, 11, 12, 15], 9: [0, 6, 11, 14], 10: [0, 2, 3, 4, 6, 7, 8, 12, 13], 11: [0, 5, 8, 9], 12: [3, 5, 8, 10, 13, 15], 13: [0, 10, 12], 14: [0, 1, 2, 9, 15], 15: [8, 12, 14]}

Maximum Independent Set: **(1, 3, 4, 9, 13, 15)**

-Enter the number of vertices (n): 18

Enter the edge probability (p between 0 and 1): 0.6

Generated Random Graph (Adjacency List): {0: [2, 3, 5, 6, 9, 11, 12, 13, 14, 15, 17], 1: [4, 5, 6, 9, 10, 11, 12, 13, 14, 15, 16, 17], 2: [0, 4, 7, 9, 13, 14, 17], 3: [0, 5, 6, 7, 9, 10, 11, 14, 15, 16], 4: [1, 2, 5, 6, 8, 9, 12, 13, 14, 16, 17], 5: [0, 1, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 17], 6: [0, 1, 3, 4, 5, 7, 12, 13, 16], 7: [2, 3, 5, 6, 8, 9, 10, 12, 13, 14, 15, 16], 8: [4, 5, 7, 10, 11, 13, 14, 15, 16], 9: [0, 1, 2, 3, 4, 5, 7, 10, 14, 15, 17], 10: [1, 3, 7, 8, 9, 13, 14, 15, 16, 17], 11: [0, 1, 3, 5, 8, 15, 17], 12: [0, 1, 4, 5, 6, 7, 14, 15, 17], 13: [0, 1, 2, 4, 5, 6, 7, 8, 10, 17], 14: [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 12, 17], 15: [0, 1, 3, 5, 7, 8, 9, 10, 11, 12], 16: [1, 3, 4, 6, 7, 8, 10, 17], 17: [0, 1, 2, 4, 5, 9, 10, 11, 12, 13, 14, 16]}

Maximum Independent Set: **(9, 11, 12, 13, 16)**

-Enter the number of vertices (n): 25

Enter the edge probability (p between 0 and 1): 0.3

Generated Random Graph (Adjacency List): {0: [6, 10, 14, 17, 18, 21, 23], 1: [5, 6, 9, 14, 15, 22], 2: [4, 6, 7, 10, 12, 13, 15, 21, 22, 23], 3: [14, 17, 21, 22], 4: [2, 8, 11, 17, 21], 5: [1, 10, 13, 17, 19, 24], 6: [0, 1, 2, 9, 11, 12, 18, 20, 22], 7: [2, 9, 13, 14, 15, 22], 8: [4, 10, 11, 18, 21], 9: [1, 6, 7, 11, 15, 17, 18, 21, 22],

10: [0, 2, 5, 8, 16, 24], 11: [4, 6, 8, 9, 13, 14, 18], 12: [2, 6, 14, 21, 22, 24], 13: [2, 5, 7, 11, 14, 16, 21, 22], 14: [0, 1, 3, 7, 11, 12, 13, 15, 18, 19, 20, 22, 24], 15: [1, 2, 7, 9, 14, 19, 21, 22, 23], 16: [10, 13, 19, 24], 17: [0, 3, 4, 5, 9, 22, 23, 24], 18: [0, 6, 8, 9, 11, 14, 22], 19: [5, 14, 15, 16, 22], 20: [6, 14], 21: [0, 2, 3, 4, 8, 9, 12, 13, 15, 22, 24], 22: [1, 2, 3, 6, 7, 9, 12, 13, 14, 15, 17, 18, 19, 21], 23: [0, 2, 15, 17], 24: [5, 10, 12, 14, 16, 17, 21]}

Maximum Independent Set: **(1, 3, 4, 7, 10, 12, 18, 19, 20, 23)**

-Enter the number of vertices (n): 27

Enter the edge probability (p between 0 and 1): 0.6

Generated Random Graph (Adjacency List): {0: [1, 2, 3, 4, 5, 6, 10, 11, 12, 13, 14, 17, 19, 21, 22, 26], 1: [0, 2, 3, 4, 5, 8, 9, 11, 12, 13, 14, 16, 17, 18, 19, 20, 25, 26], 2: [0, 1, 4, 5, 7, 9, 10, 13, 15, 17, 18, 21, 22, 23, 24, 26], 3: [0, 1, 4, 5, 6, 10, 13, 14, 15, 17, 18, 19, 21, 22, 23, 24, 25], 4: [0, 1, 2, 3, 6, 7, 8, 9, 10, 13, 16, 17, 18, 21, 22, 24, 25, 26], 5: [0, 1, 2, 3, 7, 9, 10, 12, 14, 15, 16, 23, 25, 26], 6: [0, 3, 4, 8, 10, 11, 12, 13, 14, 15, 20, 25, 26], 7: [2, 4, 5, 8, 9, 10, 11, 12, 13, 15, 19, 21, 22, 23, 24, 25], 8: [1, 4, 6, 7, 9, 10, 13, 14, 15, 16, 17, 20, 21, 24, 25, 26], 9: [1, 2, 4, 5, 7, 8, 11, 13, 14, 15, 16, 17, 18, 19, 22, 24, 25], 10: [0, 2, 3, 4, 5, 6, 7, 8, 11, 14, 16, 18, 20, 21, 23, 26], 11: [0, 1, 6, 7, 9, 10, 12, 14, 15, 18, 19, 22, 24], 12: [0, 1, 5, 6, 7, 11, 14, 17, 20, 22, 23, 24, 25], 13: [0, 1, 2, 3, 4, 6, 7, 8, 9, 14, 15, 16, 17, 19, 20, 21, 23, 26], 14: [0, 1, 3, 5, 6, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19, 21, 24, 25, 26], 15: [2, 3, 5, 6, 7, 8, 9, 11, 13, 14, 16, 19, 20, 21, 22, 25], 16: [1, 4, 5, 8, 9, 10, 13, 15, 17, 18, 21, 24, 26], 17: [0, 1, 2, 3, 4, 8, 9, 12, 13, 14, 16, 20, 21, 23, 26], 18: [1, 2, 3, 4, 9, 10, 11, 14, 16, 19, 20, 21, 26], 19: [0, 1, 3, 7, 9, 11, 13, 14, 15, 18, 20, 21, 24], 20: [1, 6, 8, 10, 12, 13, 15, 17, 18, 19, 22, 23, 25, 26], 21: [0, 2, 3, 4, 7, 8, 10, 13, 14, 15, 16, 17, 18, 19, 22, 23, 24, 26], 22: [0, 2, 3, 4, 7, 9, 11, 12, 15, 20, 21, 23, 24, 26], 23: [2, 3, 5, 7, 10, 12, 13, 17, 20, 21, 22, 24, 25, 26], 24: [2, 3, 4, 7, 8, 9, 11, 12, 14, 16, 19, 21, 22, 23, 26], 25: [1, 3, 4, 5, 6, 7, 8, 9, 12, 14, 15, 20,

```
23], 26: [0, 1, 2, 4, 5, 6, 8, 10, 13, 14, 16, 17, 18, 20, 21, 22, 23,
24]}

Maximum Independent Set: (2, 3, 11, 16, 20)
```

5.1.2 Initial Test and Findings

The algorithm was tested with 15 randomly generated graphs with various sizes and different edge connection probability rates between 0.1 and 0.9 to create graphs with different sizes and densities. For graphs that had a number of vertices that were smaller than 25 the algorithm worked very efficiently and was very fast. For the graphs that had vertices between 25 and 30, the algorithm still worked well but it took relatively a lot more time for the code to work on the function to find the maximum independent set. When the vertice number was equal to and larger than 30, the code did not work well and started having runtime problems, the failure was in this case of vertex number input because the algorithm could not finish due to excessive computation. Also it can be observed that as the p value=density increased, the size of the maximum independent set decreased as it meant tendency to have more edges between the vertices. Based on these observations it can be determined that the brute force algorithm for the independent set problem, works effectively for graphs that have smaller sizes but starts having failures as the graphs get larger and more complex. Future trials may include optimizing the algorithm or switching to heuristic methods.

5.2 Heuristic Algorithm

```python
# Random graph generator
def generate_random_graph(n, p):
    graph = {i: [] for i in range(n)}  # Initialize adjacency list
    for i in range(n):
        for j in range(i + 1, n):  # Check all pairs (i, j)
            if random.random() <= p:  # Add edge with probability p
                graph[i].append(j)
                graph[j].append(i)
```

```
    return graph


# Greedy heuristic for finding an independent set
def greedy_independent_set(graph):
    independent_set = set()  # Initialize an empty set
    while graph:  # Continue until the graph is empty
        # Select vertex with the fewest neighbors
        vertex = min(graph, key=lambda v: len(graph[v]))
         independent_set.add(vertex)  # Add the vertex to the independent
set

        # Remove the vertex and its neighbors from the graph
        neighbors = graph[vertex]
        for neighbor in neighbors:
            if neighbor in graph:
                del graph[neighbor]
        del graph[vertex]
    return independent_set


def main():
    # Generate a random graph using parameters from Section 4
    n = int(input("Enter the number of vertices (n): "))
    p = float(input("Enter the edge probability (p between 0 and 1): "))
    random_graph = generate_random_graph(n, p)
    print("Generated Random Graph (Adjacency List):", random_graph)

    # Find an independent set using the heuristic algorithm
    independent_set = greedy_independent_set(random_graph)
           print("Independent  Set  Found  by  Heuristic  Algorithm:",
independent_set)


if __name__ == "__main__":
    main()
```

Sample Tests:

```
Enter the number of vertices (n): 10
```

```
Enter the edge probability (p between 0 and 1): 0.4
Generated Random Graph (Adjacency List): {0: [1, 2, 4, 6, 7], 1: [0, 5, 6, 9], 2: [0], 3: [4, 7, 9], 4: [0, 3, 5, 8], 5: [1, 4], 6: [0, 1, 9], 7: [0, 3], 8: [4], 9: [1, 3, 6]}
Independent Set Found by Heuristic Algorithm: {2, 5, 6, 7, 8}
Enter the number of vertices (n): 8
Enter the edge probability (p between 0 and 1): 0.3
Generated Random Graph (Adjacency List): {0: [], 1: [5], 2: [5], 3: [], 4: [6, 7], 5: [1, 2, 7], 6: [4], 7: [4, 5]}
Independent Set Found by Heuristic Algorithm: {0, 1, 2, 3, 6, 7}
Enter the number of vertices (n): 12
Enter the edge probability (p between 0 and 1): 0.6
Generated Random Graph (Adjacency List): {0: [1, 2, 3, 4, 5, 6, 7, 8, 11], 1: [0, 4, 5, 6, 9], 2: [0, 4, 7], 3: [0, 5, 6, 7, 8, 9, 10, 11], 4: [0, 1, 2, 5, 8, 9, 10, 11], 5: [0, 1, 3, 4, 6, 7, 8, 9, 10, 11], 6: [0, 1, 3, 5, 8, 10, 11], 7: [0, 2, 3, 5, 9, 10, 11], 8: [0, 3, 4, 5, 6, 9, 10, 11], 9: [1, 3, 4, 5, 7, 8, 10, 11], 10: [3, 4, 5, 6, 7, 8, 9], 11: [0, 3, 4, 5, 6, 7, 8, 9]}
Independent Set Found by Heuristic Algorithm: {11, 1, 2, 10}
Enter the number of vertices (n): 7
Enter the edge probability (p between 0 and 1): 0.6
Generated Random Graph (Adjacency List): {0: [1, 2, 3, 5, 6], 1: [0, 4, 5, 6], 2: [0, 3, 4, 5], 3: [0, 2, 4, 5], 4: [1, 2, 3], 5: [0, 1, 2, 3, 6], 6: [0, 1, 5]}
Independent Set Found by Heuristic Algorithm: {4, 6}
Enter the number of vertices (n): 13
Enter the edge probability (p between 0 and 1): 0.7
Generated Random Graph (Adjacency List): {0: [1, 3, 4, 5, 6, 8, 9, 10, 12], 1: [0, 2, 4, 5, 6, 7, 9, 11, 12], 2: [1, 3, 4, 6, 7, 11, 12], 3: [0, 2, 4, 5, 6, 7, 10, 11, 12], 4: [0, 1, 2, 3, 5, 7, 8, 9, 10, 11], 5: [0, 1, 3, 4, 6, 7, 8, 10, 12], 6: [0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 12], 7: [1, 2, 3, 4, 5, 6, 11, 12], 8: [0, 4, 5, 6, 9, 10, 12], 9: [0, 1, 4, 6, 8, 10, 11, 12], 10: [0, 3, 4, 5, 6, 8, 9, 11], 11: [1, 2, 3, 4, 6, 7, 9, 10, 12], 12: [0, 1, 2, 3, 5, 6, 7, 8, 9, 11]}
Independent Set Found by Heuristic Algorithm: {8, 2}
Enter the number of vertices (n): 12
```

```
Enter the edge probability (p between 0 and 1): 0.2
Generated Random Graph (Adjacency List): {0: [], 1: [6, 8], 2: [3, 10], 3: [2], 4: [], 5: [8], 6: [1], 7: [], 8: [1, 5, 9], 9: [8, 10], 10: [2, 9], 11: []}
Independent Set Found by Heuristic Algorithm: {0, 3, 4, 5, 6, 7, 9, 11}
Enter the number of vertices (n): 12
Enter the edge probability (p between 0 and 1): 0.7
Generated Random Graph (Adjacency List): {0: [4, 5, 6, 7, 8, 10, 11], 1: [2, 5, 6, 7, 8, 11], 2: [1, 5, 6, 8, 9, 11], 3: [7, 9, 10, 11], 4: [0, 6, 7, 9, 10], 5: [0, 1, 2, 6, 7, 8, 9, 11], 6: [0, 1, 2, 4, 5, 7, 9, 10, 11], 7: [0, 1, 3, 4, 5, 6, 8, 10, 11], 8: [0, 1, 2, 5, 7, 9, 10, 11], 9: [2, 3, 4, 5, 6, 8, 10], 10: [0, 3, 4, 6, 7, 8, 9, 11], 11: [0, 1, 2, 3, 5, 6, 7, 8, 10]}
Independent Set Found by Heuristic Algorithm: {1, 3, 4}
Enter the number of vertices (n): 14
Enter the edge probability (p between 0 and 1): 0.4
Generated Random Graph (Adjacency List): {0: [1, 2, 3, 4, 5, 9, 13], 1: [0, 3, 4, 7, 10, 11], 2: [0, 3, 5, 8, 9, 11, 12, 13], 3: [0, 1, 2, 4, 11], 4: [0, 1, 3, 6, 10], 5: [0, 2, 7, 8, 9, 13], 6: [4, 13], 7: [1, 5], 8: [2, 5], 9: [0, 2, 5], 10: [1, 4, 11], 11: [1, 2, 3, 10], 12: [2], 13: [0, 2, 5, 6]}
Independent Set Found by Heuristic Algorithm: {3, 6, 7, 8, 9, 10, 12}
Enter the number of vertices (n): 16
Enter the edge probability (p between 0 and 1): 0.8
Generated Random Graph (Adjacency List): {0: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15], 1: [0, 2, 3, 5, 7, 8, 9, 12, 13, 14, 15], 2: [0, 1, 3, 4, 6, 8, 9, 11, 13, 14, 15], 3: [0, 1, 2, 5, 6, 7, 8, 9, 10, 12, 14, 15], 4: [0, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], 5: [0, 1, 3, 4, 6, 7, 8, 10, 11, 12, 14, 15], 6: [0, 2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15], 7: [0, 1, 3, 4, 5, 6, 9, 10, 11, 13, 14, 15], 8: [0, 1, 2, 3, 4, 5, 6, 12, 13, 14], 9: [0, 1, 2, 3, 4, 7, 10, 13, 14, 15], 10: [0, 3, 4, 5, 6, 7, 9, 11, 12, 13, 14, 15], 11: [0, 2, 4, 5, 6, 7, 10, 13, 14], 12: [0, 1, 3, 4, 5, 6, 8, 10, 13, 14, 15], 13: [1, 2, 4, 6, 7, 8, 9, 10, 11, 12, 14, 15], 14: [0, 1, 2, 3, 4, 5, 6, 7, 8,
```

9, 10, 11, 12, 13, 15], 15: [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 12, 13, 14]}

Independent Set Found by Heuristic Algorithm: {8, 9, 11}

Enter the number of vertices (n): 16

Enter the edge probability (p between 0 and 1): 0.1

Generated Random Graph (Adjacency List): {0: [4], 1: [13], 2: [4], 3: [4, 5, 8, 13], 4: [0, 2, 3], 5: [3], 6: [7, 14], 7: [6, 14], 8: [3, 11], 9: [14], 10: [11, 13], 11: [8, 10, 15], 12: [], 13: [1, 3, 10], 14: [6, 7, 9], 15: [11]}

Independent Set Found by Heuristic Algorithm: {0, 1, 2, 5, 6, 8, 9, 10, 12, 15}

Enter the number of vertices (n): 13

Enter the edge probability (p between 0 and 1): 0.1

Generated Random Graph (Adjacency List): {0: [2, 5, 12], 1: [7], 2: [0], 3: [], 4: [10], 5: [0], 6: [9], 7: [1, 10, 12], 8: [], 9: [6], 10: [4, 7, 12], 11: [], 12: [0, 7, 10]}

Independent Set Found by Heuristic Algorithm: {1, 2, 3, 4, 5, 6, 8, 11, 12}

Enter the number of vertices (n): 17

Enter the edge probability (p between 0 and 1): 0.8

Generated Random Graph (Adjacency List): {0: [1, 3, 4, 5, 6, 8, 10, 12, 15, 16], 1: [0, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16], 2: [4, 5, 6, 7, 9, 10, 12, 14, 15, 16], 3: [0, 1, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15], 4: [0, 1, 2, 6, 7, 9, 10, 12, 14, 15, 16], 5: [0, 1, 2, 3, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16], 6: [0, 1, 2, 3, 4, 5, 8, 11, 12, 13, 15, 16], 7: [1, 2, 3, 4, 9, 10, 11, 12, 13, 14, 15, 16], 8: [0, 1, 3, 5, 6, 9, 10, 12, 13, 14, 16], 9: [1, 2, 3, 4, 5, 7, 8, 10, 13, 15, 16], 10: [0, 1, 2, 3, 4, 5, 7, 8, 9, 11, 12, 13, 14, 16], 11: [1, 5, 6, 7, 10, 12, 14, 15, 16], 12: [0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13, 14, 16], 13: [1, 3, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16], 14: [2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 15, 16], 15: [0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 13, 14, 16], 16: [0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]}

Independent Set Found by Heuristic Algorithm: {0, 2, 11, 13}

Enter the number of vertices (n): 35

```
Enter the edge probability (p between 0 and 1): 0.3
Generated Random Graph (Adjacency List): {0: [1, 2, 7, 10, 11, 12, 14, 15, 21, 22, 26], 1: [0, 2, 4, 7, 12, 13, 19, 25, 26, 28, 29, 30, 31, 33], 2: [0, 1, 4, 7, 9, 13, 14, 15, 17, 24, 26, 27, 29, 33], 3: [6, 7, 10, 13, 23, 25, 26, 29, 32, 33, 34], 4: [1, 2, 5, 6, 10, 11, 13, 14, 16, 24, 30, 34], 5: [4, 10, 12, 13, 27, 28, 30, 34], 6: [3, 4, 7, 10, 11, 12, 16, 19, 20, 22, 26, 29, 30, 31, 34], 7: [0, 1, 2, 3, 6, 8, 9, 12, 14, 15, 18, 21, 29, 31], 8: [7, 10, 11, 14, 18, 23, 27, 28, 30], 9: [2, 7, 12, 13, 15, 27, 32, 33, 34], 10: [0, 3, 4, 5, 6, 8, 15, 16, 17, 20, 29], 11: [0, 4, 6, 8, 13, 19, 26, 31, 32], 12: [0, 1, 5, 6, 7, 9, 14, 15, 16, 17, 21, 24, 31, 32], 13: [1, 2, 3, 4, 5, 9, 11, 14, 16, 18, 20, 32, 33], 14: [0, 2, 4, 7, 8, 12, 13, 17, 24, 25, 26, 27, 34], 15: [0, 2, 7, 9, 10, 12, 17, 20, 21, 22, 24, 30, 31], 16: [4, 6, 10, 12, 13, 18, 19, 22, 23, 24, 27, 29, 31], 17: [2, 10, 12, 14, 15, 19, 25, 28, 30, 33, 34], 18: [7, 8, 13, 16, 28, 33], 19: [1, 6, 11, 16, 17, 22, 23, 26, 29, 31], 20: [6, 10, 13, 15, 26, 27, 32, 33, 34], 21: [0, 7, 12, 15, 22, 24, 26, 27, 29, 33], 22: [0, 6, 15, 16, 19, 21, 25, 27, 32, 33], 23: [3, 8, 16, 19, 25, 26, 27, 32], 24: [2, 4, 12, 14, 15, 16, 21, 29], 25: [1, 3, 14, 17, 22, 23, 28, 31, 33], 26: [0, 1, 2, 3, 6, 11, 14, 19, 20, 21, 23, 29, 31, 32], 27: [2, 5, 8, 9, 14, 16, 20, 21, 22, 23, 29], 28: [1, 5, 8, 17, 18, 25, 33, 34], 29: [1, 2, 3, 6, 7, 10, 16, 19, 21, 24, 26, 27, 30], 30: [1, 4, 5, 6, 8, 15, 17, 29], 31: [1, 6, 7, 11, 12, 15, 16, 19, 25, 26, 32], 32: [3, 9, 11, 12, 13, 20, 22, 23, 26, 31, 33], 33: [1, 2, 3, 9, 13, 17, 18, 20, 21, 22, 25, 28, 32, 34], 34: [3, 4, 5, 6, 9, 14, 17, 20, 28, 33]}
Independent Set Found by Heuristic Algorithm: {1, 5, 9, 11, 17, 18, 20, 22, 23, 24}
Enter the number of vertices (n): 50
Enter the edge probability (p between 0 and 1): 0.4
Generated Random Graph (Adjacency List): {0: [3, 5, 8, 10, 12, 13, 15, 16, 21, 23, 27, 30, 32, 33, 35, 39, 43, 44, 47], 1: [2, 7, 8, 10, 12, 23, 24, 27, 28, 29, 30, 36, 40, 41, 42, 43, 45], 2: [1, 4, 5, 6, 9, 11, 13, 17, 18, 19, 20, 21, 23, 24, 27, 29, 30, 33, 36, 39, 40, 41, 44], 3: [0, 6, 7, 9, 10, 13, 17, 19, 20, 21, 30, 33, 34, 35, 38, 39, 41, 42, 45, 49], 4: [2, 6, 9, 13, 18, 20, 25, 27, 28, 29, 30, 31, 34, 36, 37, 38, 39, 40, 41, 44, 45], 5: [0, 2, 7, 8, 10, 11, 13, 14, 17,
```

20, 21, 22, 24, 28, 29, 30, 36, 44, 45, 46, 47, 49], 6: [2, 3, 4, 7, 8, 9, 12, 13, 14, 15, 16, 19, 23, 26, 27, 28, 29, 32, 33, 35, 40, 42, 49], 7: [1, 3, 5, 6, 8, 12, 13, 14, 18, 19, 24, 25, 27, 28, 33, 36, 40, 42, 43, 49], 8: [0, 1, 5, 6, 7, 10, 18, 20, 22, 23, 24, 27, 29, 33, 34, 38, 39, 42, 44, 45, 46, 48], 9: [2, 3, 4, 6, 14, 15, 16, 17, 20, 21, 24, 25, 26, 30, 33, 34, 41, 47, 49], 10: [0, 1, 3, 5, 8, 14, 20, 24, 27, 28, 29, 31, 32, 37, 41, 46, 47], 11: [2, 5, 17, 18, 19, 23, 24, 26, 27, 29, 30, 31, 32, 34, 36, 37, 45, 48], 12: [0, 1, 6, 7, 13, 16, 17, 19, 21, 27, 28, 36, 38, 41, 44, 45, 46, 47, 48, 49], 13: [0, 2, 3, 4, 5, 6, 7, 12, 17, 18, 19, 22, 24, 31, 33, 36, 38, 39, 43, 44, 45, 46, 47, 48], 14: [5, 6, 7, 9, 10, 16, 17, 18, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 39, 40, 41, 42, 44, 47, 48], 15: [0, 6, 9, 18, 19, 20, 21, 23, 24, 26, 29, 31, 33, 34, 35, 36, 42, 45, 47, 48], 16: [0, 6, 9, 12, 14, 21, 22, 23, 26, 27, 30, 31, 32, 36, 38, 39, 40, 45, 46, 47], 17: [2, 3, 5, 9, 11, 12, 13, 14, 25, 29, 30, 31, 32, 36, 37, 38, 40, 43, 48], 18: [2, 4, 7, 8, 11, 13, 14, 15, 20, 21, 24, 25, 31, 33, 38, 39, 40, 41, 42, 43, 45, 48, 49], 19: [2, 3, 6, 7, 11, 12, 13, 15, 21, 25, 35, 37, 38, 39, 40, 45, 47, 48], 20: [2, 3, 4, 5, 8, 9, 10, 15, 18, 22, 23, 27, 29, 30, 32, 33, 36, 37, 39, 41, 44, 46, 47, 48], 21: [0, 2, 3, 5, 9, 12, 15, 16, 18, 19, 26, 28, 32, 36, 38, 44, 45, 47, 49], 22: [5, 8, 13, 16, 20, 23, 26, 27, 29, 30, 33, 34, 35, 43, 44, 48, 49], 23: [0, 1, 2, 6, 8, 11, 14, 15, 16, 20, 22, 26, 27, 30, 31, 32, 36, 37, 48], 24: [1, 2, 5, 7, 8, 9, 10, 11, 13, 15, 18, 28, 30, 35, 39, 43, 45], 25: [4, 7, 9, 14, 17, 18, 19, 29, 30, 33, 34, 36, 40, 44], 26: [6, 9, 11, 14, 15, 16, 21, 22, 23, 27, 33, 35, 37, 38, 39, 40, 44, 48], 27: [0, 1, 2, 4, 6, 7, 8, 10, 11, 12, 14, 16, 20, 22, 23, 26, 28, 29, 30, 37, 39, 41, 43, 44, 46, 47, 49], 28: [1, 4, 5, 6, 7, 10, 12, 14, 21, 24, 27, 33, 34, 35, 36, 37, 40, 42, 44, 46, 48], 29: [1, 2, 4, 5, 6, 8, 10, 11, 14, 15, 17, 20, 22, 25, 27, 33, 34, 43, 45, 47], 30: [0, 1, 2, 3, 4, 5, 9, 11, 14, 16, 17, 20, 22, 23, 24, 25, 27, 32, 33, 37, 39, 42, 44, 47], 31: [4, 10, 11, 13, 14, 15, 16, 17, 18, 23, 32, 46, 49], 32: [0, 6, 10, 11, 14, 16, 17, 20, 21, 23, 30, 31, 33, 34, 35, 36, 38, 47], 33: [0, 2, 3, 6, 7, 8, 9, 13, 14, 15, 18, 20, 22, 25, 26, 28, 29, 30, 32, 38, 40, 42, 43, 44, 45, 49], 34: [3, 4, 8, 9, 11, 14, 15, 22, 25, 28, 29, 32, 37, 41, 42, 43, 46], 35: [0, 3, 6, 14, 15, 19, 22, 24, 26, 28, 32, 36, 38, 42, 45, 47,

48], 36: [1, 2, 4, 5, 7, 11, 12, 13, 15, 16, 17, 20, 21, 23, 25, 28, 32, 35, 44, 46, 47, 49], 37: [4, 10, 11, 17, 19, 20, 23, 26, 27, 28, 30, 34, 38, 40, 47], 38: [3, 4, 8, 12, 13, 16, 17, 18, 19, 21, 26, 32, 33, 35, 37, 46, 47, 48], 39: [0, 2, 3, 4, 8, 13, 14, 16, 18, 19, 20, 24, 26, 27, 30, 45], 40: [1, 2, 4, 6, 7, 14, 16, 17, 18, 19, 25, 26, 28, 33, 37, 42, 46, 47], 41: [1, 2, 3, 4, 9, 10, 12, 14, 18, 20, 27, 34, 42, 44, 46], 42: [1, 3, 6, 7, 8, 14, 15, 18, 28, 30, 33, 34, 35, 40, 41, 43, 45, 47], 43: [0, 1, 7, 13, 17, 18, 22, 24, 27, 29, 33, 34, 42, 44, 46, 47, 49], 44: [0, 2, 4, 5, 8, 12, 13, 14, 20, 21, 22, 25, 26, 27, 28, 30, 33, 36, 41, 43, 47], 45: [1, 3, 4, 5, 8, 11, 12, 13, 15, 16, 18, 19, 21, 24, 29, 33, 35, 39, 42, 48], 46: [5, 8, 10, 12, 13, 16, 20, 27, 28, 31, 34, 36, 38, 40, 41, 43, 47, 48], 47: [0, 5, 9, 10, 12, 13, 14, 15, 16, 19, 20, 21, 27, 29, 30, 32, 35, 36, 37, 38, 40, 42, 43, 44, 46, 48], 48: [8, 11, 12, 13, 14, 15, 17, 18, 19, 20, 22, 23, 26, 28, 35, 38, 45, 46, 47], 49: [3, 5, 6, 7, 9, 12, 18, 21, 22, 27, 31, 33, 36, 43]}

Independent Set Found by Heuristic Algorithm: {37, 6, 39, 41, 21, 22, 25, 31}

Enter the number of vertices (n): 55

Enter the edge probability (p between 0 and 1): 0.7

Generated Random Graph (Adjacency List): {0: [2, 3, 5, 6, 7, 8, 9, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 29, 30, 32, 35, 38, 39, 40, 41, 42, 43, 44, 46, 48, 49, 50, 52], 1: [2, 3, 4, 6, 8, 13, 16, 17, 19, 20, 21, 22, 24, 25, 26, 27, 30, 32, 36, 38, 40, 41, 45, 48, 49, 50, 54], 2: [0, 1, 3, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 21, 22, 23, 24, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 37, 39, 40, 42, 45, 47, 48, 49, 51, 54], 3: [0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 15, 16, 17, 18, 19, 21, 23, 24, 25, 26, 28, 29, 31, 32, 34, 36, 37, 38, 39, 40, 42, 44, 45, 46, 47, 48, 49, 52, 53, 54], 4: [1, 3, 6, 7, 8, 10, 11, 12, 13, 14, 16, 19, 20, 21, 23, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 39, 40, 41, 42, 43, 46, 48, 49, 50, 51, 52, 53, 54], 5: [0, 3, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 22, 23, 24, 26, 27, 29, 30, 31, 33, 34, 35, 39, 40, 42, 43, 45, 46, 49, 51, 52, 53, 54], 6: [0, 1, 2, 3, 4, 5, 8, 9, 11, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 25, 27, 29, 30, 31, 32, 33, 34, 35, 37, 38, 39, 40, 42, 43, 46, 47, 48, 51, 53], 7: [0, 2, 3, 4, 5, 8, 9, 10, 11, 12, 16, 17, 20, 21,

22, 23, 24, 25, 28, 29, 30, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 47, 49, 50, 53, 54], 8: [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 41, 42, 43, 48, 49, 50, 51, 52, 54], 9: [0, 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, 14, 19, 20, 22, 23, 25, 26, 28, 29, 30, 32, 34, 35, 36, 37, 38, 39, 40, 42, 43, 44, 46, 47, 48, 52, 54], 10: [2, 3, 4, 5, 7, 8, 9, 11, 12, 16, 18, 20, 21, 22, 23, 24, 25, 27, 28, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 43, 44, 45, 47, 48, 49, 50, 51, 53, 54], 11: [3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 17, 18, 19, 23, 25, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 41, 42, 44, 45, 46, 47, 48, 49, 51, 53], 12: [4, 5, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 20, 21, 23, 25, 28, 29, 30, 34, 36, 38, 39, 42, 44, 45, 46, 47, 48, 49, 50, 51, 53], 13: [1, 2, 4, 5, 6, 9, 11, 12, 15, 16, 18, 19, 20, 21, 22, 23, 25, 27, 28, 29, 30, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 44, 46, 47, 48, 49, 50], 14: [0, 2, 4, 6, 8, 9, 11, 12, 15, 16, 17, 19, 22, 23, 24, 25, 28, 30, 32, 34, 36, 37, 38, 39, 40, 41, 43, 44, 45, 46, 47, 51, 52, 53, 54], 15: [2, 3, 5, 6, 8, 12, 13, 14, 16, 17, 19, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32, 34, 36, 37, 39, 41, 42, 43, 45, 46, 50, 52, 53, 54], 16: [0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 26, 29, 30, 31, 32, 33, 34, 35, 36, 38, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50, 51, 53, 54], 17: [0, 1, 2, 3, 5, 6, 7, 8, 11, 12, 14, 15, 16, 18, 20, 22, 23, 24, 26, 27, 29, 30, 31, 32, 34, 36, 38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54], 18: [0, 2, 3, 5, 6, 8, 10, 11, 12, 13, 16, 17, 19, 22, 23, 27, 29, 32, 33, 36, 38, 39, 41, 42, 43, 44, 46, 47, 48, 49, 50, 51, 52, 54], 19: [0, 1, 3, 4, 8, 9, 11, 13, 14, 15, 16, 18, 21, 22, 23, 24, 25, 27, 28, 29, 30, 32, 33, 35, 36, 37, 38, 40, 41, 42, 43, 45, 47, 49, 50, 52, 54], 20: [0, 1, 4, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 21, 24, 25, 26, 27, 30, 31, 33, 35, 36, 38, 40, 41, 43, 45, 46, 47, 48, 49, 51, 52, 54], 21: [0, 1, 2, 3, 4, 6, 7, 8, 10, 12, 13, 15, 16, 19, 20, 22, 23, 25, 26, 27, 28, 29, 32, 33, 35, 36, 37, 38, 39, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54], 22: [0, 1, 2, 5, 6, 7, 8, 9, 10, 13, 14, 16, 17, 18, 19, 21, 23, 24, 25, 31, 32, 34, 36, 37, 38, 39, 42, 43, 44, 46, 48, 49, 50, 52, 53], 23: [0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 24, 25, 32, 34, 35, 36, 37, 39, 41, 42, 43, 44, 45, 49,

50, 51, 52, 53, 54], 24: [0, 1, 2, 3, 5, 7, 10, 14, 15, 17, 19, 20, 22, 23, 26, 27, 28, 29, 30, 32, 34, 35, 36, 37, 40, 42, 43, 45, 46, 47, 48, 50, 51, 52, 53], 25: [1, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 19, 20, 21, 22, 23, 26, 27, 28, 29, 31, 34, 35, 38, 39, 42, 43, 44, 45, 46, 47, 49, 51, 53, 54], 26: [0, 1, 2, 3, 4, 5, 8, 9, 15, 16, 17, 20, 21, 24, 25, 28, 29, 31, 33, 34, 35, 38, 39, 41, 44, 46, 49, 50, 51, 52, 53, 54], 27: [0, 1, 2, 4, 5, 6, 8, 10, 13, 15, 17, 18, 19, 20, 21, 24, 25, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 42, 43, 44, 45, 46, 48, 49, 51, 52], 28: [2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15, 19, 21, 24, 25, 26, 29, 30, 31, 32, 33, 35, 36, 37, 38, 39, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 54], 29: [0, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 16, 17, 18, 19, 21, 24, 25, 26, 28, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40, 42, 43, 44, 45, 47, 48, 49, 50, 52, 53, 54], 30: [0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 24, 27, 28, 29, 33, 34, 35, 36, 37, 38, 40, 41, 45, 46, 47, 48, 49, 51, 54], 31: [2, 3, 4, 5, 6, 10, 11, 15, 16, 17, 20, 22, 25, 26, 27, 28, 29, 33, 34, 35, 36, 38, 39, 40, 41, 42, 44, 45, 46, 48, 49, 51, 52, 53, 54], 32: [0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 27, 28, 29, 33, 34, 36, 37, 38, 39, 41, 42, 43, 44, 46, 48, 49, 50, 51, 52, 53, 54], 33: [2, 4, 5, 6, 8, 10, 11, 13, 16, 18, 19, 20, 21, 26, 27, 28, 30, 31, 32, 34, 36, 37, 38, 39, 40, 42, 43, 44, 47, 49, 50, 51, 52, 53, 54], 34: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 22, 23, 24, 25, 26, 27, 29, 30, 31, 32, 33, 38, 40, 41, 42, 44, 47, 48, 50, 52, 53, 54], 35: [0, 2, 5, 6, 7, 8, 9, 10, 11, 13, 16, 19, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 36, 37, 38, 39, 40, 43, 47, 48, 50, 51, 54], 36: [1, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 27, 28, 29, 30, 31, 32, 33, 35, 37, 38, 39, 40, 42, 43, 44, 45, 46, 47, 48, 49, 52], 37: [2, 3, 6, 7, 8, 9, 10, 11, 13, 14, 15, 19, 21, 22, 23, 24, 27, 28, 29, 30, 32, 33, 35, 36, 38, 39, 41, 42, 45, 47, 48, 49, 50, 52, 53, 54], 38: [0, 1, 3, 6, 7, 9, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 41, 43, 45, 46, 47, 48, 49, 50, 53, 54], 39: [0, 2, 3, 4, 5, 6, 7, 9, 10, 12, 13, 14, 15, 17, 18, 21, 22, 23, 25, 26, 27, 28, 29, 31, 32, 33, 35, 36, 37, 38, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 54], 40: [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 13, 14, 16, 19, 20,

24, 27, 29, 30, 31, 33, 34, 35, 36, 39, 45, 48, 49, 50, 51, 52, 53], 41: [0, 1, 4, 7, 8, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 23, 26, 30, 31, 32, 34, 37, 38, 39, 42, 44, 45, 48, 49, 50, 51, 52, 53, 54], 42: [0, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18, 19, 22, 23, 24, 25, 27, 28, 29, 31, 32, 33, 34, 36, 37, 39, 41, 45, 48, 49, 50, 51, 52, 54], 43: [0, 4, 5, 6, 7, 8, 9, 10, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 32, 33, 35, 36, 38, 39, 45, 46, 47, 48, 49, 50, 51, 52, 54], 44: [0, 3, 7, 9, 10, 11, 12, 13, 14, 16, 17, 18, 21, 22, 23, 25, 26, 27, 28, 29, 31, 32, 33, 34, 36, 39, 41, 45, 46, 47, 49, 52, 54], 45: [1, 2, 3, 5, 7, 10, 11, 12, 14, 15, 16, 17, 19, 20, 21, 23, 24, 25, 27, 28, 29, 30, 31, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46, 47, 48, 50, 51, 52, 53], 46: [0, 3, 4, 5, 6, 9, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30, 31, 32, 36, 38, 39, 43, 44, 45, 47, 48, 49, 50, 51, 53, 54], 47: [2, 3, 6, 7, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 24, 25, 28, 29, 30, 33, 34, 35, 36, 37, 38, 43, 44, 45, 46, 48, 50, 51, 53, 54], 48: [0, 1, 2, 3, 4, 6, 8, 9, 10, 11, 12, 13, 16, 17, 18, 20, 21, 22, 24, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 45, 46, 47, 50, 51, 52, 53, 54], 49: [0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 29, 30, 31, 32, 33, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46, 50, 51, 52, 53], 50: [0, 1, 4, 7, 8, 10, 12, 13, 15, 16, 17, 18, 19, 21, 22, 23, 24, 26, 28, 29, 32, 33, 34, 35, 37, 38, 39, 40, 41, 42, 43, 45, 46, 47, 48, 49, 51, 53], 51: [2, 4, 5, 6, 8, 10, 11, 12, 14, 16, 17, 18, 20, 21, 23, 24, 25, 26, 27, 28, 30, 31, 32, 33, 35, 39, 40, 41, 42, 43, 45, 46, 47, 48, 49, 50, 52], 52: [0, 3, 4, 5, 8, 9, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 28, 29, 31, 32, 33, 34, 36, 37, 39, 40, 41, 42, 43, 44, 45, 48, 49, 51, 53, 54], 53: [3, 4, 5, 6, 7, 10, 11, 12, 14, 15, 16, 17, 21, 22, 23, 24, 25, 26, 28, 29, 31, 32, 33, 34, 37, 38, 40, 41, 45, 46, 47, 48, 49, 50, 52], 54: [1, 2, 3, 4, 5, 7, 8, 9, 10, 14, 15, 16, 17, 18, 19, 20, 21, 23, 25, 26, 28, 29, 30, 31, 32, 33, 34, 35, 37, 38, 39, 41, 42, 43, 44, 46, 47, 48, 52]}

Independent Set Found by Heuristic Algorithm: {1, 35, 44, 15}

Enter the number of vertices (n): 75

Enter the edge probability (p between 0 and 1): 0.3

Generated Random Graph (Adjacency List): {0: [3, 4, 5, 6, 7, 9, 15, 21, 23, 25, 44, 47, 55, 56, 58, 61, 62, 64, 66, 67, 69, 71, 72, 74], 1: [2, 3, 4, 9, 12, 14, 21, 28, 31, 36, 40, 43, 44, 47, 51, 52, 53, 57, 59, 64, 67, 73, 74], 2: [1, 4, 5, 6, 7, 8, 11, 16, 20, 21, 23, 25, 26, 27, 28, 30, 31, 32, 40, 41, 42, 46, 53, 54, 56, 58, 60, 61, 62, 63, 65, 70, 71, 72], 3: [0, 1, 28, 30, 33, 34, 35, 37, 41, 45, 46, 47, 49, 53, 54, 58, 60, 61, 71, 72], 4: [0, 1, 2, 5, 7, 14, 17, 22, 24, 38, 44, 49, 51, 52, 54, 55, 71, 74], 5: [0, 2, 4, 6, 8, 11, 13, 14, 21, 24, 27, 36, 37, 44, 45, 48, 56, 58, 62, 63, 67, 68, 72], 6: [0, 2, 5, 14, 15, 19, 25, 29, 34, 35, 37, 38, 39, 40, 41, 42, 43, 46, 49, 57, 58, 61, 62, 63, 71, 72, 74], 7: [0, 2, 4, 11, 13, 14, 18, 20, 22, 24, 31, 34, 37, 39, 40, 41, 42, 43, 44, 55, 59, 60, 66, 68, 70, 71], 8: [2, 5, 10, 14, 19, 20, 23, 27, 28, 32, 35, 38, 40, 44, 48, 51, 53, 55, 68], 9: [0, 1, 10, 11, 14, 15, 17, 18, 19, 21, 22, 23, 24, 27, 36, 37, 38, 44, 46, 60, 70, 74], 10: [8, 9, 12, 19, 21, 22, 26, 29, 31, 33, 36, 42, 45, 52, 57, 59, 60, 63, 67, 70], 11: [2, 5, 7, 9, 14, 15, 26, 32, 36, 37, 44, 46, 47, 48, 50, 51, 57, 60, 62, 68, 72], 12: [1, 10, 15, 18, 28, 35, 36, 38, 39, 40, 42, 47, 51, 53, 62, 65, 66, 73, 74], 13: [5, 7, 17, 23, 26, 45, 46, 47, 60, 61, 62, 66, 70, 71], 14: [1, 4, 5, 6, 7, 8, 9, 11, 15, 17, 18, 19, 21, 23, 27, 35, 39, 44, 48, 53, 54, 55, 56, 57, 60, 61, 66, 69, 72], 15: [0, 6, 9, 11, 12, 14, 20, 23, 33, 35, 36, 41, 43, 49, 50, 56, 58, 60, 63, 65, 66, 69, 71, 72], 16: [2, 17, 19, 20, 21, 24, 26, 31, 34, 45, 53, 63], 17: [4, 9, 13, 14, 16, 22, 24, 26, 27, 32, 44, 46, 47, 51, 52, 59, 67, 70], 18: [7, 9, 12, 14, 19, 21, 22, 23, 25, 28, 29, 34, 38, 41, 45, 49, 51, 53, 64, 67, 68, 72], 19: [6, 8, 9, 10, 14, 16, 18, 20, 21, 23, 24, 32, 41, 43, 50, 51, 53, 55, 61, 64, 72, 73], 20: [2, 7, 8, 15, 16, 19, 21, 25, 28, 31, 33, 36, 38, 59, 60, 64, 65, 67, 69, 73, 74], 21: [0, 1, 2, 5, 9, 10, 14, 16, 18, 19, 20, 29, 30, 33, 38, 39, 44, 50, 51, 52, 57, 58, 61, 66, 70, 71], 22: [4, 7, 9, 10, 17, 18, 23, 25, 26, 27, 32, 35, 40, 45, 46, 47, 49, 56, 62, 65, 70, 72], 23: [0, 2, 8, 9, 13, 14, 15, 18, 19, 22, 26, 32, 35, 36, 39, 41, 46, 47, 48, 51, 53, 58, 64, 65, 66, 67, 70, 71, 73, 74], 24: [4, 5, 7, 9, 16, 17, 19, 26, 30, 35, 36, 37, 40, 42, 43, 45, 47, 50, 54, 55, 58, 61, 64, 65, 66, 71], 25: [0, 2, 6, 18, 20, 22, 26, 28, 30, 35, 38, 39, 42, 43, 44, 45, 49, 50, 54, 60, 70], 26: [2, 10, 11, 13, 16, 17, 22, 23, 24, 25, 27, 29, 30, 35, 39, 42,

45, 48, 49, 50, 51, 55, 56, 58, 64, 65, 68, 71, 72, 73], 27: [2, 5, 8, 9, 14, 17, 22, 26, 29, 33, 39, 40, 41, 51, 52, 54, 57, 64, 66, 67, 73], 28: [1, 2, 3, 8, 12, 18, 20, 25, 29, 31, 32, 33, 36, 37, 40, 41, 42, 44, 45, 48, 49, 50, 52, 53, 58, 62, 63, 69, 70, 71], 29: [6, 10, 18, 21, 26, 27, 28, 30, 36, 42, 55, 57, 59, 65, 68, 72], 30: [2, 3, 21, 24, 25, 26, 29, 33, 34, 39, 40, 42, 47, 49, 50, 52, 55, 56, 57, 62, 65, 68, 69, 70, 71, 72, 73], 31: [1, 2, 7, 10, 16, 20, 28, 32, 40, 43, 47, 51, 53, 54, 56, 59, 61, 70, 71, 74], 32: [2, 8, 11, 17, 19, 22, 23, 28, 31, 38, 39, 40, 42, 48, 53, 54, 60, 61, 68, 69], 33: [3, 10, 15, 20, 21, 27, 28, 30, 35, 37, 38, 39, 40, 42, 44, 45, 46, 51, 52, 55, 58, 59, 60, 61, 65, 67, 73], 34: [3, 6, 7, 16, 18, 30, 37, 38, 45, 47, 49, 50, 52, 56, 59, 69, 70, 74], 35: [3, 6, 8, 12, 14, 15, 22, 23, 24, 25, 26, 33, 39, 40, 44, 47, 50, 52, 55, 59, 71, 73], 36: [1, 5, 9, 10, 11, 12, 15, 20, 23, 24, 28, 29, 41, 42, 45, 47, 65, 66, 70], 37: [3, 5, 6, 7, 9, 11, 24, 28, 33, 34, 39, 40, 42, 45, 48, 50, 60, 66, 67, 69, 70], 38: [4, 6, 8, 9, 12, 18, 20, 21, 25, 32, 33, 34, 42, 45, 51, 52, 54, 55, 56, 57, 58, 63, 64], 39: [6, 7, 12, 14, 21, 23, 25, 26, 27, 30, 32, 33, 35, 37, 41, 47, 48, 58, 62, 63, 70, 72, 73], 40: [1, 2, 6, 7, 8, 12, 22, 24, 27, 28, 30, 31, 32, 33, 35, 37, 42, 45, 47, 53, 54, 56, 57, 58, 59, 60, 63, 64, 65, 66, 70, 71, 74], 41: [2, 3, 6, 7, 15, 18, 19, 23, 27, 28, 36, 39, 43, 50, 52, 53, 54, 56, 61, 62, 63, 64, 65, 68, 74], 42: [2, 6, 7, 10, 12, 24, 25, 26, 28, 29, 30, 32, 33, 36, 37, 38, 40, 47, 49, 50, 52, 58, 59, 63, 64, 65, 66, 69, 70], 43: [1, 6, 7, 15, 19, 24, 25, 31, 41, 48, 50, 51, 52, 56, 57, 61, 63, 66, 67, 71, 74], 44: [0, 1, 4, 5, 7, 8, 9, 11, 14, 17, 21, 25, 28, 33, 35, 48, 49, 51, 64, 68, 71], 45: [3, 5, 10, 13, 16, 18, 22, 24, 25, 26, 28, 33, 34, 36, 37, 38, 40, 46, 49, 51, 56, 61, 65, 68, 71], 46: [2, 3, 6, 9, 11, 13, 17, 22, 23, 33, 45, 53, 54, 60, 62, 63, 67, 69, 70, 72], 47: [0, 1, 3, 11, 12, 13, 17, 22, 23, 24, 30, 31, 34, 35, 36, 39, 40, 42, 49, 52, 58, 67, 68, 72, 74], 48: [5, 8, 11, 14, 23, 26, 28, 32, 37, 39, 43, 44, 49, 53, 62, 64, 67, 68, 72, 74], 49: [3, 4, 6, 15, 18, 22, 25, 26, 28, 30, 34, 42, 44, 45, 47, 48, 52, 53, 55, 57, 59, 64, 66, 67, 71, 72, 74], 50: [11, 15, 19, 21, 24, 25, 26, 28, 30, 34, 35, 37, 41, 42, 43, 51, 53, 56, 62, 64, 69, 70], 51: [1, 4, 8, 11, 12, 17, 18, 19, 21, 23, 26, 27, 31, 33, 38, 43, 44, 45, 50, 58, 68, 70, 71, 73, 74], 52: [1, 4, 10, 17, 21, 27, 28, 30, 33, 34,

35, 38, 41, 42, 43, 47, 49, 53, 54, 55, 58, 59, 62, 67, 69, 70, 73], 53: [1, 2, 3, 8, 12, 14, 16, 18, 19, 23, 28, 31, 32, 40, 41, 46, 48, 49, 50, 52, 68, 71, 72], 54: [2, 3, 4, 14, 24, 25, 27, 31, 32, 38, 40, 41, 46, 52, 57, 58, 61, 63, 65, 69, 70], 55: [0, 4, 7, 8, 14, 19, 24, 26, 29, 30, 33, 35, 38, 49, 52, 57, 61, 65, 67, 69, 70, 71, 74], 56: [0, 2, 5, 14, 15, 22, 26, 30, 31, 34, 38, 40, 41, 43, 45, 50, 60, 62, 64, 66, 70], 57: [1, 6, 10, 11, 14, 21, 27, 29, 30, 38, 40, 43, 49, 54, 55, 60, 61, 63, 64, 67, 71, 72, 73], 58: [0, 2, 3, 5, 6, 15, 21, 23, 24, 26, 28, 33, 38, 39, 40, 42, 47, 51, 52, 54, 61, 62, 65, 73], 59: [1, 7, 10, 17, 20, 29, 31, 33, 34, 35, 40, 42, 49, 52, 60, 70], 60: [2, 3, 7, 9, 10, 11, 13, 14, 15, 20, 25, 32, 33, 37, 40, 46, 56, 57, 59, 61, 62, 66, 68, 71], 61: [0, 2, 3, 6, 13, 14, 19, 21, 24, 31, 32, 33, 41, 43, 45, 54, 55, 57, 58, 60, 62, 65, 68], 62: [0, 2, 5, 6, 11, 12, 13, 22, 28, 30, 39, 41, 46, 48, 50, 52, 56, 58, 60, 61, 63, 64, 68, 70, 73], 63: [2, 5, 6, 10, 15, 16, 28, 38, 39, 40, 41, 42, 43, 46, 54, 57, 62, 65, 70, 71, 72, 73, 74], 64: [0, 1, 18, 19, 20, 23, 24, 26, 27, 38, 40, 41, 42, 44, 48, 49, 50, 56, 57, 62, 66, 68, 70, 73], 65: [2, 12, 15, 20, 22, 23, 24, 26, 29, 30, 33, 36, 40, 41, 42, 45, 54, 55, 58, 61, 63, 67, 70, 71, 73, 74], 66: [0, 7, 12, 13, 14, 15, 21, 23, 24, 27, 36, 37, 40, 42, 43, 49, 56, 60, 64, 67, 69, 71], 67: [0, 1, 5, 10, 17, 18, 20, 23, 27, 33, 37, 43, 46, 47, 48, 49, 52, 55, 57, 65, 66, 70, 74], 68: [5, 7, 8, 11, 18, 26, 29, 30, 32, 41, 44, 45, 47, 48, 51, 53, 60, 61, 62, 64], 69: [0, 14, 15, 20, 28, 30, 32, 34, 37, 42, 46, 50, 52, 54, 55, 66, 70, 71, 73], 70: [2, 7, 9, 10, 13, 17, 21, 22, 23, 25, 28, 30, 31, 34, 36, 37, 39, 40, 42, 46, 50, 51, 52, 54, 55, 56, 59, 62, 63, 64, 65, 67, 69, 71, 72], 71: [0, 2, 3, 4, 6, 7, 13, 15, 21, 23, 24, 26, 28, 30, 31, 35, 40, 43, 44, 45, 49, 51, 53, 55, 57, 60, 63, 65, 66, 69, 70], 72: [0, 2, 3, 5, 6, 11, 14, 15, 18, 19, 22, 26, 29, 30, 39, 46, 47, 48, 49, 53, 57, 63, 70], 73: [1, 12, 19, 20, 23, 26, 27, 30, 33, 35, 39, 51, 52, 57, 58, 62, 63, 64, 65, 69], 74: [0, 1, 4, 6, 9, 12, 20, 23, 31, 34, 40, 41, 43, 47, 48, 49, 51, 55, 63, 65, 67]}

Independent Set Found by Heuristic Algorithm: {3, 4, 69, 67, 8, 11, 12, 13, 16, 56, 25, 29}

Enter the number of vertices (n): 100

Enter the edge probability (p between 0 and 1): 0.2

Generated Random Graph (Adjacency List): {0: [4, 7, 9, 10, 16, 21, 44, 50, 52, 59, 67, 68, 75, 79, 80, 87, 91, 96], 1: [4, 16, 25, 26, 38, 43, 48, 50, 61, 63, 67, 83, 89, 92], 2: [5, 7, 13, 20, 21, 26, 30, 35, 43, 45, 46, 49, 58, 66, 68, 76, 85, 90, 91, 94, 95, 96, 97, 99], 3: [4, 5, 13, 15, 16, 17, 20, 25, 42, 44, 47, 48, 50, 57, 58, 61, 66, 71, 75, 80, 84, 85, 90, 95], 4: [0, 1, 3, 7, 8, 9, 14, 18, 22, 25, 30, 31, 37, 38, 42, 54, 61, 62, 68, 74, 75, 76, 78, 80, 82, 89, 93, 97], 5: [2, 3, 6, 13, 15, 39, 40, 44, 52, 62, 66, 67, 72, 75, 78, 81, 82, 86, 91, 93], 6: [5, 17, 19, 20, 25, 30, 34, 40, 45, 57, 59, 61, 63, 65, 69, 72, 75, 87, 93, 96, 99], 7: [0, 2, 4, 19, 22, 28, 33, 34, 40, 44, 56, 57, 68, 84, 89], 8: [4, 20, 22, 24, 32, 37, 38, 39, 44, 48, 51, 52, 53, 58, 66, 67, 70, 82, 87, 89, 92], 9: [0, 4, 23, 24, 27, 30, 32, 40, 56, 57, 64, 68, 71, 81, 82, 85, 88, 90, 95, 99], 10: [0, 12, 13, 16, 19, 25, 26, 27, 32, 35, 37, 44, 45, 46, 47, 50, 67, 68, 71, 72, 89], 11: [19, 30, 32, 52, 54, 58, 62, 65, 72, 80, 84, 88, 92, 93, 96, 97], 12: [10, 15, 17, 19, 26, 34, 35, 36, 37, 40, 41, 42, 47, 54, 57, 70, 78, 96, 98], 13: [2, 3, 5, 10, 16, 26, 32, 36, 44, 50, 58, 66, 69, 71, 81, 84, 88, 96, 99], 14: [4, 38, 53, 66, 67, 68, 70, 73, 90, 93], 15: [3, 5, 12, 21, 26, 30, 31, 34, 39, 42, 58, 65, 75, 80, 86, 95, 98], 16: [0, 1, 3, 10, 13, 27, 29, 30, 37, 39, 43, 54, 55, 56, 65, 66, 67, 73, 74, 75, 82, 86, 89], 17: [3, 6, 12, 25, 27, 28, 31, 39, 45, 47, 48, 61, 62, 64, 68, 72, 73, 92, 98], 18: [4, 30, 32, 37, 38, 47, 51, 57, 59, 64, 67, 71, 73, 80, 86, 87, 97], 19: [6, 7, 10, 11, 12, 25, 31, 42, 47, 48, 50, 56, 58, 61, 64, 65, 85, 93], 20: [2, 3, 6, 8, 28, 34, 44, 46, 50, 63, 64, 66, 79, 80, 82, 86, 89, 93], 21: [0, 2, 15, 22, 29, 31, 36, 42, 43, 44, 45, 50, 54, 59, 67, 78, 83, 99], 22: [4, 7, 8, 21, 23, 28, 35, 36, 43, 50, 53, 54, 56, 58, 66, 70, 84, 85, 92, 96, 98], 23: [9, 22, 25, 31, 32, 33, 42, 46, 47, 51, 55, 61, 63, 74, 77, 78, 80, 81, 84, 92], 24: [8, 9, 25, 42, 43, 44, 45, 49, 53, 59, 60, 61, 62, 63, 66, 68, 69, 71, 73, 80, 88, 90, 94, 97], 25: [1, 3, 4, 6, 10, 17, 19, 23, 24, 26, 36, 37, 38, 40, 49, 59, 62, 65, 68, 69, 73, 74, 75, 76, 82, 88, 89, 90, 91, 94], 26: [1, 2, 10, 12, 13, 15, 25, 30, 34, 40, 41, 45, 48, 49, 51, 52, 57, 65, 69, 72, 81, 85, 91, 96, 99], 27: [9, 10, 16, 17, 30, 34, 36, 38, 48, 51, 53, 54, 58, 60, 68, 73, 77, 82, 90, 94, 98], 28: [7, 17, 20, 22, 37, 42, 46, 55, 65, 66, 68, 79, 82, 83, 91, 93, 94, 95], 29: [16, 21, 30, 37, 40, 45,

69, 78, 84, 87, 89, 91, 92, 96], 30: [2, 4, 6, 9, 11, 15, 16, 18, 26, 27, 29, 37, 38, 48, 55, 61, 63, 68, 74, 77, 79, 80, 82, 85, 91, 92, 95], 31: [4, 15, 17, 19, 21, 23, 39, 47, 51, 53, 75, 77, 83, 86, 90, 93, 98], 32: [8, 9, 10, 11, 13, 18, 23, 45, 51, 61, 70, 75, 81, 86, 90, 92], 33: [7, 23, 36, 37, 46, 47, 50, 52, 57, 60, 83, 89, 94, 96, 97], 34: [6, 7, 12, 15, 20, 26, 27, 37, 42, 46, 47, 50, 54, 62, 71, 72, 75, 77, 83, 86, 88, 89, 96], 35: [2, 10, 12, 22, 41, 46, 54, 59, 64, 67, 69, 71, 74, 79, 80, 88, 98], 36: [12, 13, 21, 22, 25, 27, 33, 38, 39, 42, 44, 52, 61, 62, 67, 70, 73, 74, 76, 78, 89, 90, 92, 94, 97], 37: [4, 8, 10, 12, 16, 18, 25, 28, 29, 30, 33, 34, 40, 48, 51, 52, 53, 59, 62, 65, 70, 73, 80, 81], 38: [1, 4, 8, 14, 18, 25, 27, 30, 36, 53, 61, 72, 75, 86, 93], 39: [5, 8, 15, 16, 17, 31, 36, 41, 50, 53, 58, 68, 69, 76, 77, 80, 81, 82, 84, 88, 98], 40: [5, 6, 7, 9, 12, 25, 26, 29, 37, 45, 46, 53, 58, 61, 68, 69, 70, 74, 78, 81, 85, 86, 89], 41: [12, 26, 35, 39, 43, 56, 58, 59, 65, 68, 69, 95], 42: [3, 4, 12, 15, 19, 21, 23, 24, 28, 34, 36, 53, 56, 58, 60, 68, 71, 91, 93, 94, 97], 43: [1, 2, 16, 21, 22, 24, 41, 45, 46, 50, 57, 58, 60, 79, 90, 92, 98, 99], 44: [0, 3, 5, 7, 8, 10, 13, 20, 21, 24, 36, 51, 55, 58, 71, 78, 86, 88, 89, 90, 91, 93, 96, 99], 45: [2, 6, 10, 17, 21, 24, 26, 29, 32, 40, 43, 47, 59, 63, 67, 74, 77, 79, 94, 96], 46: [2, 10, 20, 23, 28, 33, 34, 35, 40, 43, 48, 65, 66, 82, 86, 87, 96], 47: [3, 10, 12, 17, 18, 19, 23, 31, 33, 34, 45, 50, 66, 75, 76, 81, 91, 93, 95], 48: [1, 3, 8, 17, 19, 26, 27, 30, 37, 46, 51, 59, 60, 68, 69, 76, 79, 90, 91, 93, 94, 97], 49: [2, 24, 25, 26, 50, 67, 68, 73, 74, 76, 83, 85, 87, 90, 93, 94], 50: [0, 1, 3, 10, 13, 19, 20, 21, 22, 33, 34, 39, 43, 47, 49, 51, 52, 56, 58, 59, 64, 65, 74, 83, 84, 88, 91, 96], 51: [8, 18, 23, 26, 27, 31, 32, 37, 44, 48, 50, 52, 53, 59, 62, 63, 66, 67, 68, 70, 75, 82, 88, 92], 52: [0, 5, 8, 11, 26, 33, 36, 37, 50, 51, 55, 66, 67, 76, 79, 80, 81, 82, 84, 90, 94, 97], 53: [8, 14, 22, 24, 27, 31, 37, 38, 39, 40, 42, 51, 54, 56, 57, 60, 69, 77, 82, 88, 95], 54: [4, 11, 12, 16, 21, 22, 27, 34, 35, 53, 59, 65, 72, 73, 85, 88, 94, 97, 98], 55: [16, 23, 28, 30, 44, 52, 60, 79, 82, 84, 86, 88, 89], 56: [7, 9, 16, 19, 22, 41, 42, 50, 53, 60, 65, 67, 73, 81, 85, 86, 91, 99], 57: [3, 6, 7, 9, 12, 18, 26, 33, 43, 53, 62, 70, 71, 73, 81, 82, 86, 88, 91, 94], 58: [2, 3, 8, 11, 13, 15, 19, 22, 27, 39, 40, 41, 42, 43, 44, 50, 61, 62, 67, 70, 71, 73, 79, 82, 83, 87, 93,

98], 59: [0, 6, 18, 21, 24, 25, 35, 37, 41, 45, 48, 50, 51, 54, 64, 65, 69, 70, 79, 89], 60: [24, 27, 33, 42, 43, 48, 53, 55, 56, 63, 64, 68, 74, 78, 82, 88, 99], 61: [1, 3, 4, 6, 17, 19, 23, 24, 30, 32, 36, 38, 40, 58, 62, 67, 82, 87, 94, 95, 98], 62: [4, 5, 11, 17, 24, 25, 34, 36, 37, 51, 57, 58, 61, 69, 82, 83, 86, 88], 63: [1, 6, 20, 23, 24, 30, 45, 51, 60, 64, 65, 70, 79, 81, 83, 88], 64: [9, 17, 18, 19, 20, 35, 50, 59, 60, 63, 67, 71, 77, 78, 86, 88], 65: [6, 11, 15, 16, 19, 25, 26, 28, 37, 41, 46, 50, 54, 56, 59, 63, 68, 72, 73, 78, 79, 80, 97], 66: [2, 3, 5, 8, 13, 14, 16, 20, 22, 24, 28, 46, 47, 51, 52, 69, 70, 71, 72, 73, 75, 76, 81, 83, 90, 98], 67: [0, 1, 5, 8, 10, 14, 16, 18, 21, 35, 36, 45, 49, 51, 52, 56, 58, 61, 64, 68, 71, 76, 81, 89, 98], 68: [0, 2, 4, 7, 9, 10, 14, 17, 24, 25, 27, 28, 30, 39, 40, 41, 42, 48, 49, 51, 60, 65, 67, 72, 77, 88, 89, 91, 92, 93, 99], 69: [6, 13, 24, 25, 26, 29, 35, 39, 40, 41, 48, 53, 59, 62, 66, 75, 84, 86, 96], 70: [8, 12, 14, 22, 32, 36, 37, 40, 51, 57, 58, 59, 63, 66, 71, 76, 78, 81, 86, 89, 97], 71: [3, 9, 10, 13, 18, 24, 34, 35, 42, 44, 57, 58, 64, 66, 67, 70, 76, 77, 79, 80, 81, 82, 91, 94], 72: [5, 6, 10, 11, 17, 26, 34, 38, 54, 65, 66, 68, 82, 87, 88, 89, 90, 94, 95], 73: [14, 16, 17, 18, 24, 25, 27, 36, 37, 49, 54, 56, 57, 58, 65, 66, 75, 78, 82, 83, 89, 91], 74: [4, 16, 23, 25, 30, 35, 36, 40, 45, 49, 50, 60, 87, 92], 75: [0, 3, 4, 5, 6, 15, 16, 25, 31, 32, 34, 38, 47, 51, 66, 69, 73, 76, 78, 80, 81, 85, 87, 89, 93, 94], 76: [2, 4, 25, 36, 39, 47, 48, 49, 52, 66, 67, 70, 71, 75, 83, 98], 77: [23, 27, 30, 31, 34, 39, 45, 53, 64, 68, 71, 79, 84, 91, 96, 99], 78: [4, 5, 12, 21, 23, 29, 36, 40, 44, 60, 64, 65, 70, 73, 75, 85, 90, 92], 79: [0, 20, 28, 30, 35, 43, 45, 48, 52, 55, 58, 59, 63, 65, 71, 77, 83, 92, 96, 97, 99], 80: [0, 3, 4, 11, 15, 18, 20, 23, 24, 30, 35, 37, 39, 52, 65, 71, 75, 81, 86, 93, 94], 81: [5, 9, 13, 23, 26, 32, 37, 39, 40, 47, 52, 56, 57, 63, 66, 67, 70, 71, 75, 80, 90, 92, 93, 95], 82: [4, 5, 8, 9, 16, 20, 25, 27, 28, 30, 39, 46, 51, 52, 53, 55, 57, 58, 60, 61, 62, 71, 72, 73, 84, 94, 96, 99], 83: [1, 21, 28, 31, 33, 34, 49, 50, 58, 62, 63, 66, 73, 76, 79, 84, 89, 92, 95, 96, 98], 84: [3, 7, 11, 13, 22, 23, 29, 39, 50, 52, 55, 69, 77, 82, 83, 90, 93], 85: [2, 3, 9, 19, 22, 26, 30, 40, 49, 54, 56, 75, 78, 86, 89, 92, 96, 97, 99], 86: [5, 15, 16, 18, 20, 31, 32, 34, 38, 40, 44, 46, 55, 56, 57, 62, 64, 69, 70, 80, 85, 94, 98], 87: [0, 6, 8, 18, 29, 46, 49, 58, 61,

```
72, 74, 75], 88: [9, 11, 13, 24, 25, 34, 35, 39, 44, 50, 51, 53, 54,
55, 57, 60, 62, 63, 64, 68, 72, 90], 89: [1, 4, 7, 8, 10, 16, 20, 25,
29, 33, 34, 36, 40, 44, 55, 59, 67, 68, 70, 72, 73, 75, 83, 85, 98],
90: [2, 3, 9, 14, 24, 25, 27, 31, 32, 36, 43, 44, 48, 49, 52, 66, 72,
78, 81, 84, 88, 94, 99], 91: [0, 2, 5, 25, 26, 28, 29, 30, 42, 44, 47,
48, 50, 56, 57, 68, 71, 73, 77, 95, 98], 92: [1, 8, 11, 17, 22, 23,
29, 30, 32, 36, 43, 51, 68, 74, 78, 79, 81, 83, 85, 96, 99], 93: [4,
5, 6, 11, 14, 19, 20, 28, 31, 38, 42, 44, 47, 48, 49, 58, 68, 75, 80,
81, 84, 98, 99], 94: [2, 24, 25, 27, 28, 33, 36, 42, 45, 48, 49, 52,
54, 57, 61, 71, 72, 75, 80, 82, 86, 90], 95: [2, 3, 9, 15, 28, 30, 41,
47, 53, 61, 72, 81, 83, 91], 96: [0, 2, 6, 11, 12, 13, 22, 26, 29, 33,
34, 44, 45, 46, 50, 69, 77, 79, 82, 83, 85, 92, 97], 97: [2, 4, 11,
18, 24, 33, 36, 42, 48, 52, 54, 65, 70, 79, 85, 96], 98: [12, 15, 17,
22, 27, 31, 35, 39, 43, 54, 58, 61, 66, 67, 76, 83, 86, 89, 91, 93],
99: [2, 6, 9, 13, 21, 26, 43, 44, 56, 60, 68, 77, 79, 82, 85, 90, 92,
93]}
Independent Set Found by Heuristic Algorithm: {64, 1, 99, 37, 7, 41,
91, 11, 76, 45, 14, 15, 87, 55, 27}
```

6. Experimental Analysis of The Performance (Performance Testing)

```python
import time
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import t

# Heuristic algorithm
def greedy_independent_set(graph):
    independent_set = set()
    while graph:
        vertex = min(graph, key=lambda v: len(graph[v]))
        independent_set.add(vertex)
        neighbors = graph[vertex]
        for neighbor in neighbors:
            if neighbor in graph:
                del graph[neighbor]
        del graph[vertex]
```

```
    return independent_set

# Measure runtime
def measure_runtime(algorithm, graph, runs=30, repeat=100):
    times = []
    for _ in range(runs):
        start_time = time.perf_counter()
        for _ in range(repeat):
            algorithm(graph.copy())
        end_time = time.perf_counter()
          times.append((end_time - start_time) / repeat)   # Average
per repetition
    return np.mean(times), np.std(times)

# Confidence interval
def compute_confidence_interval(mean, std, n, confidence=0.9):
    t_value = t.ppf((1 + confidence) / 2, df=n-1)
    margin_of_error = t_value * (std / np.sqrt(n))
    return mean - margin_of_error, mean + margin_of_error

# Run experiments
def run_experiments():
     sizes = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120,130, 140,
150, 160, 170, 180, 190, 200, 220, 230, 240, 250, 260, 270 ]   #
Larger sizes
    p = 0.5
    runs = 30
    repeat = 50

    means = []
    margins = []
    for n in sizes:
        print(f"Running for input size: {n}")
        graph = generate_random_graph(n, p)
                                          mean_time,     std_time     =
measure_runtime(greedy_independent_set, graph, runs, repeat)
        ci = compute_confidence_interval(mean_time, std_time, runs)
        means.append(mean_time)
        margins.append(ci[1] - mean_time)  # Margin of error
        print(f"n={n}, Mean Time={mean_time:.6f}, CI={ci}")
```

```
    # Plot runtime with confidence intervals
            plt.errorbar(sizes,   means,   yerr=margins,   fmt='o-',
label='Runtime with CI')
    plt.xlabel('Input Size (n)')
    plt.ylabel('Average Runtime (seconds)')
    plt.title('Runtime Analysis with Confidence Intervals')
    plt.legend()
    plt.grid()
    plt.show()

if __name__ == "__main__":
    run_experiments()
```

```
Running for input size: 10
n=10,       Mean       Time=0.000006,       CI=(5.110069464440051e-06,
6.959217202564486e-06)
Running for input size: 20
n=20,       Mean       Time=0.000013,       CI=(1.098373049794972e-05,
1.5398910836495884e-05)
Running for input size: 30
n=30,       Mean       Time=0.000020,       CI=(1.8226049213378115e-05,
2.088743611736676e-05)
Running for input size: 40
n=40,       Mean       Time=0.000021,       CI=(1.9978680094661947e-05,
2.204152924026457e-05)
Running for input size: 50
n=50,       Mean       Time=0.000030,       CI=(2.7557977675047396e-05,
3.151003031989055e-05)
Running for input size: 60
n=60,       Mean       Time=0.000033,       CI=(3.1499497821692694e-05,
3.407283284948556e-05)
Running for input size: 70
n=70,       Mean       Time=0.000063,       CI=(5.329590518916379e-05,
7.19385294708753e-05)
Running for input size: 80
```

n=80, Mean Time=0.000055, CI=(5.1140354498799315e-05, 5.914172283881589e-05)

Running for input size: 90

n=90, Mean Time=0.000049, CI=(4.598278622291787e-05, 5.14358804439602e-05)

Running for input size: 100

n=100, Mean Time=0.000064, CI=(5.8519893308993434e-05, 6.884651735758849e-05)

Running for input size: 120

n=120, Mean Time=0.000062, CI=(5.9755549689635863e-05, 6.446603697995561e-05)

Running for input size: 130

n=130, Mean Time=0.000087, CI=(8.044425564571989e-05, 9.260229235625276e-05)

Running for input size: 140

n=140, Mean Time=0.000090, CI=(8.395713196285927e-05, 9.695379603615842e-05)

Running for input size: 150

n=150, Mean Time=0.000093, CI=(8.983016483929794e-05, 9.648651782798919e-05)

Running for input size: 160

n=160, Mean Time=0.000099, CI=(9.284882283508702e-05, 0.00010145317638340821)

Running for input size: 170

n=170, Mean Time=0.000107, CI=(0.00010300025507578991, 0.00011146017959400379)

Running for input size: 180

n=180, Mean Time=0.000106, CI=(0.00010261747118996131, 0.00010840344614583544)

Running for input size: 190

n=190, Mean Time=0.000106, CI=(0.00010320648680940884, 0.00010914478385815941)

Running for input size: 200

n=200, Mean Time=0.000119, CI=(0.00010777975791468946, 0.00012986423408612292)

Running for input size: 220

n=220, Mean Time=0.000131, CI=(0.0001262909845206149 7, 0.0001363221114797594 8)

Running for input size: 230

n=230, Mean Time=0.000138, CI=(0.0001308608907221895, 0.0001443870159435648)

Running for input size: 240

n=240, Mean Time=0.000140, CI=(0.0001365284911786671 7, 0.0001432395621580711 8)

Running for input size: 250

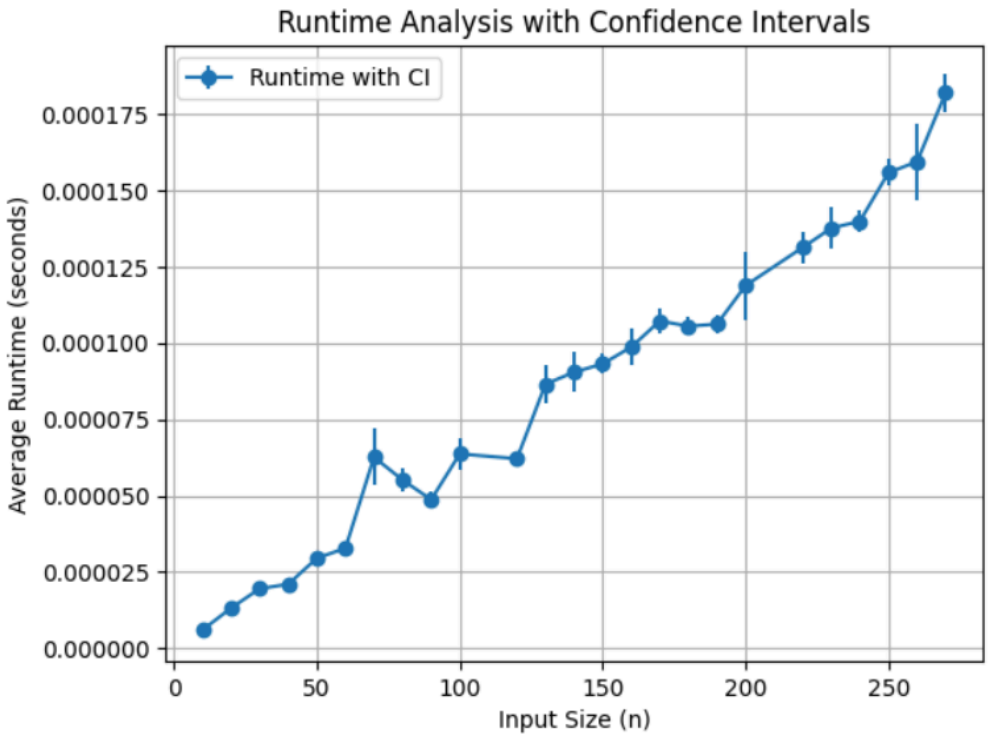
n=250, Mean Time=0.000156, CI=(0.0001514334024519637 6, 0.0001601637135485209 5)

Running for input size: 260

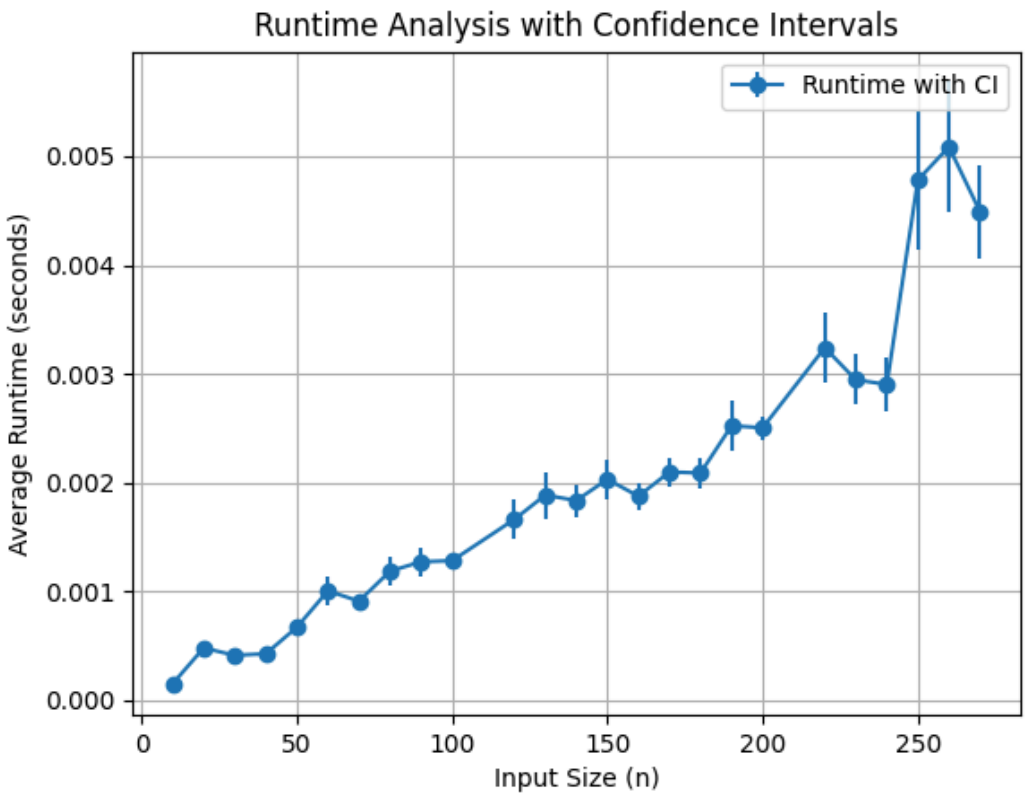
n=260, Mean Time=0.000159, CI=(0.0001469069244604654, 0.0001718603222048609 6)

Running for input size: 270

n=270, Mean Time=0.000182, CI=(0.0001756591228382556 7, 0.0001881966384896609 5)



(made running the code on Google Colab)



(made by running the code on Visual Studio)

The runtime analysis, accompanied by the data and the graph, demonstrates the algorithm's performance is correlated with input size. For smaller inputs (e.g., n≤50n \leq 50n≤50), runtimes are minimal and consistent. As the input size increases, the average runtime grows steadily, reflecting the algorithm's increased computational demands. Notably, the confidence intervals (CIs) increase with larger inputs, particularly beyond n=200n = 200n=200, indicating greater variability in runtime performance. This pattern suggests that while the algorithm maintains manageable runtimes for moderate-sized inputs, its efficiency becomes less predictable as input size approaches the higher limits of testing. The findings agree with theoretical expectations, showing the connection between input size and execution time.

7. Experimental Analysis of the Quality

```
n=5, p=0.2
Brute Force Size: 4, Heuristic Size: 4
```

```
Relative Error: 0.00%
Brute Force Time: 0.0001s, Heuristic Time: 0.0000s
========================================
n=5, p=0.5
Brute Force Size: 3, Heuristic Size: 3
Relative Error: 0.00%
Brute Force Time: 0.0001s, Heuristic Time: 0.0000s
========================================
n=5, p=0.8
Brute Force Size: 2, Heuristic Size: 2
Relative Error: 0.00%
Brute Force Time: 0.0000s, Heuristic Time: 0.0000s
========================================
n=8, p=0.2
Brute Force Size: 5, Heuristic Size: 5
Relative Error: 0.00%
Brute Force Time: 0.0003s, Heuristic Time: 0.0000s
========================================
n=8, p=0.5
Brute Force Size: 3, Heuristic Size: 3
Relative Error: 0.00%
Brute Force Time: 0.0003s, Heuristic Time: 0.0000s
========================================
n=8, p=0.8
Brute Force Size: 3, Heuristic Size: 2
Relative Error: -33.33%
Brute Force Time: 0.0002s, Heuristic Time: 0.0000s
========================================
n=10, p=0.2
Brute Force Size: 5, Heuristic Size: 5
Relative Error: 0.00%
Brute Force Time: 0.0012s, Heuristic Time: 0.0000s
========================================
n=10, p=0.5
Brute Force Size: 4, Heuristic Size: 2
Relative Error: -50.00%
```

```
Brute Force Time: 0.0010s, Heuristic Time: 0.0000s
========================================
n=10, p=0.8
Brute Force Size: 2, Heuristic Size: 2
Relative Error: 0.00%
Brute Force Time: 0.0014s, Heuristic Time: 0.0000s
========================================
n=12, p=0.2
Brute Force Size: 6, Heuristic Size: 6
Relative Error: 0.00%
Brute Force Time: 0.0072s, Heuristic Time: 0.0001s
========================================
n=12, p=0.5
Brute Force Size: 4, Heuristic Size: 4
Relative Error: 0.00%
Brute Force Time: 0.0037s, Heuristic Time: 0.0000s
========================================
n=12, p=0.8
Brute Force Size: 3, Heuristic Size: 3
Relative Error: 0.00%
Brute Force Time: 0.0036s, Heuristic Time: 0.0000s
========================================
n=16, p=0.2
Brute Force Size: 8, Heuristic Size: 8
Relative Error: 0.00%
Brute Force Time: 0.1097s, Heuristic Time: 0.0001s
========================================
n=16, p=0.5
Brute Force Size: 5, Heuristic Size: 4
Relative Error: -20.00%
Brute Force Time: 0.0859s, Heuristic Time: 0.0006s
========================================
n=16, p=0.8
Brute Force Size: 3, Heuristic Size: 2
Relative Error: -33.33%
Brute Force Time: 0.0483s, Heuristic Time: 0.0000s
```

```
========================================
n=18, p=0.2
Brute Force Size: 8, Heuristic Size: 7
Relative Error: -12.50%
Brute Force Time: 0.4607s, Heuristic Time: 0.0000s
========================================
n=18, p=0.5
Brute Force Size: 5, Heuristic Size: 3
Relative Error: -40.00%
Brute Force Time: 0.2932s, Heuristic Time: 0.0000s
========================================
n=18, p=0.8
Brute Force Size: 4, Heuristic Size: 3
Relative Error: -25.00%
Brute Force Time: 0.2247s, Heuristic Time: 0.0001s
========================================
n=20, p=0.2
Brute Force Size: 10, Heuristic Size: 8
Relative Error: -20.00%
Brute Force Time: 1.5678s, Heuristic Time: 0.0001s
========================================
n=20, p=0.5
Brute Force Size: 6, Heuristic Size: 4
Relative Error: -33.33%
Brute Force Time: 0.8032s, Heuristic Time: 0.0000s
========================================
n=20, p=0.8
Brute Force Size: 3, Heuristic Size: 3
Relative Error: 0.00%
Brute Force Time: 0.4951s, Heuristic Time: 0.0000s
========================================
n=22, p=0.2
Brute Force Size: 10, Heuristic Size: 8
Relative Error: -20.00%
Brute Force Time: 4.2232s, Heuristic Time: 0.0000s
========================================
```

```
n=22, p=0.5
Brute Force Size: 6, Heuristic Size: 6
Relative Error: 0.00%
Brute Force Time: 2.1176s, Heuristic Time: 0.0000s
========================================
n=22, p=0.8
Brute Force Size: 3, Heuristic Size: 3
Relative Error: 0.00%
Brute Force Time: 1.8421s, Heuristic Time: 0.0000s
========================================
n=24, p=0.2
Brute Force Size: 10, Heuristic Size: 8
Relative Error: -20.00%
Brute Force Time: 17.4810s, Heuristic Time: 0.0001s
========================================
n=24, p=0.5
Brute Force Size: 6, Heuristic Size: 6
Relative Error: 0.00%
Brute Force Time: 13.6428s, Heuristic Time: 0.0000s
========================================
n=24, p=0.8
Brute Force Size: 4, Heuristic Size: 2
Relative Error: -50.00%
Brute Force Time: 10.3744s, Heuristic Time: 0.0001s
========================================
```

Based on the experimental testing, it can be said that the relative error is very small meaning that the heuristic algorithm works very similar to the brute force algorithm in most cases considering the testing was done with small sizes of n values, with the exception that, when p is large (0.8), heuristic algorithm produces smaller independent sets leading to a large amounts of negative relative error. The error tends to increase as n values get larger as heuristic algorithms produce smaller sets than brute force algorithms. As a time convenience brute force algorithm takes considerably larger time compared to heuristic algorithms (it shows 0.000s or 0.0001s as time is too small for the code to test). Therefore it can be driven from the experimenting tests that the heuristic algorithm performs quite well especially for smaller n values and denser graphs, however it underestimates the size of the sets as n values get larger and more sparse.

8. **Experimental Analysis of the Correctness of the Implementation (Functional Testing)**

For this question, Black Box Testing was decided to be implemented because, the main requirement for the heuristic algorithm testing in this case is to return a valid independent set. Black Box Testing focuses on verifying the correct output from the algorithm, without needing to understand the internal logic aligning with the goal of checking the correctness of the algorithm. In the testing function, few test sets are used and the message "Test passed" is prompted to check the validity of the heuristic algorithm's produced sets. The consistency in the success of passed tests indicates confidence in the heuristic algorithm's working process across a range of test cases and graph structures.

```
import random
from itertools import combinations
import time

# 1. Helper function to check if the independent set is valid
def is_independent_set(graph, independent_set):
    """
      Checks whether the given independent set is valid (no two
vertices are adjacent).
    """
     for i, j in combinations(independent_set, 2):  # Check all pairs
in the set
            if j in graph[i]:   # If there is an edge between two
vertices, not independent
             return False
    return True

# 2. Function to generate a random graph (from previous questions)
# 3. Simple heuristic: Greedy selection of vertices with the least
neighbors
def heuristic_independent_set(graph):
    n = len(graph)
    independent_set = []
    vertices = list(range(n))
```

```
    for vertex in vertices:
        # Check if vertex can be added to the independent set
            if all(neighbor not in independent_set for neighbor in
graph[vertex]):
                independent_set.append(vertex)  # Add the vertex to the
independent set

    return independent_set

# 4. Functional Testing - Black Box Testing
def black_box_testing():
     # List of test cases with (n, p) pairs (n: number of vertices,
p: edge probability)
    test_cases = [
        (5, 0.2), (5, 0.5), (5, 0.8),
        (8, 0.2), (8, 0.5), (8, 0.8),
        (10, 0.2), (10, 0.5), (10, 0.8),
        (15, 0.2), (15, 0.5), (15, 0.8),
        (20, 0.2), (20, 0.5), (20, 0.8),
        (25, 0.2), (25, 0.5), (25, 0.8),
        (30, 0.2), (30, 0.5), (30, 0.8),
        (40, 0.2), (40, 0.5), (40, 0.8),
    ]

    for n, p in test_cases:
        # Generate random graph
        graph = generate_random_graph(n, p)
        print(f"Testing graph with n={n}, p={p}")

         #     Apply     your     heuristic     algorithm     (assuming
`heuristic_independent_set` is defined)
        heuristic_set = heuristic_independent_set(graph)

        # Validate if the heuristic set is independent
            assert is_independent_set(graph, heuristic_set), f"Test
failed for n={n}, p={p}! Heuristic set is not independent."

        # Print success message for this test case
```

```
            print(f"Test passed for n={n}, p={p}. Heuristic set is
independent.")

        # Print the results for verification
        print(f"Independent Set: {heuristic_set}")
        print("-" * 40)

# Run the test
print("Running Black Box Tests...")
black_box_testing()
```

```
Running Black Box Tests...
Testing graph with n=5, p=0.2
Test passed for n=5, p=0.2. Heuristic set is independent.
Independent Set: [0, 1, 3, 4]
----------------------------------------
Testing graph with n=5, p=0.5
Test passed for n=5, p=0.5. Heuristic set is independent.
Independent Set: [0]
----------------------------------------
Testing graph with n=5, p=0.8
Test passed for n=5, p=0.8. Heuristic set is independent.
Independent Set: [0]
----------------------------------------
Testing graph with n=8, p=0.2
Test passed for n=8, p=0.2. Heuristic set is independent.
Independent Set: [0, 1, 2, 3, 5]
----------------------------------------
Testing graph with n=8, p=0.5
Test passed for n=8, p=0.5. Heuristic set is independent.
Independent Set: [0, 3, 4]
----------------------------------------
Testing graph with n=8, p=0.8
Test passed for n=8, p=0.8. Heuristic set is independent.
Independent Set: [0, 1]
----------------------------------------
Testing graph with n=10, p=0.2
```

Test passed for n=10, p=0.2. Heuristic set is independent.
Independent Set: [0, 3, 5, 6, 9]

Testing graph with n=10, p=0.5
Test passed for n=10, p=0.5. Heuristic set is independent.
Independent Set: [0, 1]

Testing graph with n=10, p=0.8
Test passed for n=10, p=0.8. Heuristic set is independent.
Independent Set: [0, 5]

Testing graph with n=15, p=0.2
Test passed for n=15, p=0.2. Heuristic set is independent.
Independent Set: [0, 1, 2, 3, 6, 7, 14]

Testing graph with n=15, p=0.5
Test passed for n=15, p=0.5. Heuristic set is independent.
Independent Set: [0, 3, 5]

Testing graph with n=15, p=0.8
Test passed for n=15, p=0.8. Heuristic set is independent.
Independent Set: [0, 3]

Testing graph with n=20, p=0.2
Test passed for n=20, p=0.2. Heuristic set is independent.
Independent Set: [0, 1, 3, 6, 7, 10, 18]

Testing graph with n=20, p=0.5
Test passed for n=20, p=0.5. Heuristic set is independent.
Independent Set: [0, 1, 6, 9, 16]

Testing graph with n=20, p=0.8
Test passed for n=20, p=0.8. Heuristic set is independent.
Independent Set: [0, 7, 15]

Testing graph with n=25, p=0.2

Test passed for n=25, p=0.2. Heuristic set is independent.
Independent Set: [0, 1, 2, 6, 13, 14, 18, 23]
--
Testing graph with n=25, p=0.5
Test passed for n=25, p=0.5. Heuristic set is independent.
Independent Set: [0, 3, 8, 11, 19]
--
Testing graph with n=25, p=0.8
Test passed for n=25, p=0.8. Heuristic set is independent.
Independent Set: [0, 3, 19]
--
Testing graph with n=30, p=0.2
Test passed for n=30, p=0.2. Heuristic set is independent.
Independent Set: [0, 1, 2, 9, 12, 13, 20, 28, 29]
--
Testing graph with n=30, p=0.5
Test passed for n=30, p=0.5. Heuristic set is independent.
Independent Set: [0, 1, 2, 5, 6, 28]
--
Testing graph with n=30, p=0.8
Test passed for n=30, p=0.8. Heuristic set is independent.
Independent Set: [0, 1, 28]
--
Testing graph with n=40, p=0.2
Test passed for n=40, p=0.2. Heuristic set is independent.
Independent Set: [0, 1, 2, 8, 10, 11, 19, 36]
--
Testing graph with n=40, p=0.5
Test passed for n=40, p=0.5. Heuristic set is independent.
Independent Set: [0, 2, 3, 7, 15, 28]
--
Testing graph with n=40, p=0.8
Test passed for n=40, p=0.8. Heuristic set is independent.
Independent Set: [0, 1, 8]
--

9. Discussion

The Independent Set Problem is a challenging problem related to graph theory with several applications that can be studied and applied in the real world. This report explored two ways to handle the problem: a brute-force algorithm and a heuristic method. Each approach has unique advantages and limitations that offer valuable insights while solving the problem.

The brute-force algorithm guarantees finding the maximum independent set but its computational complexity of O(n^2 · 2^n) makes it impractical for graphs with more than a few vertices. The practical tests also aligned with this information as the program could not handle running for test cases with large vertice numbers due to excessive computation time. In contrast, the heuristic algorithm uses a greedy approach to identify an approximate solution in polynomial time. This method selects vertices iteratively based on the fewest conflicts and removes them along with their neighbors. Although it does not guarantee finding the maximum independent set, the heuristic method consistently produced valid independent sets in testing. With a time complexity of O(n^2 + m), it is well-suited for larger graphs. The practical testing also has highlighted its time and space efficiency aligning with the goal of using the heuristic method while having some error rate and not finding the maximum number of sets correctly.

For future directions; combining heuristic methods with brute force algorithms to balance efficiency and accuracy could be studied for better improvements in performance.

REFERENCES

[1] E. W. Weisstein, "Independent Set," *MathWorld--A Wolfram Web Resource*. [Online]. Available: https://mathworld.wolfram.com/IndependentSet.html. [Accessed: Nov. 14, 2024].

[2] Wikipedia, "Maximal independent set," *Wikipedia*, the free encyclopedia, Oct. 9, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Maximal_independent_set. [Accessed: Nov. 14, 2024].

[3] GeeksforGeeks, "Proof that Independent Set in Graph Theory is NP Complete," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/proof-that-independent-set-in-graph-theory-is-np-complete/ . [Accessed: Nov. 14, 2024].

[4]A. Vince, "Counting connected sets and connected partitions of a graph," *Australasian Journal of Combinatorics*, vol. 67, no. 2, pp. 281–293, 2017. [Online]. Available: https://ajc.maths.uq.edu.au/pdf/67/ajc_v67_p281.pdf. [Accessed: Nov. 14, 2024].

[5] GeeksforGeeks, "Erdős–Rényi Model for Generating Random Graphs," GeeksforGeeks, 22-Feb-2023. [Online]. Available: https://www.geeksforgeeks.org/erdos-renyl-model-generating-random-graphs/. [Accessed: 16-Nov-2024].

[6] J. van Leeuwen, D. Smit, and P. Scheer, *Algorithmic Modeling and Complexity, Fall 2002: Lecture 7 (23 September)*. Available: https://userpages.cs.umbc.edu/jekkin1/JEKKIN/MIS.pdf.

[7] D. N. Gainanov, N. Mladenović, V. Rasskazova, and D. Urošević, "Heuristic Algorithm for Finding the Maximum Independent Set with Absolute Estimate of the Accuracy," *CEUR Workshop Proceedings*, vol. 2098, pp. 1–10, 2018. Available: https://ceur-ws.org/Vol-2098/paper12.pdf.

[8] T. Aarsen, *Maximum Independent Set*, GitHub repository. [Online]. Available: https://github.com/tomaarsen/MaximumIndependentSet/tree/master. [Accessed: Nov. 16, 2024].