Exhaustive search – 1st project

Daniel-Mihai Miricel

Abstract - **This report explores the implementation and analysis of two approaches to solving the clique problem in undirected graphs: the exhaustive search algorithm and a greedy heuristic method. The clique problem, which involves finding a complete subgraph of size k within a given graph, is computationally intensive and classified as NP-complete. The exhaustive search algorithm guarantees accuracy by systematically examining all possible subsets of vertices, making it suitable for small graph instances but impractical for larger graphs due to exponential time complexity. In contrast, the greedy heuristic method incrementally constructs a clique by selecting vertices with the highest connectivity, offering computational efficiency but without guaranteed optimality. Both algorithms are evaluated on randomly generated graphs of varying sizes and densities. The analysis underscores the practical utility of heuristic methods for large-scale problems while reaffirming the exhaustive search's role in exact solutions for smaller cases.**

I. Introduction

The clique problem is a fundamental challenge in graph theory and combinatorial optimization. The problem entails identifying a complete subgraph, or clique, of a given size k within an undirected graph G(V,E). Despite its deceptively simple formulation, the problem is NP-complete [1] , meaning no polynomial-time algorithm is known to exist for solving it in the general case. This computational complexity makes the clique problem a perfect example for evaluating algorithmic strategies, particularly in balancing solution quality and computational efficiency.

This report focuses on two contrasting approaches to solving the clique problem: an exhaustive search algorithm and a greedy heuristic method. The exhaustive search approach, while computationally intensive, guarantees finding a solution by exploring all possible subsets of vertices. This makes it an ideal candidate for exact solutions on small graphs. On the other hand, the greedy heuristic incrementally builds a clique by selecting vertices with the highest degree of connectivity, offering a computationally efficient alternative for larger or more complex graphs but without guaranteeing optimality.

The algorithms are tested on randomly generated graphs with varying characteristics, using both sparse and dense edge configurations.

II. Exhaustive search algorithm

Exhaustive search is a brute-force approach that systematically explores all possible subsets of vertices to find one that satisfies the clique property. While this method guarantees finding a solution (or confirming its absence), its computational cost grows exponentially with the size of the input, making it impractical for large graphs [2].

The first step is to generate all subsets of k vertices from the graph. The total number of such subsets is given by the binomial coefficient representing the total combinations of k elements from a set of size V [3].

For each subset of vertices, the algorithm verifies whether the subset satisfies the clique property. This is done by checking every pair of vertices within the subset and ensuring that each pair is directly connected by an edge in E. If even one pair is not connected, the subset is immediately rejected as a clique. The algorithm continues this process, systematically evaluating all subsets of size k.

This process ensures a comprehensive search for a solution, providing a guarantee of correctness at the cost of significant computational complexity due to the exponential growth in the number of subsets and the pairwise checks required for each subset.

III. Greedy heuristic algorithm

The greedy heuristic algorithm is an approximate approach that incrementally constructs a clique by selecting vertices based on a locally optimal decision at each step. While this method does not guarantee finding the largest clique or even a clique of the desired size, it is computationally efficient and performs well in practice for many graph instances.

The algorithm begins by sorting all vertices in the graph based on their degree, which represents the number of edges connected to each vertex. Sorting ensures that vertices with higher connectivity are considered first, as they are more likely to form part of a clique. Once sorted, the algorithm iteratively selects vertices to add to the clique, starting with the highest-degree vertex.

For each vertex selected, the algorithm verifies whether it is adjacent to all the vertices already included in the clique. If it is, the vertex is added to the clique. If not, the vertex is skipped, and the algorithm moves to the next vertex in the sorted list. This process continues until the

desired clique size k is reached, or all vertices have been evaluated.

By focusing on high-degree vertices and checking adjacency incrementally, the greedy heuristic significantly reduces computational effort compared to exhaustive search. However, its reliance on local decisions means that it may miss larger cliques or fail to find a clique of size k even if one exists. Despite this limitation, the method provides a practical balance between efficiency and solution quality, making it suitable for large or dense graphs where exhaustive search is infeasible.

IV. Formal computational complexity analysis

This part of the report will show the formal time complexity analysis of the two algorithms. For this purpose I used the "big O notation" [4].

A. Exhaustive search

As mentioned earlier, the exhaustive search algorithm generates all subsets of k vertices, then checks every pair of vertices within the subset, ensuring a complete evaluation of subsets of size k.

The total number of operations required to generate all subsets of k vertices from the set V is given by the binomial coefficient.

$$\binom{V}{k} = \frac{V!}{k!(V-k!)}$$

For each subset of size k, the algorithm checks all combinations pairs of vertices to ensure they are connected. Each adjacency check involves $O(deg(u))$ operations, where $deg(u)$ is the degree of a vertex u.

$$\binom{k}{2} * deg(u)$$

The total time complexity will be:

$$O\left(\binom{V}{k} * \binom{k}{2} * deg(u)\right)$$

For large V and k, this grows exponentially, making the method not feasible for larger graphs.

B. Greedy heuristic

The greedy heuristic algorithm sorts all vertices and builds the clique iteratively based on adjacency.

Sorting the vertices based on their degree, using the built-in Python function, requires a complexity of $O(V \log V)$ [5].

For each vertex, the algorithm checks whether it is adjacent to all vertices already in the clique. In the worst case, this involves checking $O(k)$ vertices for $O(V)$ candidates.

Total time complexity will be:

$$O(V \log V + V*k*deg(u))$$

V. Experimental computational complexity analysis

For the purpose of the experimental computational complexity analysis, I added a global counter into each of the algorithms, counting the total number of operations. In this phase I used graphs of increasing sizes, from 4 to 25 vertices, but with a fixed density of 50%.

A. Exhaustive search

The number of operations, for graphs of increasing sizes, from 4 to 25, was 2, 60, 140, 592, 1454, 5040, 11088, 33908, 76006, 216017, 463151, 1247628, 2699460, 7001280, 14780480, 37162023, 78918858, 193993474, 405622971, 977390848, 2051778596 and 4867080379. The number of operations increases exponentially with each vertex added and so does the time necessary for running the algorithm. The total time it took to generate the graphs and run the exhaustive search algorithm was 8.4 seconds. For graph instances with densities of 12.5%, 25%, 50% and 75%, the total time was 33.2 seconds. Increasing the number of vertices past 25 resulted in the code running for too long.

In the case of the graph with 10 vertices, and a density of 50%, resulting in 22 edges, and a clique size of 5, I will compare the output of the experimental computational complexity analysis with the formal computational complexity analysis. With the experimental computational complexity analysis, the number of operations counted was 11088. With the formal computational complexity analysis we have the former formula, and the following variables:

$$V = 10, k = 5, E = 22$$

For $deg(u)$ we will approximate with an average degree of $E/V = 22/10 = 2.2$

$$O\left(\binom{10}{5} * \binom{5}{2} * 2.2\right) = 252 * 10 * 2.2 = 5544$$

A considerable difference between the two methods can be noticed. Because in the experimental analysis I calculate the degree of each vertex instead of approximating it once for the whole graph, I decided to calculate the average degree using the edges divided by

number of vertices formula, and to compute the average degree experimentally, and compare the results.

Using the formula for the average degree, we would get, for graphs of increasing number of vertices, from 4 to 25, with the density of 50%, the following approximated average degrees: 0.8, 1.0, 1.2, 1.4, 1.8, 2.0, 2.2, 2.5, 2.8, 3.0, 3.2, 3.5, 3.8, 4.0, 4.2, 4.5, 4.8, 5.0, 5.2, 5.5, 5.8, 6.0.

Computing the average degree experimentally for the same graphs, in the exhaustive search, we get the following results: 2.0, 2.0, 2.3, 2.9, 3.5, 4.0, 4.4, 4.9, 5.5, 6.0, 6.4, 6.9, 7.5, 8.0, 8.4, 8.9, 9.5, 10.0, 10.5, 11.0, 11.5, 12.0. The average degree I got to experimentally is approximately 2 times greater for each graph, compared to the one calculated using the formula. This difference can be explained by the fact that in the exhaustive search algorithm we look at k-sized subsets of graphs, avoiding more isolated vertices.

B. Greedy heuristic

The number of operations, for graphs of increasing sizes, from 4 to 25, was 13, 18, 21, 27, 30, 37, 40, 43, 42, 49, 52, 51, 63, 61, 64, 72, 70, 73, 81, 84, 93, 85. This suggests that the number of operations grows at an average rate of about 1.1 per vertex added.

Since the number of operations is low for small graphs, I tried to generate larger graph instances, and run the greedy heuristic algorithm.

For a graph with 5000 vertices, and a density of 50%, the number of edges was 6,248,750. The number of operations was 15105, and the total time was 21.6 seconds. The relatively low number of operations for the greedy algorithm suggests that most of the time cost is created by generating a graph of this size.

For a graph with 6000 vertices, and a density of 12.5%, the number of edges was 2,249,625. The number of operations was 18015, and the total time was 41.3 seconds.

Beyond this point, despite waiting and trying with larger numbers, the code was running for too long.

VI. Precision of the Greedy Heuristic

To analyze the efficiency of the greedy heuristic algorithm, I executed both the greedy and exhaustive search algorithms on the same set of graphs. The graphs were generated randomly, with a total number of vertices ranging from 4 to 25, densities of 12.5%, 25%, 50% and 75%, and the value k set to 12.5%, 25%, 50% and 75% of the number of graph vertices.

The total number of graphs generated was 352. The two algorithms generated different results 12 times, resulting in a precision of 96.6%. The 12 times the greedy heuristic algorithm gave a wrong answer were all false negatives.

References

[1] Proof that Clique Decision problem is NP-Complete, *Geeksforgeeks*. URL: https://www.geeksforgeeks.org/proof-that-clique-decision-problem-is-np-complete/

[2] Nievergelt, Jürg. "Exhaustive search, combinatorial optimization and enumeration: Exploring the potential of raw computing power." *International Conference on Current Trends in Theory and Practice of Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000

[3]Binomial coefficients. *Encyclopedia of Mathematics.* URL: https://encyclopediaofmath.org/index.php?title=Binomial_coefficients

[4]Big O notation, *Wikipedia*. URL: https://en.wikipedia.org/wiki/Big_O_notation

[5]Sort() in Python, *Geeksforgeeks*. URL: https://www.geeksforgeeks.org/sort-in-python/

.