Design of Graph Mining tool implementing Genetic Algorithm

1st Aryan Jung Shah
North Central College
Naperville, IL United States
ajshah@noctrl.edu

2nd Ramon Vilarroig
North Central College
Naperville, IL United States
rvilarroig@noctrl.edu

3rd Asal Sharma Dhungana
North Central College
Naperville, IL United States
asharmadhungana@noctrl.edu

***Abstract*—The purpose of this paper is to present the conceptual design and implementation of a graph mining tool that employs genetic algorithms to identify a minimum-cost subgraph within a network graph. This tool aims to efficiently connect a specified subset of nodes (M) while minimizing the total number of links (L) utilized. By utilizing genetic algorithms, the program evolves solutions over multiple generations, thereby optimizing the connectivity of the desired nodes. This paper discusses the implementation details of the genetic algorithm, including the strategies for fitness evaluation, selection, crossover, and mutation. Experimental results highlight the effectiveness of the proposed approach under varying parameters, demonstrating its capability to minimize costs while ensuring robust connectivity.**

Keywords—Genetic algorithms, graph mining, minimum cost subgraph, Python, optimization, network graph

I. Introduction

Genetic Algorithm is an optimization and search heuristic inspired by the principles of natural selection and genetics. It is used to solve complex problems by mimicking the process of biological evolution. In this paper, it analyzes the Graph implementing the Genetic Algorithm. Executing the Genetic Algorithm has the following procedure—initializing the population, Evaluating the Fitness of the given population, Selection, Crossover, and Mutation. The initialization process begins with a randomly generated solutions to the problem. The initialized population is then evaluated using a fitness function which measures how a particular solution performs relative to the objective goal. After the fitness of each solution is calculated the selection process chooses specific individuals for reproduction based on their fitness scores. The crossover process consists of selected individuals paired with recombined solutions to generate a new offspring. Furthermore, to maintain diversity within the population and to avoid premature convergence random mutations are introduced. In this project we implemented the genetic search algorithm for a graph based on the European backbone network. That calculates the optimal subgraph with the best fitness score.

II. Assumption

There were several factors that were taken into account before the subgraphs were implemented. We assume we have a given network graph G. With a set of nodes and links that connect each node (G = (N, L)). The graph miner considers this graph and selects a number of target nodes. It then creates a set of subgraphs of one hundred random samples. The result will try to find the optimal solution with the corresponding subgraph. We considered a text file COST239 network with nodes and the links that correspond to it. Moreover, the population size remains constant throughout the evolution process.

III. Methods

The process is initialized with a given network graph, where G = (N, L) where N is a set of nodes and L is a set of links from which a subgraph was randomly generated. This subgraph served as the starting point for the algorithm, which aimed to find an optimal or near-optimal solution. The algorithm initializes a population of subgraphs, each representing a possible solution. To assess the quality of these solutions, a fitness function was applied to each subgraph, measuring how well it solved the problem at hand. Subgraphs with higher fitness scores were considered better solutions.

After evaluating the fitness of each subgraph, the algorithm proceeded to the selection phase, where the top-performing subgraphs were chosen for reproduction. Reproduction involved a crossover process, in which two parent subgraphs were selected, and portions of their structure (i.e., their "genetic material") were combined to create new offspring. This was achieved by swapping nodes between the parent subgraphs at random, generating a new subgraph that inherited traits from both parents.

To ensure diversity in the population and avoid premature convergence on suboptimal solutions, a mutation phase was introduced. During mutation, random changes were made to some of the offspring, modifying their structure slightly. This introduced variability into the population, enabling the algorithm to explore new areas of the solution space. The cycle

of selection, crossover, and mutation was repeated until a satisfactory solution was found, or a stopping condition (such as a maximum number of generations) was met.

The heuristic for estimating the cost of subgraph is determined by the fitness function which measures how well the subgraph solves the optimization problem. The primary heuristic for this problem is based on node connectivity. Where the first component of the fitness function evaluates whether all the target nodes are connected in the subgraph. If a subgraph does not connect all the target nodes then it is given a low fitness score

A criterion for finding an optimal solution is based on the maximum fitness score. If a subgraph achieves a good fitness score then the algorithm considers it. Whereas, if the algorithm does not find a subgraph with perfect fitness within a specified number of generations (in this case, 500 generations) it returns the best subgraph found so far based on the fitness score. Another criterion was set that decides which individual gets to reproduces. Individuals with higher fitness scores are more likely to be selected for reproduction. We also considered individuals with higher fitness score not only have a greater chance of reproducing but might also be selected more than once for reproduction. These points outline how the genetic graph algorithm approaches the problem by evolving a population of subgraphs to minimize cost while ensuring all the target nodes are connected.

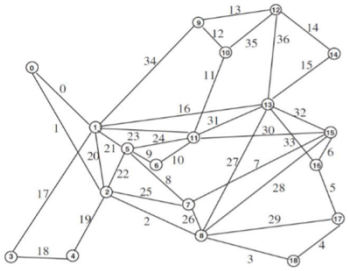

Fig 1 : Initial Graph G

A. Genetic Graph Miner Algorithm

The genetic graph miner is a genetic algorithm that is designed to explore subgraphs G' from a graph G. The class is initialized which takes parameters such as the graph, target nodes, population size and the maximum number of generations for the algorithm to evolve. The class implements the initialize_population() method which creates the initial population of subgraphs where each subgraph is a random sample of edges from the initial graph. The population consists of random subgraphs that are diverse in terms of edge counts and their structures. The fitness function then takes into account for each subgraph and evaluates the quality of each subgraph based on its connectivity to the target nodes while considering unnecessary nodes and edges. The function also penalizes extra nodes that are not target nodes. The final fitness score reflects the trade-off between covering target nodes. The crossover function then combines edges from two parent subgraphs with good fitness to create a new child subgraph the new subgraph inherits characteristics from both parents promoting genetic diversity in the population. The last two processes include the mutation function and evolve function. The mutation function introduces random changes to a subgraph to maintain diversity and prevent premature convergence. A probability of 50% is set where the mutation function will either remove a random edge from the subgraph or add a new edge from the graph that isnt already part of the subgraph. The outcome of this allows the algorithm to explore new configurations with respect to the subgraph. Lastly, the evolve function acts as the core of the genetic algorithm it evolves the population over several generations and optimizes all the subgraphs based on their fitness scores The process continues for a specified number of generations and returns the best subgraph discovered during the evolution process. Fig 2 shows some sample iterations of the results produced by the genetic search algorithm.

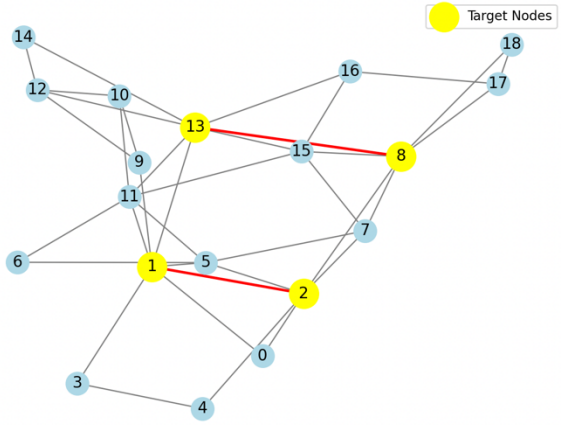

Fig 2 sample subgraphs for targeted nodes

B. Graph Visualization

The visualize graph function is then used to visually represent a graph and utilizes the NetworkX and Matplotlib libraries. It takes in a graph with the target nodes M and then highlights the target nodes in the visualization process. The original graph is then drawn with its edges and overall structure. The subgraph is highlighted with red edges representing the best possible solution.

IV. Experimentation

All of the following experiments were tested using the following M's: 1, 2, 8, and 13

The Trials of the Experiment shown in Table demonstrate the algorithmic performance when the variable population was altered to a diverse range. The experiment was performed through range of 10-100 population size and then, the number of edges for the best Fitness score and average population fitness was received. When the algorithm was implemented with size of 10 population the best fitness score received was 70.70 with 3 number of edges, the average population fitness was 65.76. Similarly, when the population size was increased the fitness, and the number of edges was exact while there was minor difference in average population fitness.

Table I.
ALGORITHM PERFORMANCE FOR DIFFERENT POPULATION SIZES OVER 500 GENERATIONS

Pop Size	Number of Edges	Fitness	Avg Pop Fitness
10	3	70.70	65.76
25	2	94.87	75.90
50	2	94.87	76.62
75	2	94.87	77.27
100	2	94.87	78.24

The Trials of the Experiment shown in Table demonstrate the algorithmic performance when the variable generation was altered to a diverse range. The experiment was performed through range of 10-500 generational size and then, the number of edges for the best Fitness score and average population fitness was calculated through genetic algorithm. When the algorithm was implemented with size of 10 generation the best fitness score received was 63.95 with 5 number of edges, the average population fitness was 56.51. Similarly, to as table I when the generational size was increased the fitness, and the number of edges was exact while there was minor difference in average population fitness.

Table II.
ALGORITHM PERFORMANCE FOR DIFFERENT NUMBERS OF GENERATIONS FOR A POPULATION OF 100 INDIVIDUALS

Gen size	Number of Edges	Fitness	Avg Pop Fitness
10	5	63.95	56.51
50	2	94.87	76.63
100	2	94.87	77.50
200	2	94.87	78.04
500	2	94.87	79.52

The average fitness of the historical best solution found by the algorithm also is also better at larger population sizes.

IV. Discussion

This design of the Graph mining tool applying a Genetic Algorithm explores the lowest cost to reach the targeted nodes. The algorithm utilizes the initialized population which was the subgraph, and the generation used for analyzing the best fitness score through which the process of Genetic Algorithm was achieved.

In terms of discussing future implementations and additions of features to our system. The team can explore different generations and initialize different sets of populations to calculate and analyze better fitness score to achieve more optimized result.

V. Conclusion

The Graph mining tool design completed above provides the results from the performance trials of the genetic algorithm, confirms the idea that maintaining diversity within population is crucial for its success. A diverse population allows the algorithm to explore a wider range of the search space, reducing the likelihood of converging prematurely on local optima. The result also shows that the fitness value increases immensely when the size of population or generation was altered but remain constant after some point. The difference of fitness score on the first two range of size also determined the best solution taking number of edges into account.

Acknowledgment

We extend our gratitude to Professor Sun-Il Kim for providing us with this opportunity to design and implement a safe and secure storage system in P2P. This paper has allowed us to develop a deep understanding of the P2P system and security techniques that can be applied in general for the sharing and retrieval of files. We were able to take into consideration both end-users while designing this system. This allowed us to extensively enhance our knowledge on both client and server sides in terms of security. The research topic allowed us to take different parameters and identify different problems that emerge in the security of file sharing in P2P systems.

REFERENCES

[1] "Tutorial — python-igraph 0.10.6 documentation." python-igraph. Accessed: Oct. 1, 2024. [Online]. Available: https://python.igraph.org/en/stable/tutorial.html

[2] "graph-tool." Graph-tool Library. Accessed: Oct. 1, 2024. [Online]. Available: https://graph-tool.skewed.de/

.