A Genetic Algorithm Applied to Graph Problems Involving Subsets of Vertices

Yaser Alkhalifah
Department of Mathematical
and Computer Sciences
University of Tulsa
600 South College Avenue
Tulsa, OK 74104 USA

Roger L. Wainwright
Department of Mathematical
and Computer Sciences
University of Tulsa
600 South College Avenue
Tulsa, OK 74104 USA
rogerw@utulsa.edu

Abstract - **Many graph problems seek subsets of their vertices that maximize or minimize objective functions on the vertices. Among these are the capacitated p-median problem, the geometric connected dominating set problem, the capacitated k-center problem, and the traveling tourist problem. Prior genetic algorithms research in this area applied a simple mutation of an allele by random replacement. Recently an enhanced operator called hypermutation was developed, proving to be very effective for solving the capacitated p-median problem. We propose a GA with a new heuristic called the nearest four neighbors heuristic (N4N) for solving graph problems requiring a subset of vertices It is an extension of the hypermutation operator. Genetic algorithms that use each of these three mutation operators (simple, hypermutation, N4N) are applied to instances of the four graph-subset problems listed above. Results show that our N4N heuristic obtained superior results compared to the hypermutation and the simple mutation operators in every test case.**

I. INTRODUCTION

There are numerous graph problems where the search for the optimal solution involves obtaining a subset of the vertices of a graph to minimize or maximize some objective function. This paper investigates a genetic algorithm (GA) using the nearest four neighbors mutation operator (N4N), designed specifically for solving graph problems requiring a subset of vertices. The N4N heuristic is an extension to the hypermutation heuristic operator developed recently by Correa, *et al.* [4]. We applied the N4N mutation operator to four representative problems involving subsets of vertices: the capacitated p-median problem, the geometric connected dominating set problem, the capacitated k-center problem, and the traveling tourist problem. All four of these problems have been shown to be NP-hard [3, 8].

The rest of this paper is organized as follows: Section II formally describes the capacitated p-median problem, the geometric connected dominating set problem, the capacitated k-center problem, and the traveling tourist problem. In Section III the fitness functions are reviewed for the four graph problems. In Section IV a genetic algorithm for graph problems is presented including a discussion of the simple mutation operator, the hypermutation heuristic, and our N4N mutation operator. Section V reports the computational results, and Section VI gives the conclusions.

II. THE GRAPH PROBLEMS

A. The Capacitated P-Median Problem

The p-median problem is a facility location problem which can be applied to telecommunications, transportation, scheduling, and distribution problems. Informally, the goal of the p-median problem is to determine p facilities in a predefined set with n ($n > p$) candidate facilities such that the total sum of the Euclidean distances between each demand point and its nearest facility is minimized. The p facilities composing a solution for the problem are called medians. Formally, assuming all vertices of a graph are potential medians, the p-median problem can be defined as follows. Let $G = (V, E)$, be an undirected graph, where V are the vertices and E are the edges. The goal is to find a set of vertices $V_p \subset V$ (median set) with cardinality p, such that the sum of distances between each remaining vertex in $\{V-V_p\}$ (the demand set) and its nearest vertex in V_p is minimized. The capacitated p-median problem has the additional constraint that each candidate facility has a fixed capacity, that is each facility can handle a maximum number of demand points. Correa *et al.* [4] describe a real-world application of the capacitated p-median problem. They describe a university's admission examination where 26 facilities must be selected among 43 available facilities. Each facility has a fixed number of students it can handle to take the examination. There are 19,710 students and each student must be assigned to a facility. The 26 facilities are selected such that the total sum of the distances between each student's home and the facility assigned to the student is minimized.

B. Geometric Connected Dominating Set Problem

The geometric connected dominating set (GCDS) problem is defined as follows. Given a graph $G = (V, E)$, find a minimal subset S of vertices, such that the subgraph induced by S is connected and S forms a dominating set in G. A dominating set is one in which each vertex of the graph is either in the dominating set, or adjacent to some vertex in the dominating set.

For example, there are a set of cities we want to send a radio signal to, and each city has a receiver. However, some cities could have a transmitter as well, and the transmitter will have a known distance for transmitting a signal, called the cover length. Given a fixed number of transmitters, the problem is to determine which cities should be assigned to a transmitter such that all of the cities are covered by at least one transmitter, and each transmitter is covered by another transmitter. The geometric dominating set problem in graph terms asks for a minimal subset of vertices with the following property: each vertex is required to be either in the dominating set, or adjacent to some vertex in the dominating set. The problem is to find a connected dominating set of minimum size, where the graph induced by vertices in the dominating set is required to be connected [6]. The geometric connected dominating set problem can be applied to various wireless communication and network testing problems [6]. Guha and Khuller [6,7] describe an approximation algorithm for the geometric connected dominating set problem. Additional work concerning the geometric connected dominating set problem can be found in [5,9,11].

C. The Capacitated K-Center Problem

The capacitated k-center problem (CKC) is a facility location problem, where it is required to locate *k* facilities in a graph, and assign vertices to these facilities, in order to minimize the maximum distance from a vertex to the facility to which it is assigned. In addition, each facility may be assigned at most *L* vertices. The capacitated k-center problem is defined as follows: given an edge-weighted graph *G = (V, E)* find a subset *S* of *V* with size at most *k* such that each vertex in *V* is "close" to some vertex in *S* [8]. Formally, the objective function is defined as follows in Equation (1):

$$\min_{S\subseteq V}\max_{u\in V}\min_{v\in S} d(u,v) \tag{1}$$

where d is the distance function.

As an application, one may wish to install *k* fire stations and minimize the maximum distance (response time) from every location to its closest fire station. Khuller and Sussmann [8] give an excellent polynomial time approximation algorithm for the capacitated k-center problem. For further reading concerning the capacitated k-center problem see [2,5]

D. The Traveling Tourist Problem

The traveling tourist problem is defined as follows: given a graph *G = (V, E)* find the shortest walk visiting a subset of vertices, such that each vertex is either visited, or has at least one of its neighbors visited. For example, the vertices of the graph could correspond to attractions a tourist might like to see, and an edge between two vertices denotes visibility of one attraction from another. The shortest such walk would guarantee that the tourist sees all points of interest. This problem is closely related to the geometric connected dominating set problem. The difference is that we are not only looking for a connected subset that covers all of the points, but also we are looking for the shortest connected subset that cover all the points [6].

III. FITNESS FUNCTIONS

A. Capacitated p-Median Problem

Once the medians are selected, each vertex is assigned to the median that is the nearest one to it. However, since each median has a fixed capacity, some vertices will have to be assigned to the second (or third, fourth, ...) nearest median to it. Once each vertex is assigned, the fitness of a chromosome can be computed by calculating the sum of Euclidean distances between each vertex and its assigned median. The minimum sum is the optimal solution.

Correa, *et al.* [4] used the ranking-based selection method proposed by Mayerle (1996), given by Equation (2) below:

$$Select(R) = \left\{ r_j \in R \,/\, j = P - \left\lfloor \frac{-1+\sqrt{1+4.rnd(P^2+P)}}{2} \right\rfloor \right\} \tag{2}$$

where *R* is a list *R*=(r_1,r_2, …, r_P), with *P* individuals sorted in increasing order by fitness value, *rnd* $\in$ [0,1) is a uniformly-distributed random number. The symbol $\lfloor\ \rfloor$ is the floor function. The formula returns the position in the list *R* of the individual to be selected. The formula is biased to favor the selection of individuals with smaller fitness values.

B. Geometric Connected Dominating Set Problem

The fitness evaluation for the geometric connected dominating set problem needs to differentiate between any two chromosomes based on the number of points that the subset covers, and also the number of points in the subset which are connected. This is because it is possible to find a solution that covers all of the points, but perhaps not all of the subset points are connected.

Therefore, the fitness function we developed for this problem is shown below in Equation (3):

$$fit = \frac{1}{(X\times 0.8)+(Y\times 0.2)} \tag{3}$$

where X is the number of points covered in the solution, and Y is the size of the connected subset. We arrived at the weighting factors of 0.8 and 0.2 after an extensive number of runs and trials. This seems to be the "best" combination of weights for the type and size of problems we were solving. The weights represent the importance of the components in the candidate solution. Hence, we are concerned more about the number of points covered in the solution (X), than the size of the connected subset, (Y). These weighted factors can be adjusted depending on the data set and the type of problem. For example, if we had a problem in which we do not care about covering all the points as much as we care about how many points we can cover, then we can give Y a larger (or equal) weight in the formula.

C. Capacitated K-Center Problem

In the capacitated k-center problem, the best solution is the one with the minimum sum of edges distances. Hence the fitness function we developed is the summation of the lengths for all the edges.

D. Traveling Tourist Problem

In the traveling tourist problem there are two main factors to consider when determining the fitness function. First, the number of points that have been covered in the solution, and secondly the length of the edges of the connected subset. Hence the fitness function we propose for this problem is shown below in Equation (4):

$$fit = (Y \times 10^{-5}) + \frac{1}{X} \quad (4)$$

where X is the number of points that have been covered in the solution, and Y is the sum of the length of the edges of the connected subset. The smaller the fitness value the better the solution. The reason we gave Y a small weight is because we want to cover as many points as possible in the solution. To differentiate between solutions that cover the same number of points, we use the length of the connected subset (Y). If Y is given more weight in the formula, the GA will try to look for the shortest length for the subset, ignoring the number of points that have been covered in the solution.

IV. A GENETIC ALGORITHM FOR GRAPH PROBLEMS

A. Problem Representation

The chromosome used for all graph problems is simply a list of vertices in the subset. In this GA the genome is interpreted as a set of vertices, in the mathematical sense of a set. That is, there are no duplicated vertices and there is no ordering among the vertices. The vertices are represented as integers in the chromosome, and every chromosome is the same length. We assume the reader is familiar with the fundamental concepts of evolutionary computation and in particular, genetic algorithms. An excellent introduction to genetic algorithms can be found in [10].

B. Crossover

After two parents (chromosomes) have been selected for crossover, the GA computes two exchange vectors, one for each parent, as follows. For each gene of parent 1, the GA checks whether the allele of that gene is also present (in any position) in parent 2. If not, that allele is copied to the exchange vector of parent 1. This means that this allele could be transferred to parent 2 as a result of crossover, since this transfer would not create any duplicate alleles in parent 2's genotype. The same procedure is performed for each allele in parent 2. For instance, let the two parents be {1, 2, 3, 4, 5, 6, 7} and {2, 5, 7, 9, 10, 12, 20}. Their respective exchange vectors are: vp1 = {1, 3, 4, 6} and vp2 = {9, 10, 12, 20}. Once the alleles that can be exchanged have been identified, the crossover operator can be applied.

Crossover is performed as follows. A random natural number *r*, varying from 1 to the number of elements in the exchange vectors minus 1, is generated. This number *r* determines how many alleles of each exchange vector will be swapped between the two parents. Note this procedure guarantees that there will be no duplicate alleles in any of the two children produced by crossover [4].

Crossover is performed whenever the two parents are not identical, i.e. whenever there is at least one vertex in the exchange vectors of parent 1 and parent 2. If the two parents are identical, (that is, their exchange vectors are empty), then one of the duplicate parents is reproduced unaltered for the next generation.

C. Mutation Operators

The main motivation of this research is the implementation and comparison of the GA mutation operators in solving graph problems involving subsets of vertices. The mutation operators are described below.

1 The Simple Mutation Operator

The simple mutation operator simply selects one vertex in the subset of vertices and replaces it with another vertex randomly selected from vertices not in the subset.

2 Hypermutation Heuristic

The hypermutation heuristic proposed by Correa, *et al.* [4] is described below. This operator starts by randomly selecting a percentage of the chromosomes of the population. Correa suggests randomly selecting 10% of the population. Recall a chromosome represents a subset of the nodes of the graph problem. The algorithm then tries to improve the fitness of each of the selected chromosomes as follows. For each node of the chromosome, the algorithm tries to replace it with each of the nodes that are not currently present in the chromosome. For a given node, the replacement that most improves the chromosome's fitness is performed. This is a computationally expensive operator, since a large number of fitness functions must be performed each time it is applied. In precise terms, the hypermutation heuristic operator is implemented as follows:

```
Procedure Hypermutation [4]
Step1:
Randomly select a subset of 10% of the
chromosomes from the entire population

Step2:
FOR EACH chromosome X selected in Step1
    DO
    Let H be the set of nodes in the graph that are not
     currently present in chromosome X
     FOR EACH node index "i" included in set H
        DO
        BEST = X
        FOR EACH node index "j" that is currently
        present in chromosome X
            DO
            Let Y be a new chromosome with the set of
            nodes given by: $(X - \{j\}) \cup \{i\}$
            Calculate the fitness of Y
            If fitness(Y) < fitness(BEST) then BEST = Y
        END FOR
        if fitness(BEST) < fitness(X) then X = BEST
    END FOR
    Insert the new X into the population replacing the old X
END FOR
```

To illustrate the use of the hypermutation operator, consider a simple graph problem with seven nodes, labeled {1, 2, 3, 4, 5, 6, 7}, out of which we want to select a subset of three nodes. Consider a chromosome X, selected to undergo hypermutation, containing the subset of nodes {1, 3, 6}. The set H is {2, 4, 5, 7}, and initially BEST = X = {1, 3, 6}. The algorithm first considers node {2} in H, so that the following new chromosomes are evaluated: {2, 3, 6}, {1, 2, 6} and {1, 3, 2}. Next, the algorithm considers node {4} in H, so that the following chromosomes are evaluated: {4, 3, 6}, {1, 4, 6}, {1, 3, 4}. A "running" BEST is kept considering all options tested. Next node {5} and then node {7} are considered as the algorithm marches through H producing respectively the following additional options to consider: {5, 3, 6}, {1, 5, 6}, {1, 3, 5}, and {7, 3, 6}, {1,7, 6}, {1, 3, 7}. The current value of BEST from this process replaces the original chromosome X in the population. This process is performed for each individual undergoing hypermutation.

3 N4N Heuristic

We developed an enhancement to the hypermutation operator, which we call the nearest four neighbors algorithm (N4N). As described above, the hypermutation operator takes the entire chromosome and mutates it node by node with all nodes not included in the chromosome. Even with the good results that this produces, it wastes a lot of time calculating all possible solutions, good and bad. As a response to this observation we developed a modified algorithm called the nearest four neighbors algorithm (N4N). The idea behind this new algorithm is motivated by the question: what if the best solution comes by mutating a gene with its neighbor, or what if we are in the right neighborhood of the best solution, but can not reach the best solution fast enough because we were too busy looking everywhere else? The N4N algorithm is a local optimization rather than a global optimization technique. The main idea is to mutate every gene with only its neighbors, and not with the entire graph as in the hypermutation operator. This is a very simple concept, but proved to be extremely significant in performance. We tested our algorithm extensively using from one to seven neighbors. The result of our extensive empirical testing showed that using four neighbors proved to give significantly better results compared to using one, two or three neighbors. Our tests also showed that using five, six or seven neighbors did not significantly improve the results over four neighbors. Thus we decided to use four neighbors. Note for extremely large problems, larger number of neighbors might prove to be successful. The N4N algorithm is the same as the hypermutation algorithm except we only consider mutation with the nearest four neighbors to a node as defined by Euclidean distance. We calculate the nearest four neighbors for every vertex in the graph *a priori*. The N4N algorithm is in fact a form of a hybrid optimization. Hybrid optimization techniques are extremely popular. For a general discussion on hybrid optimization techniques see [1]. The N4N algorithm is described below.

```
Procedure N4N
Step1:
Randomly select a subset of 10% of the
chromosomes from the entire population

Step2:
FOR EACH chromosome X selected in Step1
    DO
    FOR EACH node "i" included in set X
        DO
        BEST = X
        Let H be the set (of up to four) of the neighbors of
        node "i" that are not currently present in
        chromosome X
            FOR EACH node index "j" that is currently
            present in the set H
                DO
                Let Y be a new chromosome with the set of
                nodes given by: $(X - \{i\}) \cup \{j\}$
                Calculate the fitness of Y
                If fitness(Y) < fitness(BEST) then BEST = Y
            END FOR
            if fitness(BEST) < fitness(X) then X = BEST
    END FOR
    Insert the new X into the population replacing the old X
END FOR
```

Note set H is calculated for each node in a chromosome. For a given node, *i,* H will contain, among the four closest neighbors to *i,* those neighbors that are not already present in the chromosome. If H is empty, that is, all four neighbors of a node are already in the chromosome, then this node is not mutated.

V. COMPUTATIONAL RESULTS

We developed genetic algorithms for the capacitated p-median problem, the geometric connected dominating set problem, the capacitated k-center problem, and the traveling tourist problem. We used the ranking-based selection method (described above) for the capacitated p-median problem, and roulette selection for the other problems. We used the same crossover operator (described above) in all problems tested. In each problem three different mutation operators (as described above) were applied. In all cases we used a population size of 100 and crossover rate of 0.9. Data sets used to test our graph problems came from ftp://ftp.zib.de/, specifically ftp://ftp.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsp/. Each entry in all four tables in this section depicts the average results from executing each algorithm ten times.

Table 1 depicts the results of the geometric connected dominating set problem, applying the simple mutation operator, a GA with the hypermutation heuristic, and a GA with the N4N heuristic on two different datasets. Dataset 1 has 656 nodes with a cover length of 300, and dataset 2 has 318 nodes with a cover length of 400. These datasets are called d657.tsp and lin318.tsp, respectively in the above ftp site. Using dataset 1 we tested various subsets sizes of vertices ranging from 43 down to 39. A solution for 38 vertices could not be found. The GA with the simple mutation operator never did find a solution in any of the cases tested. The GA with hypermutation found a solution for a subset of 43 and 42 vertices, but nothing smaller. The N4N algorithm found a solution in every case, and in addition required significantly less generations. In dataset 2 the best solution found was 40 vertices. The N4N algorithm found the solution taking on the average 48 generations, while the GA with the simple mutation operator and the GA with hypermutation took an average of 609 and 106 generations, respectively

Table 2 depicts the results of the capacitated k-center problem, applying a GA with the simple mutation operator, a GA with the hypermutation heuristic, and a GA with the N4N hypermutaton on the same dataset 1 and dataset 2. In each data set we used k (number of facilities) = 15, with L (capacity) = 44. As shown in Table 2, considering both data sets, the N4N hypermutation algorithm gave superior results (smaller distances).

Table 3 depicts the results of the traveling tourist problem, applying a GA with the simple mutation operator, a GA with the hypermutation heuristic, and the N4N hypermutaton algorithm on dataset 1 and dataset 2. In dataset 1 a solution using 51 vertices was found, and in dataset 2 a solution using 50 vertices was found. As shown in Table 3, considering both data sets, the N4N hypermutation algorithm again gave superior results (smaller distances).

We tested the capacitated p-median problem using three large datasets with 15,112, 18,482 and 33,750 nodes respectively. In each case p = 30, n = 60, and the capacity is 525, 650, and 1150, respectively. As in previous examples we ran each algorithm ten times using the GA with the simple mutation operator, a GA with hypermutation, and with the N4N heuristic. The results shown below in Table 4 indicate the average distance over all executions. Clearly, the N4N heuristic provided superior results in every instance (smaller distances). As a time comparison, the GA with hypermutation for the 33,750 size problem ran in six hours, and the GA with the N4N for the same problem ran in four hours on the same computer.

So why does the N4N heuristic perform better than the hypermutation when the hypermutation is trying all of the options that the N4N algorithm is trying plus additional options? Our best conjecture is that it is a combination of two factors: First speed; considerably more work (generations) can be done in the same amount of time if only four neighbors are considered, and in addition these four neighbors are most likely to produce the best results anyway. Secondly it is well known among combinatorial problems that a sequence of locally optimal moves may not always lead to a global optimal solution. How much this contributes to the superior performance of N4N is not known for sure.

.

TABLE 1. GEOMETRIC CONNECTED DOMINATING SET PROBLEM RESULTS WITH THE N4N ALGORITHM

Data Set	Size	Chrom. Length	Cover Length	GA with Simple Mutation	GA With Hyper-mutation	GA With N4N
1	656	43	300	No Solution	Solution found after 70 generations	Solution found after 17 generations
1	656	42	300	No Solution	Solution found after 52 generations	Solution found after 18 generations
1	656	41	300	No Solution	No Solution	Solution found after 248 generations
1	656	40	300	No Solution	No Solution	Solution found after 289 generations
1	656	39	300	No Solution	No Solution	Solution found after 412 generations
2	318	40	400	Solution found after 609 generations	Solution found after 106 generations	Solution found after 48 generations

TABLE 2. CAPACITATED K-CENTER PROBLEM REAULTS WITH THE N4N ALGORITHM

Data Set	Size	Chromosome Length	Capacity	GA with Simple Mutation	GA With Hypermutation	GA With N4N
1	656	15	44	Length 161,044	Length 159,997	Length 158,725
2	318	15	44	Length 100,674	Length 94,777	Length 90,843

TABLE 3. TRAVELING TOURIST PROBLEM RESULTS WITH THE N4N ALGORITHM

Data Set	Size	Chromosome length	GA with Simple Mutation	GA With Hypermutation	GA With N4N
1	656	51	No Solution	Tour length 25,779	Tour length 23,630
2	318	50	Tour length 36,752	Tour length 32,227	Tour length 31,994

TABLE 4. THE CAPACITATED P-MEDIAN RESULTS WITH THE N4N ALGORUTHM

Data Set	Size	Chromosome length	Capacity	GA with Simple Mutation	GA With Hypermutation	GA With N4N
1	15,112	30 out of 60	525	57,680	30,064	29,232
2	18,452	30 out of 60	650	14,104	10,407	9,343
3	33,750	30 out of 60	1150	1,189,869	873,047	838,554

VI. CONCLUSIONS

We implemented the hypermutation heuristic concept presented by Correa, *et al.* [4]. The hypermutation heuristic represents one of the best known heuristics to date for solving graph problems involving subsets of vertices. We developed a modification to the hypermutation heuristic called the nearest four neighbors (N4N) algorithm. We developed genetic algorithms that applied the traditional simple mutation operator, the hypermutation mutation operator, and the N4N mutation heuristic to four representative graph problems: capacitated p-median problem, the geometric connected dominating set problem, the capacitated k-center problem, and the traveling tourist problem. Results show that our N4N heuristic obtained superior results compared to the hypermutation and the simple mutation operators in every test case.

REFERENCES

[1] Baeck, T., Fogel, D. and Zbigniew, M., editors, Handbook of Evolutionary Computation, Chapter D: Hybrid Approaches, IOP Publishing, 2000.

[2] Bar-Ilan, J., Kortsarz, G., and Peleg, D. “How to allocate network centers", J. Algorithms, 15:385-415, 1993.

[3] Carey, M. R. and Johnson, D. S., "Computers and Intractability: A guide to the theory of NP-Completeness", Freeman, San Francisco, 1978.

[4] Correa, E. S., Steiner, M. T. A., Freitas, A. A., Carnieri, C. A Genetic Algorithm for the P-median Problem In: Genetic and Evolutionary Computation Conference - GECCO 2001, 2001, San Francisco, California. Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2001. San Francisco, California: Morgan Kaufmann Publishers, 2001.

[5] Feige, U. “A threshold of ln n for approximating set-cover", 28th ACM Symposium on Theory of Computing, pages 314-318, 1996.

[6] Guha, S., and Khuller, S., “Approximation algorithms for connected dominating sets", Proceedings of the 4th Annual European Symposium on Algorithms, pages 179-193, 1996.

[7] Guha, S., and Khuller, S., “Improved approximation algorithms for node weighted Steiner trees", manuscript,1997.

[8] Khuller, S., and Sussmann, Y. J. The capacitated k-center problem. In Proceedings of the ,4th Annual European Symposium on Algorithms, Lecture Notes in Computer Science 1136, pages 152-166, Berlin, Springer, 1996.

[9] Mata, C. S., and Mitchell, J. S. B. “Approximation algorithms for geometric tour and network design problems", Proceedings of the 11th Annual Symposium on Computational Geometry, pages 360-369, 1995.

[10] Mitchell, M. An Introduction to Genetic Algorithms MIT Press, 1996.

[11] Slavik, P. “A tight analysis of the greedy algorithm for set cover", 28th ACM Symposium on Theory of Computing, pages 435-441, 1996.