E. Determination of Best Solution

The algorithm runs for a predetermined number of generations. After a new generation is produced, it searches for the best solution in the new generation. The best solution must both be complete (i.e., includes all nodes in M and not have disconnected edges) and have the smallest number of edges. Once the generation's best solution is found, it is compared with the best solution found in previous generations (the historical best). If the current generation's best solution is better than the historical best, then the algorithm marks the current best solution as the new historical best. After all the generations have been completed, the historical best solution is deemed to be the best solution the algorithm could find. Depending on many factors, such as the size and structure of G, the number of nodes in M, the relative position of the nodes in M on G, and the initial population size, the algorithm might find the best solution after very few generations. In such cases, the algorithm will still explore the search space until the total number of allowed generations has passed.

III. Experimental Results

Experimental trials of the algorithm's performance reveal that its performance depends more on population size than the number of generations the algorithm is allowed to run. Table I shows the results of running the algorithm for 100 generations at various population sizes. Table II shows the results of running the algorithm for a different number of total generations for a population of 100 chromosomes. (Trials were performed on the graph in Fig. 1 with M = {3, 12, 15, 18}.)

TABLE I.
ALGORITHM PERFORMANCE FOR DIFFERENT POPULATION SIZES OVER 100 GENERATIONS

Pop. Size	Averages for Historical Bests (after 100 Generations)[a]				
	Num. of Edges	*Fitness*	*Generation Found*	*Solutions Found*	*Avg. Population Fitness*[b]
10	13.75	22.18	28.0	3 /10	22.31
20	10.33	32.69	10.66	5 / 10	29.30
50	8.33	38.30	12.11	9 / 10	32.67
100	6.90	44.43	12.5	10 / 10	34.70
500	6.00	50.00	9.9	10 / 10	35.65
1000	6.00	50.00	10.00	10 / 10	35.64

[a] Averages based on ten trials per population size

[b] Avg. fitness for all chromosomes in population in final generation

The population size trials show a relationship between solution quality and the size of the population. Average fitness scores for all chromosomes in the population are higher for larger populations. The average fitness of the historical best solution found by the algorithm also is also better at larger population sizes. E.g., a population size of 50 finds best solutions that on average on contain 8.33 edges and a fitness score of 38.30, whereas a population size of 500 finds solutions with only 6.00 edges and a fitness score of 50.00. For this particular M, 6 edges is the optimal cost. In addition to better average fitness of the population as a whole and improved solution quality for the historical best, larger population size also helps the algorithm to find a solution in more of the trials. E.g., population sizes of 10 and 20 only find the solution in 3 and 5 out of the ten trials respectively, whereas a population size of 50 finds a solution ninety percent of the time, and populations over 100 individuals always find a solution. Although larger population size does correlate with better performance, there is a point after which increased population size does not yield benefits. E.g., in these trials, populations larger than 500 individuals do not improve performance.

Given how well the algorithm performs with a population size of I 00 individuals, the threshold where size increase no longer provides benefits might, with this particular graph and M, be much lower than 500.

Unlike population size, the number of generations the algorithm is allowed to run for, when tested with a population size of 100, does not seem to impact performance. The only effect of the total number of generations is seen in the average chromosome fitness for the population at large when comparing a total run of 10 versus 20 generations. Average population fitness is better when the algorithm runs for 20 generations compared to a 10 generation run, but no benefit is seen from running the algorithm longer than that. This is not surprising based on the findings in Table I, which show that for population sizes of 50 or more, the historical best solution is found after only 12.5 generations. Bringing together the results of these two experiments, it seems that when a population size is large enough (e.g., 500), the algorithm finds the optimal solution (in this example, 6 edges) on average by the tenth generation. One takeaway is that an important task in performance tuning the algorithm for a given graph (or problem) is determining the population size at which optimal (or near optimal) solutions are found (and found quickest), but beyond which no benefits are had by increasing size.

TABLE II.
ALGORITHM PERFORMANCE FOR DIFFERENT NUMBER OF GENERATIONS FOR POPULATION OF 100 INDIVIDUALS

Gens.	Averages for Historical Bests (Pop. Size = 100)[a]				
	Num. of Edges	*Fitness*	*Generation Found*	*Solutions Found*	*Avg. Population Fitness*[b]
10	7.50	40.77	7.60	10 / 10	23.90
20	8.10	39.44	11.1	10 / 10	32.2
50	7.50	42.10	10.5	10 / 10	34.47
100	6.80	45.14	13.5	10 / 10	33.51
500	8.60	39.66	9.00	10 / 10	31.98
1000	6.60	47.01	12.30	10 / 10	32.87

[a] Averages based on ten trials per number of generations

[b] Avg. fitness for all chromosomes in population in final generation

IV. Conclusion

The findings from the performance trials of the generic algorithm discussed here support the intuition that maintaining a diverse chromosome population is key to the effectiveness of a genetic algorithm. This is because a population with diverse chromosomes will explore a larger portion of the search space and be less likely to get stuck on local optima. The strategies in the algorithm outlined above attempt to foster this diversity (e.g., through initial random population creation, reproduction and mutation strategies, and tournament selection) so that widest range of solution possibilities are explored. The fitness heuristic, on the other hand, narrows the population by selecting for the chromosomes in the population that are helpful to building an optimal final solution. With too small of a population size, performance can suffer because the fitness score can reduce the diversity of the population too quickly, allowing certain chromosomes (and their genes) to dominate before enough of the search space is explored. In that case, solutions might be found, but not necessarily high quality ones. Larger population size (up to a limit) improves the effectiveness of these strategies, because it allows enough space for diversity in the population to flourish, before the fitness score hones in on the best solutions.

IV. Experimentation

All of the following experiments were tested using the following M's: 3, 12, 15, and 18.

A. Varying the Starting Population

To see how the miner works, we tested it by altering the value of its starting population, the results of which can be found in Table 1. As we increased the population, the cost tended to decrease, but the semi-stochastic nature of our genetic algorithm resulted in some deviating results. It seemed to be mostly random whether or not a certain starting population would yield low cost subgraphs, but the only certain effect of increasing the starting population was that the ending population increased greatly. A small starting population can yield some slightly-less-than-optimal results, but the drawback to having a large starting population is how much additional time it takes to execute. A potential solution to this could be increasing the number of ways *generation()* classifies and purges unneeded individuals.

TABLE I
INCREASING STARTING POPULATION.

Population	Total Solutions	Cost of Best Subgraph	Ending Population
5	23	10	1729
10	109	8	4096
15	50	7	7784
20	62	9	10317
50	521	6	26285
100	105	8	57260

Only 10 generations were run for each population.

B. Varying the Number of Generations

Next, we varied the number of generations that the program ran for, seen in Table 2. The most immediately notable difference between increasing population and increasing generations was the amount of time it took each time we increased the number of generations. Much like the population, we increased the generations by 5 each time, but both the time and the space it required increased exponentially with each iteration. In our test of 5 generations, we did not get a single solution, though the most fit subgraph was 1 active edge from being connected. For 10 generations, the ending subgraph was incredibly messy, and the last test ended with a huge population and a result that still had a random, unconnected active edge in the graph along with the solution.

For our program, it appears that to get better results, the wisest decision is to increase the starting population whilst keeping the number of generations constant. Unfortunately, it appears too inefficient to run it otherwise.

TABLE II
INCREASING GENERATIONS RAN.

Generations	Total Solutions	Cost of Best Subgraph	Ending Population
5	0	0	241
10	2	13	4101
15	486	8	93437

Starting population remained at 10 for all tests.

V. Conclusion

Our program was designed to be a graph miner - to find the least cost subgraph of a graph of assorted vertices and edges. The cost of each subgraph was determined by the total number of edges used to connect a given set of vertices, called M's. By using a genetic algorithm, we were able to achieve a program that almost accomplished its ultimate goal. In most of our experimenting, the results we received were often not the most optimal solution, and the amount of time it took to reach these results was especially excessive since the additional time rarely increased the optimality of the final result. Despite this, it appeared that a moderately sized starting population with few generations tended to yield the best results.

If both the way generations were handled and the way the fitness of an individual was calculated were altered to be both more efficient and inclusive of some elements unbeknownst to us, our miner could have been more time efficient and produced more optimal results.

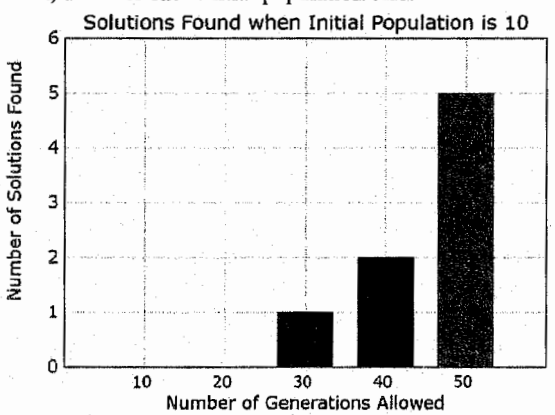
III. Experiments

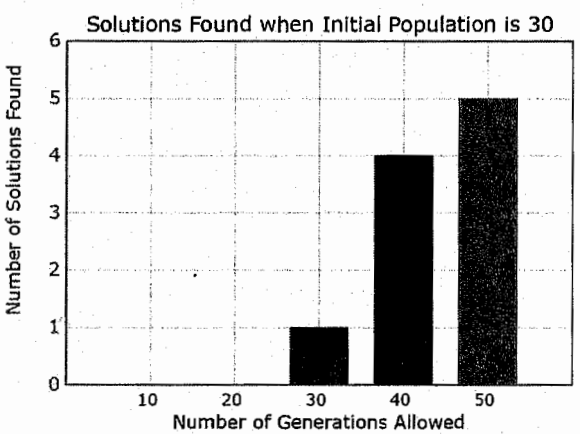
A. Variables and Controls

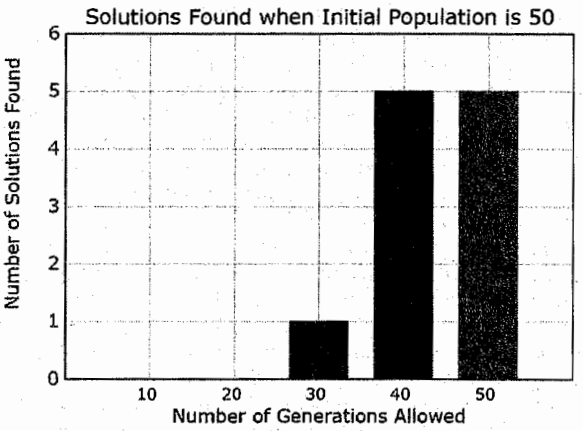
We performed experiments to determine how varying the initial population size affects the likelihood of finding a solution within a given generation limit. The experiments we performed consisted of five different initial population sizes (10, 30, 50, 70, and 90) and five different generation limits that the algorithm was allowed to run for (10, 20, 30, 40, and 50). Our algorithm was run with each initial population size, each size being run for each generation limit. This would be repeated five times for each pairing of initial population sizes and generation limits, in order to gather a number of how many times a solution was found for each pairing.

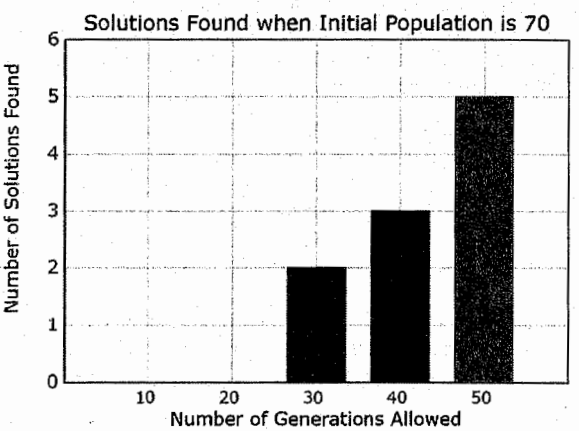
B. Results

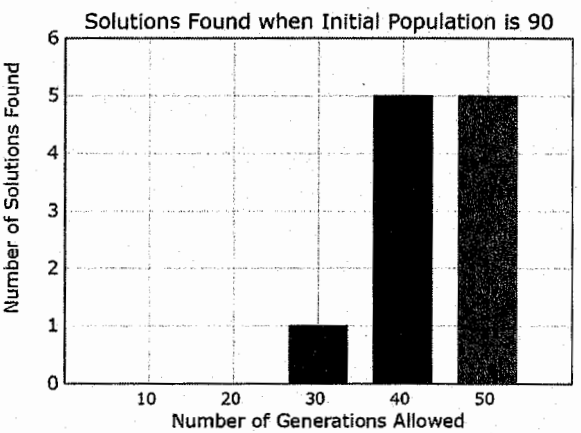
The results are shown in terms of the number of times a solution was found within five attempts in a generation limit, done for each initial population size:











C. Discussion

The results of the experiment were rather similar in a few ways. None of the 5 different population sizes ever resulted in a found solution when the generation limit was set to 10 or 20. In addition, all five of the initial population sizes resulted in five found solutions when the generation limit was set to 50. Interestingly, however, the highest amount of solutions overall came from the initial populations with sizes of 50 and 90, with the initial populations of sizes 30 and 70 having