New England University Transportation Center
77 Massachusetts Avenue, E40-279
Cambridge, MA 02139
utc.mit.edu

New England
University Transportation Center

Year 25 Final Report

Grant Number: DTRT13-G-UTC31

Project Title:

Clustering Algorithms for Transit Network Design

Project Number:	Project End Date:	Submission Date:
UCNR25-35	May 31, 2018	August 27, 2018

Principal Investigator:	Nicholas E. Lownes	**Co-Principal Investigator:**	
Title:	Associate Professor	**Title:**	
University:	University of Connecticut	**University:**	
Email:	Nicholas.lownes@uconn.edu	**Email:**	
Phone:	(860)486-2717	**Phone:**	

1. Introduction

The transit network design problem (TNDP) dates back over five decades, receiving increasing attention over the past two. Simply put, the TNDP seeks to do two things: 1) Select groups of transit stops that should form routes and 2) Establish a sequence in which those stops should be visited. Because the underlying optimization problems are combinatorial in nature and fall into the class of NP-Hard problems, approximation algorithms are used in any realistically-sized network. Previous attempts to solve the TNDP have taken path-based approaches; working from an initial route set and perturbing, extending or shortening routes in that initial set. This approach remains the standard today, though it's concepts originated in an era where computational power was limited and prohibitively expensive.

A modified Genetic Algorithm (GA) will be employed, which will allow variants on traditional cost minimization, such as equity maximization. This report contains a detailed discussion and validation of the genetic algorithm (GA) used to solve equitable Traveling Salesman Problem (EqTSP). First, each step of the GA will be discussed, focusing on the procedures and algorithmic structures which are unique to this specific algorithm. Then experimental evidence will be provided to validate decisions regarding its algorithmic structure, including the decision to clone winning solutions into the next generation and the use of rogue parents. Finally, a sensitive analysis is conducted on the five input parameters: population size, tournament size, number of rogue parents, mutation rate, and convergence criteria. All of the preliminary experiments and sensitive analysis were conducted on the Sioux Falls network. Once the best algorithmic structure and input parameters were determined, the GA was tested on the Qatar national network.

2. Overview of the GA

This section will describe the GA in technical terms and focus on the components which make it unique. Most of the decisions made in the GA can be explained in terms of either intensification or diversification. The purpose of a GA is to drive closer and closer to the optimal solution with each generation, a process known as intensification (Blum and Roli 2003) However, the population of solutions must maintain enough diversity to overcome local minima and find the true optimal solution. Maintaining diversity means that some individual solutions may move further from the optimal solution even as the best solutions continue to improve.

The pseudocode for the primary routine is shown below.

PRIMARY ROUTINE: Genetic Algorithm

Subroutines indicated in bold, italics.

```
begin
    R := ∅
    while |R| < p do
    begin
        r := Generate Solution (N, c_ij, α)
        R := R ∪ r
    end;
    genCount := 1; convergenceVal := ∞
    while genCount < genConverge do
    begin
        WIN := Run Tournament (R, t, fit(r))
        pop := WIN; reproPool := ∅
        while |reproPool| < numRogues do
        begin
            r := Generate Solution (N, δ_ij)
            reproPool := reproPool ∪ r
        end;
        reproPool := reproPool ∪ WIN
        while |pop| < p do
        begin
            PA := randomly selected r from reproPool
            reproPool := reproPool − PA
            PB := randomly selected r from reproPool
            reproPool := reproPool ∪ PA
```

```
                    if EAX Crossover(PA, PB) produces a solution do:
                        offspring := EAX Crossover(PA,PB)
                        n := random number between 0 and 1
                        if n > mutationRate then pop := pop ∪ offspring
                        else do
                            offspring := Mutate(offspring)
                            pop := pop ∪ offspring
            end;
            bestSolution := Find Best Solution (R, fit(r))
            if bestSolution ≥ convergenceVal then genCount := genCount + 1
            else do:
                convergenceVal := fit(bestSolution); genCount := genCount + 1
                solution := bestSolution
        end;
end;
```

Notation:

i	origin nodes
j	destination node
c_{ij}	travel time between i and j along the shortest path
r	route (solution)as an ordered set nodes. Subscripts refer to the position of an element (node)in ordered set r.
N	set of nodes
R	set of routes (solutions) r
p	population size
t	tournament size
numRogue	number of rogue parents
genConverge	number of generations to convergence
mutationRate	proportion of offsprings mutated

2.1 Generating Initial Solutions

Generating good initial solutions is necessary for the algorithm to converge on optimal or near-optimal solutions. Given the size of the solution space for this problem, it is possible to generate an initial population that contains both diverse and good solutions. The solution space contains

all possible permutations of the nodes within a network. The size of the solutions space can therefore be calculated as shown below:

$$P(n,k) = \frac{n!}{(n-k)!} \quad (1)$$

$$Size\ of\ Solution\ Space = \frac{|N|!}{(|N| - |N|)!} = \quad |N|! \ (2)$$

Consider the Sioux Falls network which contains only 24 nodes. Even though it is a relatively small network, it contains $6.2045*10^{23}$ possible solutions. This is why it is important to start with a reasonably good population.

The pseudocode below was used to generate initial solutions.

SUBROUTINE: Generate Random Solution

Notation in primary routine: GENERATE SOLUTION (N, c_{ij}, α)

Output: A route (solution)

begin
$S := N;\ r := \emptyset$
$i :=$ randomly selected element from set S
$S := S - i$
$r := r \cup i$
$firstNode := i$
while $S \neq \emptyset$ **do**
begin
Assign probability of selection to each node in $S, p_{sel,j \in S} = f\left(\frac{1}{{c_{ij}}^{\alpha}}\right)$
$j :=$ element selected from S according to probability distribution p_{sel}
$S := S - j$
$r := r \cup i$
$i := j$
end;
$r := r \cup (i, firstNode)$
Output r
end;

The α parameter allows the user to adjust the importance of the nearness of nodes in generating initial solutions. A higher α value places greater emphasis on the nearness of nodes. For the Sioux Falls experiments, α was set equal to 1. For larger networks, it is necessary to increase α. For larger networks, it was helpful to apply a second strategy of not allowing initial solutions which exceeded a certain threshold. The TSP solution cannot exceed twice the cost of the minimum spanning tree (MST). The cost of the MST can be found *a priori* using Kruskal's algorithm (Kruskal 1956). Solutions that do not meet this threshold are not added to the initial population.

2.2 Tournament

The tournament determines which solutions will be allowed to enter the reproductive phase of the GA and which will be discarded. The population is split into smaller groups of size *t* and the best solution from each group is then added to the reproductive pool. While it may seem most sensible to simply rank the solutions and the pick the top solutions for inclusion in the reproductive pool, most GAs implement an indirect process, such as tournament, in an effort to maintain a diverse set of good solutions. Note that this process does guarantee the best solution, will be included in the reproductive pool.

The pseudocode below was used to conduct the tournament.

SUBROUTINE: Conduct a tournament on set of routes R given tournament size t. Each route r will be evaluated using fitness function, fit(r).

Notation in primary routine: RUN TOURNAMENT *(R, t, fit(r))*

Output: Set of "fittest" solutions

begin
    $WIN := \emptyset; POP := R$
    while $POP \neq \emptyset$ **do**
    begin

```
            T := ∅
            while |T| < t do
            begin
                r:= randomly selected route from R
                POP := POP − r
                T := T ∪ r
            end;
            WIN_VAL := ∞
            for each r in T do
                if fit(r) < WIN_VAL then WIN_VAL:= fit(r) and WIN_R := r
            WIN := WIN ∪ WIN_R
        end;
        Output WIN
end;
```

2.3 Reproduction

The purpose of the reproduction phase is to create a new generation of solutions which drives the algorithm closer to the optimal solution while providing new diversity. This GA automatically clones a copy of the tournament winners into the next generation, a decision which will be further investigated later. The purpose of this step is to ensure that the population maintains a certain level of quality. Then several newly generated solutions, or rogues, are added to the reproductive pool. To the best of the author's knowledge, this is a completely unique procedure. This is a method for adding diversity to the population and must be implemented in moderation. Its effectiveness will be discussed extensively. A crossover function then generates new solutions from the solutions in the reproductive pool. This function uses pieces from two good solutions and therefore, will hopefully create new good solutions which contribute to both the intensification and diversification processes. Finally, a mutation function is applied to small

proportion of solutions. This function makes small, random changes to solutions, increasing diversity in the wider population.

2.3.1 Crossover Function

This GA used the EAX crossover proposed by Nagata and Kobayashi (1999). Because the solutions are represented as permutations rather than binary arrays, they require a special crossover function. The pseudocode for the crossover can be found in Nagata and Kobayashi (1999).

2.3.2 Mutation Function

The mutation function is only applied to a small proportion of the newly generated offspring solutions. This is because, like the addition of rogue parents, the mutation functions primary purpose is to diversify the population. Some of the mutations will help the algorithm overcome local minima and find better solutions, while others will worsen solutions. Initially, the intention of the authors was to replace the mutation function with the rogue parents. However, as will be shown in the following sections, both functions proved necessary to finding optimal solutions. This mutation function selects a small, random segment of the solution and reinserts it into another portion of the solution. This segment may or may not be reversed before reinsertion. The pseudocode below shows exactly how the mutation function operates.

SUBROUTINE: Mutate route

Notation in primary routine: MUTATE(r)

Output: Route r with mutation

begin

$maxLength := \lfloor (|r| - 1) \,/\, 5 \rfloor$

$length :=$ random integer between 2 and $maxLength$

$start :=$ random integer between 1 and $|r| - length - 2$

$tempSeg := [r_{start}, r_{start+length}]$

$r := r - tempSeg$

```
        insertPt := random integer between |r| − 2
                    k := random number between 0 and 1
        if k ≤ 0.5 then do
            begin
                reverse tempSeg
            end;
        insert tempSeg after insertPt
end;
```

2.4 Convergence

At the end of each generation, the algorithm checks for convergence. This GA uses the number of generations without an improvement to the best solution as the convergence criteria. Ideally, the convergence criteria should balance quality of solutions with time to convergence. The convergence criteria should be set to a value at which it is unlikely the best solution will substantially improve if the algorithm were to continue running. A discussion of where this value should be set will included in the following sections.

3. Description of Experiments

An extensive set of experiments was conducted to validate and explore the two unique features of the GA:

1. The automatic "cloning" of tournament winners into the next generation
2. The addition of "rogue parents" (or newly generated solutions) into the reproductive pool.

These experiments were conducted to determine the impact of these features on the quality of the solutions and the efficiency of the algorithm. The experiments were also used to determine the effects of the five GA parameters (population size, tournament size, rogue parents, mutation rate, and convergence criteria).

In this analysis, four different algorithmic structures were tested, based on all possible combinations of two different decisions. The first decision was whether or not to clone parents into the next generation. The second decision was how to incorporate rogue parents into the algorithm. The first possibility is simply generating new solutions each iteration; however this may be time consuming, particularly for large networks. An alternative method which may save time is generating a small pool of rogue parents *a priori* which can be pulled from whenever rogue parents are necessary. Table 3.1 below outlines the 4 algorithm structures. Algorithm 1 is the algorithm described by the pseudocode in the previous section.

Table 3.1 Algorithmic Structures for Experiments

Algorithm 1	With clones; New rogue parents each generation
Algorithm 2	With clones; Rogue parents from pool
Algorithm 3	Without clones; New rogue parents each generation
Algorithm 4	Without clones; Rouge parents from pool

The algorithmic structures were tested on the travelling salesman problem (TSP) for Sioux Falls using all possible combinations of the parameters shown in Table 3.2. For each of the 4 algorithmic structures and 120 parameter combinations, the GA was run 30 times. The solutions found using the GAs were compared to the known optimal solution to the problem which was found using a branch and bound method in GAMs.

Table 3.2 Parameters for Experiments

Population Size	100, 300
Tournament Size (as a proportion of population)	0.05,0.10
Rogue Parents (as a proportion of population)	0, 0.03, 0.05, 0.1, 0.2
Mutation Rate	0, 0.01, 0.10
Convergence Criteria (Number of generations without improvement)	10,40

4. Discussion of Results

Each of the algorithmic structures and parameters will be evaluated with regards to both the quality of solutions and the time to convergence. Finding quality solutions is prioritized over finding solutions quickly, however time to convergence cannot become so high that the GA becomes unusable.

4.1 Results of Algorithmic Structure Testing

Algorithm 1, which clones parents into the next generation and generates new rogue parents each iteration, converged on the known optimal solution significantly more frequently than the other algorithmic structures. Figures 3.1 (a-d) plots the time to convergence versus the proportion of runs converging on the optimal solution for each of the experiments run on the different algorithmic structures. Please note, some outliers took more than 100 seconds to converge and are not shown in these figures.

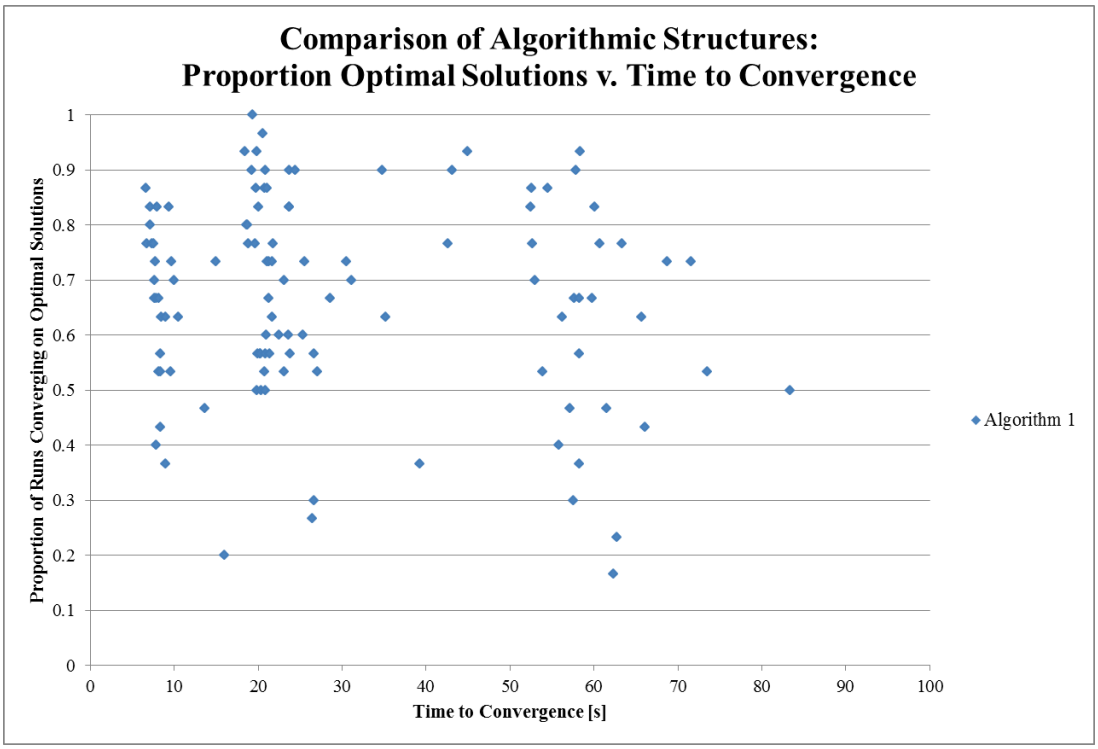

Figure 3.1 (a) Algorithm 1: Clones, New Rogue Parents

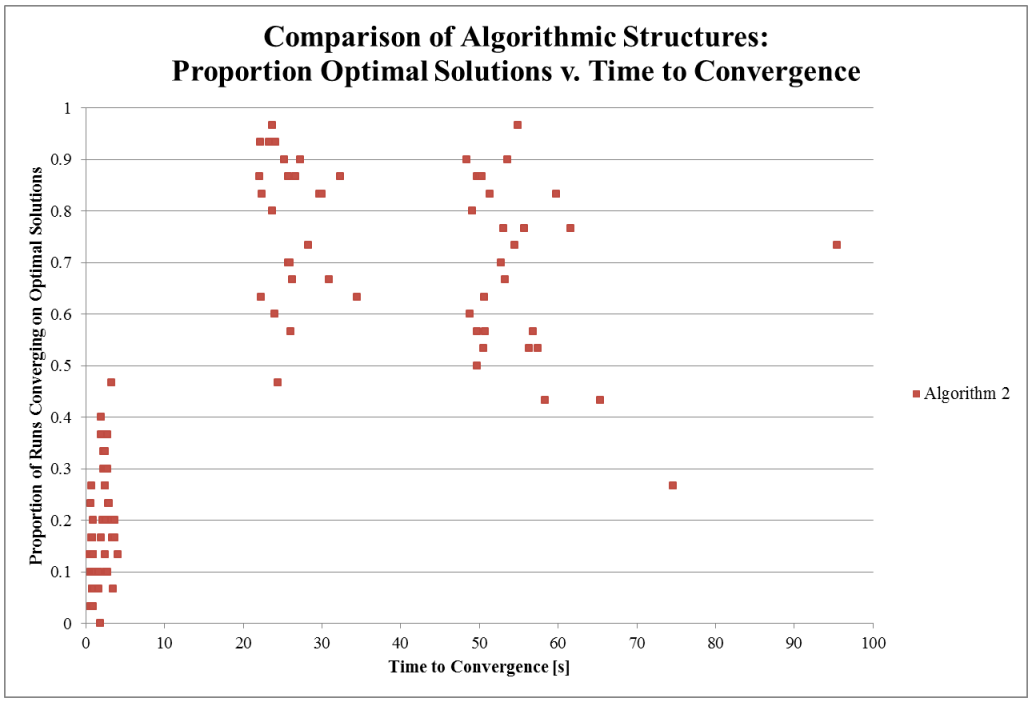

Figure 3.1 (b) Algorithm 2: Clones, Rogue Parents from Pool

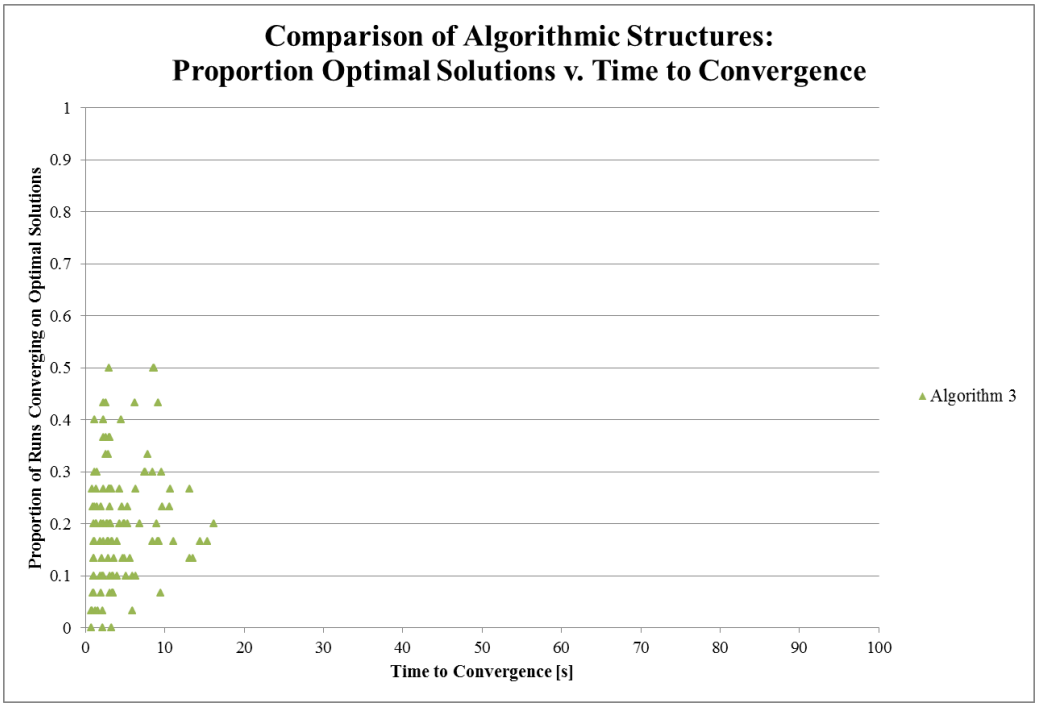

Figure 3.1(c) Algorithm 3: No Clones, New Rogue Parents

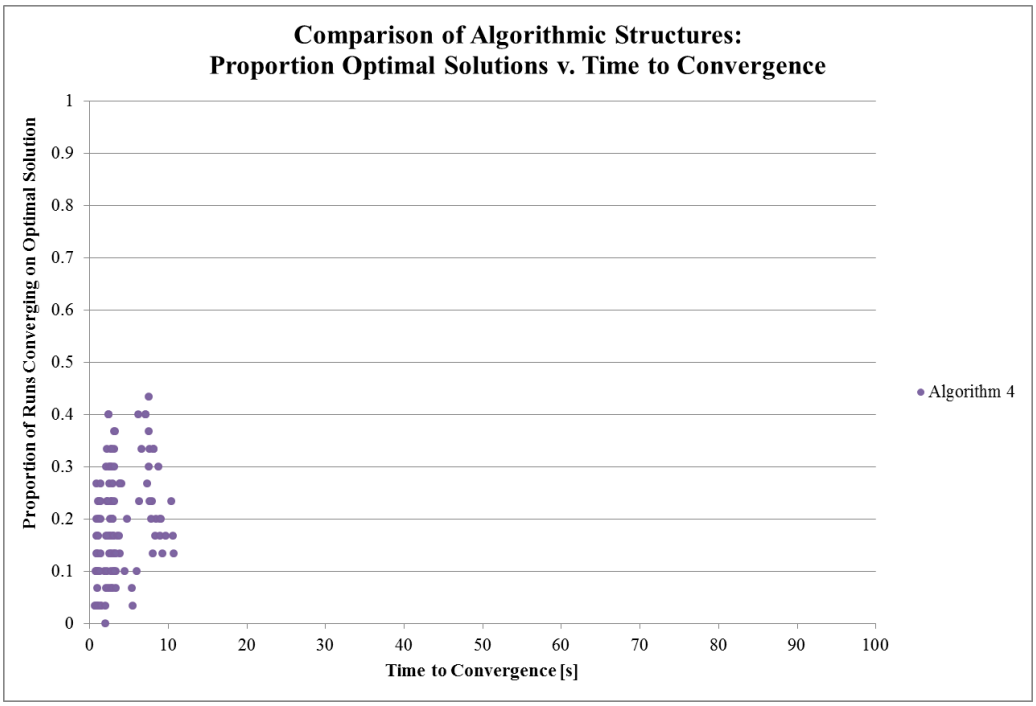

Figure 3.1(d) Algorithm 4: No clones, Rogue parents from pool

Figure 3.2 below compares the distribution of results for the different algorithmic structures using box plots.

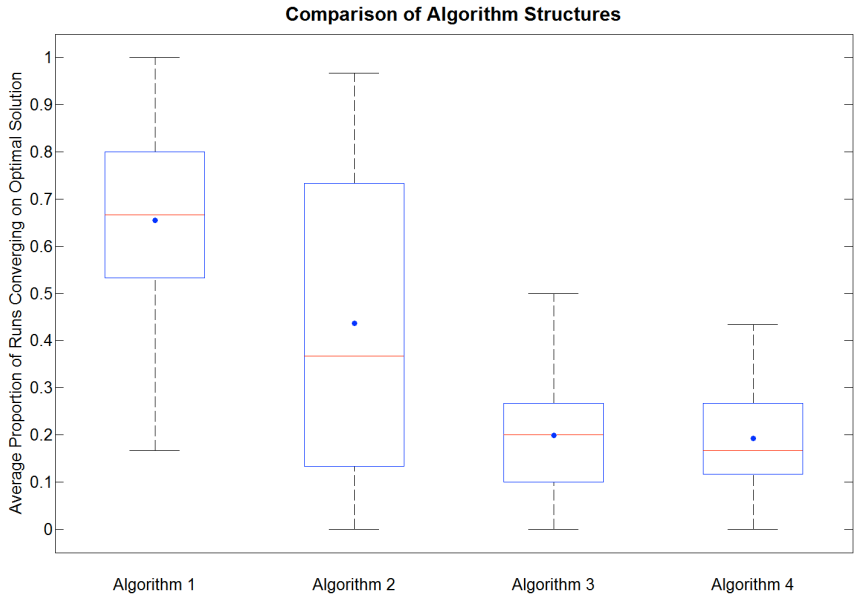

Figure 3.2 Comparisons of Algorithmic Structures by Quality of Solutions

Figures 3.1(a-d) and Figure 3.2 show that algorithms 3 and 4, those which did not clone tournament winners into the next generation were unable to converge on the optimal solution even 50% of the time. These algorithmic structures were immediately discarded. The differences in the quality of algorithmic structures 1 and 2 were less obvious. A t-test comparing algorithms 1 and 2 showed that algorithm 1 converged on the optimal solution significantly more often than algorithm 2 ($p< 0.001$). There was no statistically significant difference in time to convergence between algorithms 1 and 2.

To summarize, cloning the tournament winners into the next generation significantly increased the proportion of runs converging on the known optimal solution. Generating new rogue parents each generation also significantly increased the proportion of runs converging on the known optimal solution as compared to pulling clone parents from a pre-generated pool. There was not a significant difference in the time to convergence between the two treatments of rogue parents. Given these results, algorithm 1 will be used for all future testing.

4.2 Results of Parameter Testing

Calibrating the parameters to their actual optimal values would require the development of a secondary heuristic; a line of research which is outside the scope of this project. However, the experiments conducted on the selected set of parameters for algorithm 1 provides insights on the effects of parameter values. While each of the five parameters was tested for their impact on the quality and time to convergence, this analysis will focus on the rogue parent and mutation rate parameters. Before going into a detailed discussion of these parameters, this section will provide a brief summary of the others.

The smaller population size (100) converged on the optimal solution significantly more often ($p<0.001$) in significantly less time ($p=0.04$). The smaller tournament size (0.05) converged on

the optimal solution significantly more often ($p<0.001$) and had no significant influence on time. The higher convergence criteria (40 generations without improvement) converged on the optimal solution significantly more often ($p<0.001$) but required significantly more time ($p=0.04$). However, it is worth noting that while the GAs with a 40 generation convergence criteria did require significantly more time to converge, that does not mean the time was necessarily unreasonable. Table 3.3 below shows the combinations of parameters in which more than 90% of the runs converged on the optimal solution. (A table showing the full set of parameter combinations can be found in Appendix A.) For example, the parameter combination shown in the first row has a convergence criterion of 40 generations and returns the optimal solution in 100% of runs, yet each run only takes 19.33 seconds on average.

Table 3.3 Results for Best Parameter Combinations (Sioux Falls)

Pop Size	Tournament	Rogues	Mutation Rate	Gen	Runs Converging on Optimal [%]	Avg Time to Convergence [s]
100	0.05	0.05	0.10	40	100.00	19.33
100	0.05	0.10	0.01	40	96.67	20.67
100	0.05	0.10	0.00	40	93.33	18.47
300	0.05	0.03	0.01	10	93.33	19.73
100	0.05	0.00	0.10	40	93.33	45.98
300	0.05	0.03	0.10	40	93.33	58.81
100	0.05	0.05	0.00	40	90.00	19.06
100	0.05	0.05	0.01	40	90.00	21.07
100	0.05	0.03	0.10	40	90.00	22.67
100	0.05	0.20	0.10	40	90.00	24.98
300	0.05	0.00	0.01	10	90.00	35.88
100	0.05	0.00	0.01	40	90.00	43.20
300	0.05	0.03	0.01	40	90.00	56.37

Unlike the other parameters, both the rogue parent and mutation rate parameter had the option of being set to 0. This means that in some experiments no rogue parents were added and/or the

mutation function was never applied. The five possible values for the rogue parent parameter were compared to each other using a t test. The results are shown in Table 3.4 below.

Table 3.4: Comparison of Rogue Parent Parameters: t-Test p values

Value of Rogue Parent Parameter	0.00	0.03	0.05	0.10	0.20
0.00		0.0002	0.0008	0.0037	0.2676
0.03			0.1179	0.1410	0.0001
0.05				0.8826	0.0005
0.10					0.0236
0.20					

GAs with rogue parent parameters set at 0.03, 0.05, or 0.10 converged on the optimal solution significantly more frequently than those set at 0 (no rogue parents) or 0.20. The interpretation of these results can be further colored by the boxplots of shown in Figure 3.3.

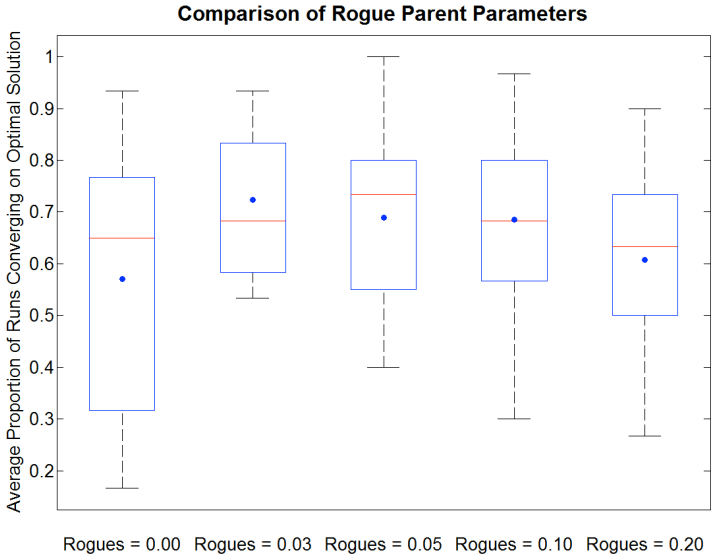

Figure 3.3 Comparison of Rogue Parent Parameter by Quality of Solution

Figure 3.3 shows that the distribution of the proportion of runs converging on optimal is much larger when rogue parents are not included. This suggests that not including the rogues made the GA much more sensitive to the parameters. Even when the rogue parent parameter was set to

0.2, which led to the GA converging on the optimal solution significantly less than the other parameters, the lower bound of the distribution is noticeably higher than when rogue parents are not included at all.

Additionally, GAs with rogue parent parameters of 0.05 or 0.10 converged significantly faster than those set at other values. These tests suggest that the introduction of rogue parents into the population is a useful way to improve both the quality and efficiency of the GA for small network TSPs.

The GAs with mutation rates of 0.10 converged on the known optimal solution significantly more often than those with mutation rates of 0.01 and 0. The distribution of the results can be seen in figure 3.4 below.

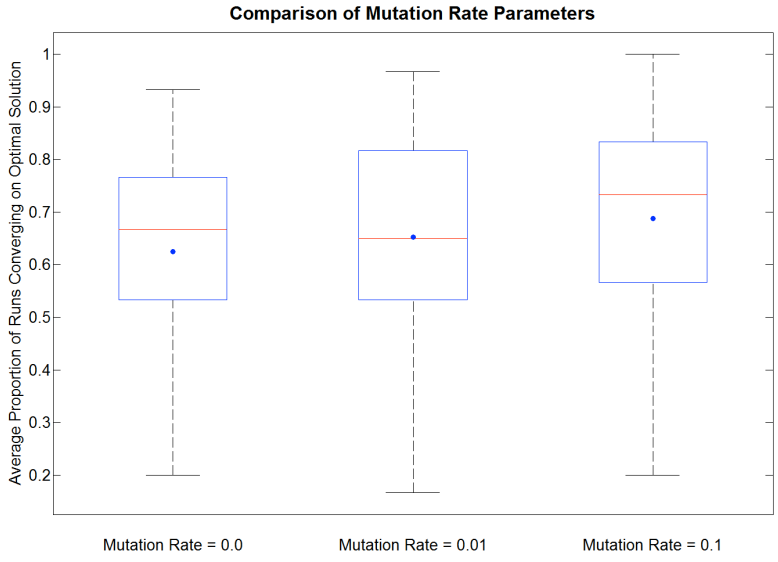

Figure 3.4 Comparison of Mutation Rate

This was a surprising result. The insertion of the rogue parents into the reproductive pool was initially intended to replace the mutation function. Not only do these results suggest it is necessary to keep the mutation function but the results also suggestion maintaining a particularly

high mutation rate. This may be due to the limited parameter values tested in this analysis but it is still unusual. The high mutation rate did lead to significantly higher time to convergence.

4.3 Larger Networks

After running this extensive set of experiments on the Sioux Falls network, a smaller set of experiments was conducted on the much larger Qatar national network (National TSPs, 2009). The Qatar national network is a fully connected network containing 194 nodes. Because the network was much larger, the GA took longer to converge, especially for poor parameter combinations. For this reason, the set of experiments on the Qatar TSP does not contain the complete set of parameter combinations. Table 3.5 below shows the results of some of these experiments. All of the experiments have a tournament size of 0.05. Due to space constraints, this parameter is not shown in Table 3.5.

Table 3.5: Results for Best Parameter Combinations (Qatar National)

Pop Size	Rogues	Mutation Rate	Gen	Runs within 1% of Optimal [%]	Runs within 5% of Optimal [%]	Avg Time to Convergence [s]
300	0.03	0.10	80	0.27	1.00	670.82
300	0.10	0.10	80	0.27	1.00	918.55
300	0.05	0.00	40	0.23	0.97	465.82
100	0.05	0.00	80	0.20	1.00	215.96
300	0.05	0.01	80	0.17	1.00	758.32
300	0.05	0.01	40	0.13	1.00	472.89
300	0.10	0.10	40	0.13	1.00	666.53
100	0.05	0.00	40	0.10	1.00	128.30
100	0.10	0.10	40	0.10	1.00	201.70
100	0.10	0.01	80	0.10	1.00	274.56
100	0.10	0.10	80	0.10	0.93	290.14
300	0.03	0.10	40	0.10	1.00	445.82
300	0.05	0.00	80	0.10	1.00	729.16
300	0.10	0.01	80	0.10	1.00	879.88

The Qatar experiments found a similar relationship between parameters and solution quality as the Sioux Falls experiments. The one exception is that GAs using the larger population parameter (300) were found to converge within 1% of optimal significantly more often. This change aligns with previous research that larger problems benefit from larger initial populations. Though the GA takes longer to converge, it is capable of converging on near-optimal solutions for Qatar national TSP.

5. Conclusions

The GA presented contained two unique features: cloning tournament winners into the next generation and the use of rogue parents. Cloning tournament winners was shown to significantly increase the number of runs converging on the optimal solution. The inclusion of rogue parents was also shown to significantly increase the number of runs converging on the optimal solution. However, including too many rogue parents can decrease the quality of the solutions. The recommended number of rogue parents to insert into the reproductive pool is between 0.05 and 0.10 of the population. These parameter values also caused the GA to converge significantly faster than the other rogue parent values.

This GA was tested on both the small Sioux Falls TSP and the larger Qatar National TSP. For the right combination of parameters, the GA was able to find the optimal solution for the Sioux Falls TSP in 30 out of 30 runs. While the GA was not able to find the exact optimal solution for the Qatar National TSP, it was able to find solutions within 1% of optimal for 8 out of 30 runs and 5% of optimal for 30 out of 30 runs for some parameter combinations. The best solution found by the GA on the Qatar national network was within 0.08% of optimal. As these models are applied to larger networks, it would be worthwhile to parallelize this algorithm. GAs are easy

to parallelize, as there are many independently functioning pieces. Parallelizing this algorithm could drastically improve computational time.

6. References

Blum and Roli (2003) Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Computing Surveys. 35(3): 268-308.

Kruskal, J. B. (1956) On the shortest spanning subtree of a graph and the travelling salesman problem. *Proceedings of the American Mathematical Society*, Vol 7, pp 48-50.

Nagata, Yuichi and Kobayashi, Shigenobu (1999). Analysis of edge assembly crossover for the travelling salesman problem. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics v3: p III-628-III-633

Appendix A: Full Set of Parameter Experiments on Sioux Falls TSP

Pop Size	Tournament	Rogues	Mutation Rate	Gen	Runs Converging on Optimal [%]	Avg Time to Convergence [s]
100	0.05	0.05	0.10	40	100.00	19.33
100	0.05	0.10	0.01	40	96.67	20.67
100	0.05	0.10	0.00	40	93.33	18.47
300	0.05	0.03	0.01	10	93.33	19.73
100	0.05	0.00	0.10	40	93.33	45.98
300	0.05	0.03	0.10	40	93.33	58.81
100	0.05	0.05	0.00	40	90.00	19.06
100	0.05	0.05	0.01	40	90.00	21.07
100	0.05	0.03	0.10	40	90.00	22.67
100	0.05	0.20	0.10	40	90.00	24.98
300	0.05	0.00	0.01	10	90.00	35.88
100	0.05	0.00	0.01	40	90.00	43.20
300	0.05	0.03	0.01	40	90.00	56.37
100	0.05	0.05	0	10	86.67	6.68
300	0.05	0.1	0.1	10	86.67	19.72
100	0.05	0.1	0.1	40	86.67	20.74
300	0.05	0.03	0.1	10	86.67	21.09
300	0.05	0.1	0.1	40	86.67	52.60
300	0.05	0.05	0.01	40	86.67	54.51
100	0.05	0.1	0.1	10	83.33	7.16
100	0.05	0.03	0.1	10	83.33	8.00
100	0.05	0	0.01	10	83.33	9.38
300	0.05	0.03	0	10	83.33	20.05
100	0.05	0.03	0	40	83.33	23.75
100	0.05	0.03	0.01	40	83.33	23.75
300	0.05	0.05	0.1	40	83.33	52.44
300	0.05	0.03	0	40	83.33	60.11
300	0.05	0	0.01	40	83.33	444.36
100	0.05	0.03	0	10	80.00	7.15
100	0.05	0.2	0.01	40	80.00	18.62
100	0.05	0.2	0	40	80.00	18.78
100	0.05	0.05	0.01	10	76.67	6.77
100	0.05	0.1	0.01	10	76.67	7.36
100	0.05	0.05	0.1	10	76.67	7.51
300	0.05	0.1	0.01	10	76.67	18.80
300	0.05	0.05	0.01	10	76.67	19.68
300	0.05	0.05	0	10	76.67	21.78
100	0.05	0	0	40	76.67	42.63
300	0.05	0.1	0	40	76.67	52.69

300	0.05	0.2	0	40	76.67	60.68
300	0.05	0.05	0	40	76.67	63.33
300	0.05	0	0	40	76.67	148.45
100	0.1	0.1	0	10	73.33	7.75
100	0.05	0.2	0.1	10	73.33	9.70
100	0.05	0	0.1	10	73.33	14.98
300	0.05	0.05	0.1	10	73.33	21.04
100	0.1	0.1	0.1	40	73.33	21.24
300	0.05	0.2	0	10	73.33	21.66
100	0.1	0.05	0.1	40	73.33	25.54
100	0.1	0.2	0.1	40	73.33	30.51
300	0.05	0.2	0.01	40	73.33	68.69
300	0.05	0.2	0.1	40	73.33	71.58
300	0.05	0	0.1	40	73.33	762.52
100	0.05	0.03	0.01	10	70.00	7.65
100	0.05	0	0	10	70.00	10.01
100	0.1	0.05	0	40	70.00	23.14
300	0.05	0.2	0.1	10	70.00	31.08
300	0.05	0.1	0.01	40	70.00	52.95
100	0.05	0.1	0	10	66.67	7.67
100	0.1	0.1	0.01	10	66.67	7.84
100	0.05	0.2	0	10	66.67	8.11
300	0.05	0.1	0	10	66.67	21.25
100	0.1	0.03	0.1	40	66.67	28.60
300	0.1	0.05	0.1	40	66.67	57.63
300	0.1	0.03	0.01	40	66.67	58.29
300	0.05	0	0.1	10	66.67	59.76
100	0.1	0	0.1	40	66.67	354.50
100	0.05	0.2	0.01	10	63.33	8.51
100	0.1	0.1	0.1	10	63.33	8.95
100	0.1	0.03	0.1	10	63.33	10.48
300	0.1	0.03	0.01	10	63.33	21.65
300	0.05	0	0	10	63.33	35.19
300	0.1	0.03	0	40	63.33	56.21
300	0.1	0.2	0	40	63.33	65.66
100	0.1	0	0.01	40	63.33	114.02
300	0.1	0.1	0	10	60.00	21.01
100	0.1	0.2	0	40	60.00	22.45
100	0.1	0.2	0.01	40	60.00	23.62
100	0.1	0.03	0	40	60.00	25.32
100	0.1	0.05	0.01	10	56.67	8.41
300	0.1	0.05	0.1	10	56.67	19.96
300	0.1	0.03	0.1	10	56.67	20.20

300	0.1	0.05	0	10	56.67	20.27
300	0.1	0.03	0	10	56.67	20.81
300	0.1	0.1	0.01	10	56.67	21.33
300	0.1	0.1	0.1	10	56.67	23.81
100	0.1	0.03	0.01	40	56.67	26.65
300	0.1	0.03	0.1	40	56.67	58.23
100	0.1	0.03	0	10	53.33	8.17
100	0.1	0.03	0.01	10	53.33	8.41
100	0.1	0.2	0.01	10	53.33	9.63
100	0.1	0.1	0.01	40	53.33	20.78
100	0.1	0.05	0.01	40	53.33	23.08
300	0.05	0.2	0.01	10	53.33	27.06
300	0.1	0.05	0	40	53.33	53.93
300	0.1	0.2	0.1	40	53.33	73.48
300	0.1	0.05	0.01	10	50.00	19.89
100	0.1	0.1	0	40	50.00	20.30
100	0.1	0	0.1	10	50.00	20.91
100	0.1	0	0.01	10	50.00	83.32
100	0.1	0.2	0.1	10	46.67	13.64
300	0.1	0.1	0.01	40	46.67	57.12
300	0.1	0.1	0.1	40	46.67	61.50
100	0.1	0.05	0.1	10	43.33	8.39
300	0.1	0.2	0.01	40	43.33	66.04
100	0.1	0.05	0	10	40.00	7.88
300	0.1	0.05	0.01	40	40.00	55.83
100	0.1	0.2	0	10	36.67	8.93
300	0.1	0.2	0.1	10	36.67	39.24
100	0.1	0	0	40	36.67	58.27
300	0.1	0	0.1	40	36.67	5512.16
300	0.1	0.2	0.01	10	30.00	26.63
300	0.1	0.1	0	40	30.00	57.56
300	0.1	0.2	0	10	26.67	26.46
300	0.1	0	0.01	40	26.67	3420.00
300	0.1	0	0	10	23.33	62.70
100	0.1	0	0	10	20.00	16.02
300	0.1	0	0	40	20.00	200.86
300	0.1	0	0.1	10	20.00	1534.67
300	0.1	0	0.01	10	16.67	62.32