Using Genetic Algorithms to solve the Multiple Traveling Salesman Problem through Internal Crossover

Timothy Murray
253-302-0264
tsm78@cornell.edu

ABSTRACT

The Multiple Traveling Salesmen Problem (MTSP) is an NP-hard combinatorial and scheduling optimization problem defined in the following way: a company has to send salesmen to visit *m* cities. The company has *n* salesmen to distribute the cities between, and would like to plan out the sales routes in the cheapest way possible. Additionally, they would like to make sure that the salesmen have balanced schedules, since if there are 100 cities and one salesman is visiting 96 of them, the company's customers in those 96 cities will be upset by the long time gaps between visits. We can immediately see then that there may be some conflict between minimizing the total cost, and minimizing the cost of the most expensive sales route. This problem is easily seen to be NP-hard by noting that for the specific case when $n = 1$, the MTSP is the Traveling Salesman Problem, which is NP-hard.

1. INTRODUCTION

The Multiple Traveling Salesman Problem (MTSP) is a scheduling optimization problem related to the Traveling Salesman Problem (TSP) but more similar to the Vehicle Routing Problem (VRP). The primary difference is that there is no common supply depot that all routes in the MTSP must stop at, as in the VRP. This makes the VRP more valuable for shipping and supply problems, while the MTSP is more valuable for planning out circuits, such as bus routes or police patrol routes. While both interest me, I chose to work on the MTSP over the VRP because relatively little work has been done on it, while the VRP is well-explored.

I approached this problem initially with the intent to minimize the total cost a plan of routes, and use the other properties of the problem primarily with a focus on doing that. My initial algorithms are built with this in mind. However, I found that while this was doable and not terribly difficult, it frequently resulted in unbalanced plans, where a single route bore the majority of the cost. After trying several modifications to the problem while still keeping the focus on minimizing the total cost, I continued to run into this problem. Upon starting serious testing, I found what I suspect is the answer, which I will share later.

2. SET-UP

2.1 Method

I created several algorithms that run using a combination of factors: crossover method, parent selection, diversity maintenance, plan model, and heuristic initialization. The plan model is the most important so we will address that first. Each of my algorithms implemented one or more of these techniques.

2.1.1 Plan Model

A plan is a set of routes and other properties associated with them such as their costs, the number of times each city is visited by all routes, and the variance in both the number

of cities each route visits and its cost. I created two plan models, the first of which will be referred to as a hard plan. In a hard plan, each city is visited exactly once by the set of routes, with no exceptions allowed. The routes are structured differently as well, but this will be addressed in greater detail in the implementation section. The second plan model I will refer to as a soft plan. In a soft plan, the number of times each city is visited is simply non-negative. However, there is a penalty for skipping a city entirely, and cost incentive of not visiting a city multiple times rapidly drives a newly initialized soft plan to visiting each city exactly once. While effectively the same, the soft plan is structured in a way that makes experimenting more easy. In practice, algorithms using the hard model seem to find slightly cheaper plans, and those using the soft model find more balanced ones.

2.1.2 Crossover

The next factor to consider is the crossover method. I experimented with two methods of crossover, internal and external crossover. External crossover takes place between two plans by exchanging a subset of their routes. This turned out to be a particularly ineffective technique: in the hard model the need to visit each city exactly once destroys most of the substructure of the resultant plans when they are cleaned into valid hard plans after the exchange, and the pressure that effectively forces soft model plans to visit each city exactly once means that when routes are exchanged, unless they contain the exact same subset of cities, the offspring incur large penalties for not visiting cities. They would have to be set aside and optimized through hill-climbing or some other method for several turns before they could be fairly evaluated. When I implemented external crossover on each of these models, both developed plans at a rate that put their performance several times worse than hill-climbing algorithms.

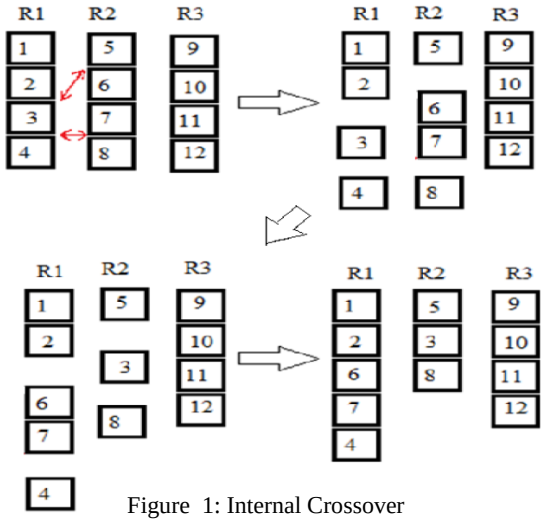

Figure 1: Internal Crossover

Fortunately, the internal crossover method is far more effective (see Figure 1 above). The technique is used frequently in VRP genetic algorithms which was where I first saw it. Here crossover is performed between routes within a plan. It is easy to implement, as each route is effectively a string of city number or can be converted to one, and prevents the possibility of a plan no longer possessing a city and incurring a penalty or having the substructure destroyed. I used two point crossover on the routes. One of the drawbacks to this method is that there is no point in keeping a population of plans unless the algorithm is modeled on a beam search, since there is no operator that allows the plans to interact with or affect each other.

2.1.3 Parent Selection

I used two methods of selecting parents for new routes. The first was to look at the ratio of route cost to number of cities visited, with a smaller ratio being preferred. I used roulette sampling to do that. I created the second method when I began trying to fix the problem of unbalanced schedules. In this method, parents are ranked on their size. The choice of first parent ranked larger

routes well, while the choice of second parent ranked smaller routes well. Using the cost of the parent routes as their size was more effective than their number of cities. The point of this type of selection was to provide more opportunities to redistribute the cities in larger routes, and it was effective.

2.1.4 City Redistribution

Diversity maintenance was a more difficult topic to address. It is difficult to determine the similarity of two routes that share no common cities. We could look at the variance of the number of cities between routes, but since we want that to be low it is not something to fix. It is better to consider what I eventually did not as diversity maintenance but as wealth redistribution, except in the sense that several routes consisting of one or two nodes each are not particularly diverse. Whenever a route obtained more cities than a certain threshold, determined by both the number of cities and the total number of routes, all cities were removed from it. The cities were then redistributed uniformly at random into all routes, including the now empty one. Before this happened the old plan was preserved, but it did happened regardless of how much more expensive the new plan was. This operator is an improvement, but the threshold needs to be adjusted.

2.1.5 Heuristic Drafting

I experimented with one other operator, a heuristic draft. Here, routes are selected uniformly at random. If they are empty then they are given an unvisited city uniformly at random. If they are not empty, then they are given the unvisited city the shortest distance from the city they most recently acquired. This continues until all cities are drafted. The point of this draft was to give the plan enough substructure that the formation of a giant route would be likely to be more expensive and thus would be less likely to occur quickly, but not enough for the plan to be stuck optimizing that substructure if something more beneficial was nearby. The results were mixed: As expected, performance was extremely good in short term, but the algorithms that implemented drafting had very highly varying performance over time.

2.2 Related Work

This project is very closely related to work being done on the VRP. Genetic algorithms are a common technique for solving, though they are not the optimal. On average, depending on how large the problem is, solutions from genetic algorithms range from 4.16 to 11.79% higher than best known solutions depending on the problem[1]. I do not remember in which article I first read about internal crossover, the technique is fairly common in solving both the MTSP and VRP, but a detailed explanation can be found in the paper *Solve the Vehicle Routing Problem with Time Windows via a Genetic Algorithm*[2] which I found helpful when I was formulating my initial algorithm. However, in addition to work done on the VRP, in recent years there has been a surge of interest in using genetic algorithms on the MTSP. One must be careful though in how it is defined. Several papers define it as a relaxation of the VRP where the salesmen have a common depot but no fuel, load, or time restrictions[3], and the goal is just to minimize cost. This is very different from the problem that I addressed, where there is no common depot and the time it takes for each route, while not restricted, is one of the factors we would like to optimize. Additionally, while there are papers that address the same problem that I do[4], they all seem to be addressing it as a single objective problem, minimize the total cost, rather than also working to create a balanced plan.

My overarching goal in this project was to develop an algorithm that encourages the routes to co-evolve into a balanced plan. I believe that my algorithm which uses a soft-model plan and all three of the balancing methods above can reasonably be said to do that. It did better on all medium and small city sets in creating a balanced plan than any other algorithm except a hill climber which is willing to take massive penalties for skipping cities in order to lower the cost of the highest route. This was especially notable on the small sets: it was the only algorithm to never once create a plan that had one route visit all but two cities, which were continuously occupied by other routes, and still maintain a very low total cost. On the large data set, two algorithms managed to create slightly balanced plans, but only one had a lower total cost. The combination of these mechanisms seems to strongly encourage the co-evolution of these routes into a balanced and cheap plan.

2.3 System Architecture and Implementation

All coding for this project was done in an object-oriented environment in MATLAB. This was done to take advantage of the remote access provided to Cornell Mathematics computers.

2.3.1 Hard-Model Plans

I came up with the idea of hard plans before soft plans, and as one might expect there are significant differences in their implementation. A hard plan is an object of type MTSPRoutes2. The hard plan tracks its routes as a matrix of route numbers and positions. A route number r and position p in column i means that i is the p^{th} city in route r. Additionally, the hard plan contains a distance matrix between cities, a penalty matrix representing the cost of stopping at each city, and other expected items such as its total cost, the costs of its routes, etc. that it needs in order to allow functions to operate on it. Hard plans are randomly initiated assigning the n cities to the r routes uniformly at random but in order, and randomly swapping the contents route numbers and positions in the route matrix between cities $n \log n$ times, so that we can expect to have involved each city in a swap at least twice. There are two forms of mutation operators: mutate1 swaps the positions of two cities within a route. Mutate2 takes a city from r_1 and inserts it in r_2. Technically this is a crossover, but if it was not included the hill-climbing algorithms would be incapable of altering the amount or number of cities in the routes. The crossover operator is given two routes. It randomly chooses break points on each and performs two point crossover. The operation is simplified by the fact the two routes share no common elements, so the cities later in the route just need to have their positions in the route raised or lowered by the appropriate amount.

2.3.2 Soft-Model Plans

A soft plan is an object of type Plan. The soft plan tracks its routes as an array of Salesman objects. Salesman objects contain a matrix of the distances between cities and a penalty matrix of the cost of stopping at each city. They also contain the expected fields such as their route, a matrix where the city at position i is the i^{th} stop in the route, the number of cities they visit, the total number of cities, how many unique cities they visit (no other salesmen) and the ratio of the cost of their route to the number of cities they visit. In addition to the array of Salesman, soft plans also carry the distance and penalty matrix as well as the number of times each city is visited, the penalty for missing a city, the variance in the number of cities and route cost of each salesman, a matrix of the cities per route, and a pareto

score and rank. The latter two are from before the external crossover attempts proved unsuccessful. Plans can be initialized in two ways. The first is randomly, each Salesman includes each city with probability *p*, and they are then swapped until the order is sufficiently random. The second is semi-randomly, if an algorithm is using Heuristic drafting. Routes are selected uniformly at random. If they are empty then they are given an unvisited city uniformly at random. If they are not empty, then they are given the unvisited city the shortest distance from the city they most recently acquired. This continues until all cities are drafted.

Soft plans also have static functions that inflict mutation, handle the redistribution operator, and evaluate their costs after a change. There are four kinds of mutation, all of which are handled by calling a Salesman static function. The first randomly swaps two cities within a Salesman's route. The second adds a city at random into the Salesman's route. The third inverts two cities in a route, and the fourth removes a city from a route, provided the route is not empty.

2.3.3 Algorithms

All algorithms are combinations of the factors plan model, redistribution, crossover, selection, and heuristic drafting. The plan models have static functions to handle these, although the MTSPRoutes2 object does not have functions implemented for redistribution or heuristic drafting.

3. EXPERIMENTATION

3.1 Methodology

Results (plans) are evaluated by their total cost and by the cost of the most expensive route within the plan. We seek to minimize both. The strategy for most of my algorithms is to evaluate based on total cost but to implement breeding selection based on maximum route cost. This resulted in some algorithms on average created slightly more expensive plans than a hill-climber that seeks only minimize total cost, but considerably more balanced ones. . I did experiment with a linear combination of the total and maximum route costs, but it appears to be too dependent on the geometry of the cities.

Independent variables are the routes within the plans. They are manipulated by the algorithms in order to measure the maximum and total cost, the dependent variables. While work on the MTSP is not uncommon, all examples I have been able to find focus on the single-variable optimization of the total cost. Since most treat it as a relaxation of the route cost restraint in the VRP, this is not unexpected, but my project is the only one I have found that seeks to encourage these restraints to grow rather than enjoy the liberation from them. In this my goal was at least somewhat unique.

My final algorithm, SpSzHeuGA, which utilizes a soft-plan model, city redistribution, size-based selection, and heuristic drafting, was on average dominated only twice across all the city sets I tested on. In both cases, it dominates the other algorithm on a different city set. It on average dominates the effective hill-climbing algorithm on all but on data set I tested on.

3.2 Results

Due to size constraints, all tables are placed on the two pages following the interpretation subsection. Table 1 explains how to interpret the names of the algorithms in Table 2. Using this table, we see that HpRrGA is a genetic algorithm that runs on hard plans and utilizes Ratio Roulette selection. I tested

the six genetic and two hill-climbing algorithms on the following data sets

Self-Travel cost in Table 2 is the cost incurred if a route travels from the city to itself. It is an experiment meant to discourage small isolated routes. The results are summarized in the following table:

Table 3 contains the experimental results of running the algorithms on the data sets. A dash in a cell means that this information is not available. For Average Total Costs this is because, due to the long computation time of some of the algorithms and sets, trials for one algorithm and set were run across multiple computers, and the results sometimes were not rejoined properly. If only an Average Max Cost cell is unavailable, this may be caused by the same problem, but is more likely caused by the fact that I initially did not foresee that unbalanced plans would be a common occurrence, and had not started tracking them yet.

3.3 Interpretation

3.3.1 Interpretation across City Sets
Looking at the data, it seems that the algorithms with the best combination of balance and total cost are SpRrHeuGA, SpSzHeuGA, and possibly HpSzGA, though it is difficult to tell with the lack of data. One trend that is clear however is that the smaller the set of cities, the more likely one large route will visit almost all of them. SpRrHeuGA produced a Total to Max cost ratio of 1 on average on the set M2, meaning that on average it had one route visit all but two of the cities, and stationed the other two permanently at cities. It nearly did the same thing every time on XY3. In contrast, on large city sets the routes are more likely to distribute the work evenly. Every algorithm produces a reasonably well balanced plan for this city set, but the most notable is MC1: MC1 only tries to optimize the total cost through hill-climbing and has no mechanisms to direct the production of well-balanced plans. This is the same MC1 which began producing unbalanced plans before any other algorithm I tried. It has 7 routes, and the total cost to maximum cost ratio is 6.66, meaning that the cost is almost perfectly distributed between the routes, and suggesting that the key to creating balanced plans may not be in correction mechanisms, but in picking the right number of routes for a given set, as the ratio of routes to cities is significantly lower than in other tests.. This does however require knowledge of the set, which can be a problem of its own.

Ultimately, the choice of which algorithm is ideal lies in the hands of the user, who will have his or her own toleration of how unbalanced a plan can be.

Table 1: Algorithm Name Prefixes

Sp/Hp	Soft plan/Hard plan
Rr/Sz	Ratio Roulette selection/Size-based selection
Heu	Utilized Heuristic drafting
GA	Genetic Algorithm
MC	Mountain/Hill-Climbing Algorithm

Table 2: Algorithms

	Sets	Dimension of cost	Self-Travel Cost	Number of Clusters	Cities	Comments
1	XY1	$\mathbb{R}^2$	0	6	100	A set of 6 approximately equal clusters with normally distributed points around them. salesmen
3	XY2	$\mathbb{R}^2$	0.1	1	400	Large number of points generated uniformly at random on [0,2],[0,2]
4	M1	Money/ Time	0	N/A	100	Distances between points generated on [.05,.25]. d(a,b) is not d(b,a).
7	M2	Money/ Time	0	N/A	30	Smaller data set
8	XY3	$\mathbb{R}^2$	0	1	30	Smaller data set
10	TSP1	$\mathbb{R}^2$	0	1	50	The first set of points from PS1

Table 3: Experimental Results

Algorithm		TSP1 5 routes	XY1 6 routes	XY2 7 routes	M1 6 routes	M2 3 routes	XY3 3 routes
	Average Total Cost	6.40388	10.16345	12.90057	6.400703	0.990499	4.60914
	Average Max Cost	3.329641	4.512979	3.217837	3.157434	0.892947	2.922013
SpSzGA	Total/Max	1.923295	2.252049	4.009081	2.027185	1.109248	1.577385
	Average Total Cost	6.258195	8.997842	10.6932	5.768342	0.953001	4.200755
	Average Max Cost	3.644405	4.682768	2.8128	-	0.953001	4.172339
SpRrHeuGA	Total/Max	1.717206	1.921479	3.801623	-	1	1.006811
	Average Total Cost	6.32155	9.53567	10.77815	5.754412	0.986291	4.531848
	Average Max Cost	2.528371	2.675127	3.008697	-	0.626666	2.73549
SpSzHeuGA	Total/Max	2.500246	3.564567	3.582332	-	1.573871	1.656686
	Average Total Cost	6.41733	9.708967	13.26037	6.084225	-	4.470141
	Average Max Cost	4.435756	4.939646	-	3.948465	-	3.382964
SpRrGA	Total/Max	1.446727	1.965519	-	1.540909	-	1.321368
	Average Total Cost	6.5049	9.378834	12.9316	6.256445	1.027664	4.66619
	Average Max Cost	3.810352	3.853928	1.942982	2.460268	0.62628	2.928141
MC1	Total/Max	1.707165	2.433578	6.655545	2.542993	1.640902	1.593567
	Average Total Cost	26.46663	52.52583	12.99778	15.3793	2.532472	14.93759
	Average Max Cost	4.572403	-	-	3.335593	0.582543	3.745948
MC2	Total/Max	5.788342	-	-	4.610664	4.347269	3.987667
	Average Total Cost	5.829443	8.942541	12.30794	5.907503	-	4.139495
	Average Max Cost	4.577871	-	4.144159	-	-	-
HpRrGA	Total/Max	1.273396	-	2.96995	-	-	-
	Average Total Cost	6.071382	9.015548	12.28372	6.140266	0.9338	4.348048
	Average Max Cost	3.471329	-	-	-	0.851812	-
HpSzGA	Total/Max	1.749008	-	-	-	1.096251	-

3.3.2 TSP1-Specific Interpretation

We would like to consider the following graphs pertaining to the running of all algorithms on TSP1:

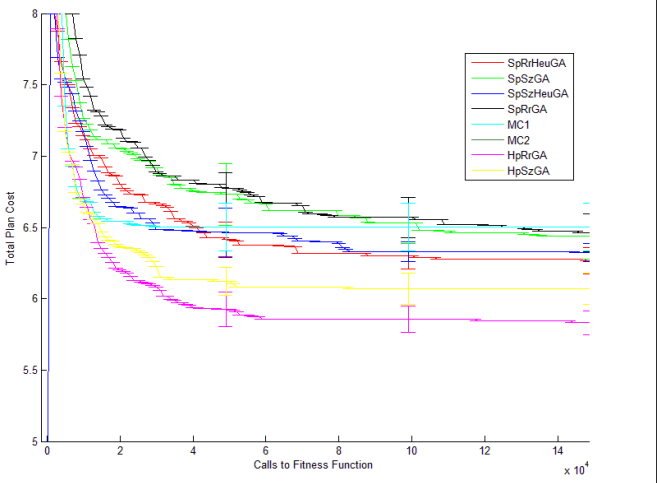

Figure 2: Total Cost over time

Figure 2 tracks the average total cost of the routes produced by each of the eight algorithms. MC2 is not pictured because its average cost was approximately 26 on this city set. If we were concerned only with minimizing the total cost, this would be the only graph necessary.

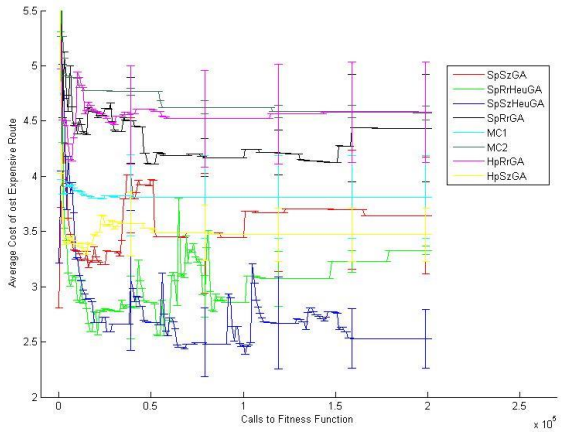

Figure 3: Average Maximum Route Cost over Time

Figure 3 is a graph of the most expensive route over time produced by each algorithm: This graph is more useful in the context of the previous graph. For instance, we can see that HpRrGA has maybe the single most expensive route at slightly over 4.5. Looking at the previous graph, we can also see that its total cost is approximately 5.8 on average for the set TSP1, which tells us that while it is cheap, it is also highly unbalanced. Neither graph could indicate this alone. Consequently, we can also guess that while SpRrHeuGA may be slightly cheaper on average than SpSzHeuGA, it is likely less well balanced.

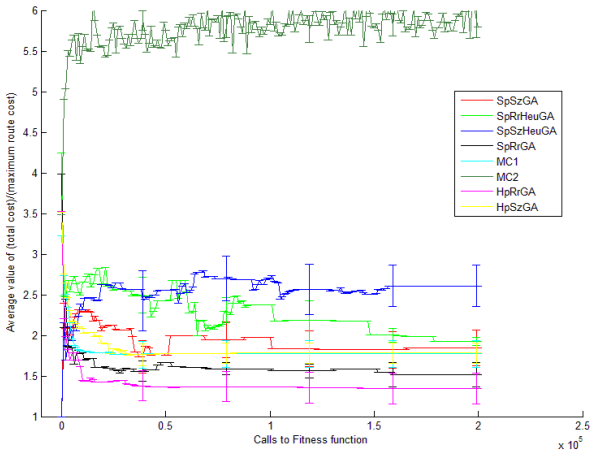

Figure 4: TotalCost/Max Cost over Time

Figure 4 is a graph which is a measure of how well balanced the plans produced be each algorithm are on average: Ideally, our algorithm would produce a graph that is slightly less than 5 over time, since there are 5 routes which the costs are distributed between. This graph also gives us a hint as to why MC2 performs so badly: the only way that the total cost over the maximum cost can be greater than the total number of routes is if the plan is incurring penalties for skipping cities. This is part of why most of my algorithms evaluate on the total cost, but focus on reducing the maximum cost.

3.4 Discussion of Results

One of the biggest problems I faced was in attempting to keep algorithms from evolving plans that had one huge routes and several small routes consisting of two or three or even one city. Having run my tests and

looked at the balance discrepancy between plans on M2 and SY2, I believe I may have an answer from Graph Theory.

Consider an optimal sales route $S_{1,1}$ on a small set of cities C which does not exhibit clustering, and suppose that for all pairs of cities c_i, c_j we have that the distance from i to j is the same as j to i. Now suppose we select one city c_i and remove it from $S_{1,1}$, making it the sole city in a new route $S_{2,2}$ and setting $S_{1,2}$ to be $S_{1,1}$ without c_1. We will assume that the distance from a city to itself is zero and that this is an improvement to the total cost of all routes, i.e. that $d(c_{i-1}, c_i) + d(c_i, c_{i+1}) \geq d(c_{i-1}, c_{i+1})$ (this is guaranteed if the cities are in a space with the triangle inequality). We now want to consider crossover between $S_{1,2}$ and $S_{2,2}$. We assume that the cities $\{c_{j+1}, c_{j+2}, \dots, c_k\}$ have been selected from $S_{1,2}$ to be inserted into $S_{2,2}$. If the set of cities $\emptyset$ has been selected to be inserted into $S_{1,2}$ from $S_{2,2}$, then this crossover will only produce an improvement if $d(c_j, c_{j+1}) + d(c_k, c_{k+1}) \geq d(c_i, c_{j+1}) + d(c_k, c_i) + d(c_{j-1}, c_{k+1})$. Intuitively, this seems highly unlikely since $S_{1,1}$ was an optimal route on C, and C does not exhibit clustering. Because C is small, it seems even more unlikely that it contains a set of points that fulfil that inequality, especially if C is in a space such as $\mathbb{R}^n$ where the triangle inequality holds. If the set of cities from $S_{2,2}$ is $\{c_i\}$ then the crossover will only produce an improvement if $d(c_j, c_{j+1}) + d(c_k, c_{k+1}) \geq d(c_j, c_i) + d(c_i, c_{k+1}) + d(c_k, c_{j+1})$, which seems intuitively unlikely without clustering for the same reasons as above.

It seems then that if two routes on a small set perform crossover and one route is left with only one city, the resulting plan will probably improve its total cause and so the change will be accepted. And if it is accepted then it will very difficult to break out of this arrangement. And since probabilistically it is only a matter of time until such an exchange occurs, we can assume that multiple routes on small sets will tend this way over time.

This is not of course a formal mathematical proof, but it is a reasonable hypothesis for the trouble I was having initially. Also in support of this is the fact that my initial test cases were all sets of 50 cities in $\mathbb{R}^2$ that did not exhibit clustering, and I typically tried them with five routes. I believe the data and my experiences support this idea, and I plan to continue testing to determine if it is likely true.

4. FUTURE WORK

There were two main weaknesses to my work. The first was time: the MTSP was not my first choice for my term project. I originally worked on the Social Golfer problem, which meant that when I discovered that it does not respond well to genetic algorithms I had lost two weeks. Fortunately the remedy to that is easy: I'm planning to continue working on this project, as I believe that this would be a good subject to expand to a senior thesis next semester. The second weakness is more project inherent that circumstantial: without some working form of external crossover, my algorithm options are severely limited. It is difficult to rank routes of a plan by anything other than ratio, and even then there are limits. For instance, if a route is bad it cannot just be dropped because since the problem is to find a set of routes for n salesmen, not $n-1$. That however would be a very interesting expansion of the MTSP: What is the optimal number and routing of salesmen if each charges d dollars an hour? Getting back to the weakness though, I have found a solution. In early-mid December I

found an article on the VRP which proposes a method of external crossover based on creating an offspring by choosing the routes with the best ratios that do not share cities with routes that have already been selected. Unfortunately due to conflicts with school and exams, I have been unable to implement it yet. I also found another simpler proposed method of crossover[5] for the MTSP earlier this week, so I am eager to try that out as well. Additionally, since seeing the result of MC1 on the set XYI want to begin experimenting varying numbers of routes on the same city set to see if that can resolve the balancing issue.

5. CONCLUSION

This project was an important stab into an aspect of the MTSP that is rarely considered: the idea of balancing the total cost of the plan across all the routes. This is unsurprising, as most people who work with the MTSP with or without genetic algorithms have a great deal of experience working with the VRP: if they need to balance the costs they can transform the project into the VRP, impose cost restrictions, and use known techniques. However, this project focused on enticing the routes to co-evolve co-operatively, rather than antagonistically and impose these restrictions without actually imposing them. This dual objective of evolving a cheap plan and a balanced one without actually imposing restrictions is uncommon in the VRP, where a frequent technique is to creatively impose restrictions that force your algorithm more quickly in the direction you want. From a practical standpoint, it is probably less efficient at creating a good solution for the MTSP than adapting some of the more advanced VRP algorithms, especially since MTSP is very close to the VRP and genetic algorithms have been shown to implement less efficiently on the VRP than more specialized algorithms[1], but the idea of imposing restrictions without actually imposing them is very interesting, and I would like to see if I can come up with another place to apply it.

6. REFERENCES

[1]Bjarnadottir, Aslaug. "Solving the Vehicle Routing Problem with Genetic Algorithms." *Solving the Vehicle Routeing Problem with Genetic Algorithms*. Informatic and Mathematical Modelling, IMM, Technical University of Denmark, 15 Apr. 2004. Web. 28 Oct. 2014. <http://etd.dtu.dk/thesis/154736/imm3183.pdf>.

[2] Chang, Yaw, and Lin Chen. "Solve the Vehicle Routing Problem with Time Windows via Genetic Algorithm." *Www.AIMSciences.org*. Discret and Continuous Dynamical Systems Supplemental, 15 June 2007. Web. 28 Oct. 2014. <https://aimsciences.org/journals/pdfs.jsp?paperID=2806&mode=full>.

[3] Kiraly, Andras, and Janos Abonyi. "A Novel Approach to Solve Multiple Traveling Salesmen Problem by Genetic Algorithm." *Computational Intelligence in Engineering*. IX ed. Vol. 313. Springer, 2010. 141-151. Print.

[4 Carter, Arthur. "Design and Application of Genetic Algorithms for the Multiple Traveling Salesperson Assignment Problem." *Design and Application of Genetic Algorithms for the Multiple Traveling Salesperson Assignment Problem*. Virginia Polytechnic Institute and State University, 21 Apr. 2003. Web. 2 Dec. 2014. <http://scholar.lib.vt.edu/theses/available/etd-04252003-123556/unrestricted/Dissertation.pdf>.

[5] Sedighpour, Mohammad, Majid Yousefikhoshbakht, and Narjes Mahmoodi Darani. "An Effective Genetic Algorithm for Solving the Multiple Traveling Salesman Problem." *An Effective Genetic Algorithm for Solving the Multiple Traveling Salesman Problem*. Journal of Optimization in Industrial Engineering, 15 Aug. 2011. Web. 17 Dec. 2014. <http://www.academia.edu/1308442/An_Effective_Genetic_Algorithm_for_Solving_the_Multiple_Traveling_Salesman_Problem>.

Acknowledgements

Thank you to my dad, who taught me to love this kind of problem.
And to my mom, who puts up with me talking about them.

Appendix

File	Description
InsertionSort.m	Implements insertion sort on the current generation of plans. Used only for external crossover, so no longer utilized often.
MasterControl.m	Runs all algorithms under user specifications on a specified city set
HpExGA.m	Runs a genetic algorithm using external crossover in the hard plan model
HpGA.m	Runs a genetic algorithm using user specified selection and the hard plan model.
MountClimb.m	Hill-climbing algorithm for total cost optimization
MountClimbBalance.m	Hill-climbing algorithm for max cost optimization
MTSPRoutes2.m	The hard plan model object class.
NextGeneration.m	Creates the next generation using external crossover and the hard plan model
NextGeneration5.m	Crates the next generation using external crossover and the soft plan model.
Plan.m	The soft plan object class
Salesman.m	The soft plan route class
SalseSizeCrossover2.m	Implements crossover using size based selection. On soft plans
SpExGA.m	Runs the soft plan external crossover genetic algorithm
SpHeuGA.m	Runs the soft plan internal crossover algorithms with heuristic drafting
SpGA.m	Runs the soft plan internal crossover algorithms
StochPointers.m	Generates pointer for stochastic universal sampling