Computer Science and Engineering
University of California at San Diego

CSE292:New Age Algorithms
Spring 94

Genetic Algorithms and the Traveling Salesman Problem

Karan Bhatia

1 Introduction

Genetic algorithms are a class of algorithms that mimic biological evolution and natural selection to "evolve" solutions that are robust and "fit". Their popularity is in part due to their perceived notion that the GA is problem independent: although the meaning of the string of bits may change, the algorithm to manipulate them does not. So by simply transforming the problem from the natural solution space (the phenotype) to the bit-string representation (the genotype), and applying a standard GA, good solutions will naturally evolve given enough time.

If this is the case, it seems only natural to use GA's to solve hard combinatorial optimization problems and, in particular, the Traveling Salesman Problem. This problem asks: given that a salesman wishes to visit N cities with the constraint that every city gets visited and no city gets visited twice, what is the minimum length tour he must take? Although this problem is conceptually easy to understand, it turns out it is very difficult to solve. With N cities, there are $N!$ number of tours the salesman can take, so doing a brute force calculation for the optimal tour quickly exceeds the computing resources of today as N increases.

The TSP problem seems ideally suited for the GA: it has a large solution space, no known polynomial solution, and a specific and easily calculated fitness function. However, after more then a decade of research, GAs still produces results that are far worse than conventional heuristics. The reason is that for the TSP, the constraints of the problem make it difficult to generate a representation of the genome for which the *building block hypothesis* holds: the standard mutation and crossover operators do not work well because the result child genome rarely produces a valid tour. Work in the last 10 years has been focused on developing new representations and new operators that produce valid tours and that build on the previous generations.

Section 2 of this paper reviews the history of the TSP and the standard heuristic algorithms used. Section 3 surveys the attempts using GA's for the TSP in the past decade. Section 4 and 5 discuss some experiments done with hybrid GAs: genetic algorithms that use local search at each generation. The results seem to indicate that while GA's in general have had

little success, GA/LS hybrids do surprisingly well, competing with the best known heuristics. Section 6 concludes with some final thoughts.

2 The Traveling Salesman Problem

The Traveling Salesman Problem is perhaps the oldest and best studied problems in an area called combinatorial optimization. It's history can be traced as far back as Euler [1759] and Hamilton [1856] who studied similar problems. But it wasn't until the 1930's that the problem got its name. At the time, researchers were studying many different optimization problems like the assignment problem and the transportation problem which looked very much like the TSP. But by the 1960's it was clear that the TSP was inherently more difficult.

Then in the early 70's, this concept of difficulty was formalized by Cook with the introduction of the theory of NP-Completeness. Any problem in this class was as difficult as any in NP (Non-deterministic Polynomial). If $NP = P$, then these NP-Complete problems did have an efficient solution, however no one has proved that this is the case. Furthermore, It is widely believed that $NP \neq P$, and the only solutions for NP-Complete problems (including TSP) are exponential-time ones.

But TSP is not only very hard, it is also very useful in areas like VLSI layout, X-ray crystallography, or job scheduling. Although optimal solutions are hard, many people have developed heuristics that find "good" sub-optimal tours. One of the reasons that TSP is an important problem is that it has provided a testing ground for new algorithmic methods such as dynamic programming, branch and bound, and more recently simulated annealing and genetic algorithms. If a new method works well on the TSP, the chances are good that that same method will work in many other problems.

However, even finding good heuristics for the TSP is not easy. Sahni & Gonzalez [1976] [1] showed that for the symmetric TSP, if a polynomial time heuristic H had a worst case upper bound, then $P = NP$. So finding a good general heuristic is as hard as finding the solution.

Another popular method for finding solutions is local search: simply start at a random solution, and if a "neighbor" of that solution (as defined by a neighborhood function) has better fitness, then replace the current solution with this neighbor, and repeat. This type of algorithm is commonly called "Hill Climbing" due to the fact that it always moves up the fitness landscape until a local optimal (a hill) is reached. Papadimitriou & Steiglitz [1977] [1] showed that using a local search procedure cannot produce tours whose length is bounded by a multiple of the optimal tour length.

There seems to be little hope for the general TSP. If, however, we add an additional constraint then we can show stronger results. This extra constraint is the triangle inequality: given any three cities, x, y, z, it must be the case that

$$d(x, z) \leq d(x, y) + d(y, z)$$

For most of the versions of TSP encountered in practice, this is a reasonable constraint.

Given this constraint, many heuristics have been developed which provide various degrees of performance.

2.1 Constructive Heuristics

The *Nearest Neighbor Algorithm* is perhaps the simplest and most intuitive heuristic for the euclidian TSP. It operates as follows: pick any starting city c_0, and repeatedly pick the ith city such that c_i is the closest unvisited city to c_{i-1}. It is clear that it will produce a valid tour, but Rosenkrantz, Stearns & Lewis [1977] showed that the quality of the tour does not have an upper bound. In other words, one can construct a TSP instance such that this procedure is arbitrarily bad. The triangle inequality does, however, limit the growth rate to $O(logn)$.

A better result can be shown for the Minimum Spanning Tree algorithm. This algorithm uses standard polynomial-time algorithms to find the minimum spanning tree (which can be found in $O(N^2)$). A lower bound on the optimal tour can be determined by observing that the optimal tour with any edge removed is a spanning tree. Therefore

$$OPT(t) > MST(t)$$

Observe that a depth-first traversal of the tree produces a tour of length at most twice the $MST(t)$. This, combined with the lower bound, gives an upper bound on the optimal tour constructed this way,

$$DFMST(t) \leq 2OPT(t)$$

where $DFMST$ is the tour obtained by the depth-first traversal.

Christofides' algorithm provides an even better worst case guarantee: $C(t) < \frac{3}{2}OPT(t)$ where $C(t)$ is the tour produced by

- construct a minimum spanning tree T
- construct a minimum matching $M*$ for the set of all odd-degree vertices
- find a eulerian tour for the graph $G = T \cup M*$.
- convert the eulerian tour into a TSP tour using "shortcuts"

For a complete description of the above algorithms and their performance see [1].

2.2 Local Improvement Heuristics

A different approach to find good tours is to start with any tour and try to iteratively improve it by making small changes. This method is called "local search" because it does not try to find the global optimal; rather it only considers tours that are the "neighbor" of the current tour, and if some neighbor exists that is "better" than the current tour, the current tour is set to this neighbor, and the process is repeated until there are no better neighbors. At this point, the tour is said to be "locally optimal."

Different local search algorithms have differing concepts of "neighbor," ranging from simple (2OPT) to very complex (Lin-Kernighan). But all these methods are very difficult to prove bound on because the number of iterations of local search is highly dependent on the initial position.

2.2.1 2-Opt

The most simplest of the local search procedures is called "2OPT." 2OPT starts with a randomly generated tour and then removes two random edges, reconnecting them so they cross. This procedure is repeated until a local optimal is found. This is called one "run." Usually many different runs, starting from different randomly generated tours, are performed and the best tour ever seen returned.

3OPT is an extension of 2OPT in which three edges are replaced with three new edges such that the tour is still valid. Experimental results [3] show that 3OPT finds much better tours than 2OPT. Both these heuristics work surprisingly well considering their simplicity.

2.2.2 Lin-Kernighan

The Lin-Kernighan heuristic extends 3OPT even more: instead of having kOPT for some fixed k, Lin and Kernighan developed a variable kOPT strategy. Given a tour t_0, a neighbor is generated by first doing a 2OPT generating t_1 and evaluating the tour. If this is better than the original, t_{best} is set to t_1. Then if t_i can be modified such that another edge can be replaced so the total cost is less than t_i, replace that edge and call the tour t_{i+1}. If t_{i+1} is better than t_{best}, reset t_{best}. Continue this process until edges can not be found to replace one in the tour. Then return t_{best}.

This process just returns the neighbor of the current tour. As with the other local search algorithms, if this is better than the current, set the current to this neighbor and repeat until a local optimal is found. Then repeat for multiple runs.

There are many details left out of the description above for the sake of readability. The actual implementation details can be found in [2]. Currently the best algorithms available for the TSP use some variation of the Lin-Kernighan heuristic. They perform very well for

practical TSP instances, performing about 2-3% above optimal for large instances. For a complete performance analysis of the heuristics see [3].

3 Genetic Algorithms for the TSP

So far, we have seen examples of two different optimization methods: what Goldberg calls random (hill climbing) and specialized (nearest neighbor, DFMST)[4]. There is another standard technique, enumeration, which for the TSP, could be brute force, dynamic programming, etc.

The problem Goldberg cites with these methods are that enumeration and random techniques perform over a large problem space, but perform uniformly badly. Specialized techniques perform much better, but only for a small class of problems. What is really needed is a "robust" scheme that works across a wide range of problems and works uniformly well.

Enter the Genetic Algorithm: search algorithms based on the mechanics of natural selection and biological genetics. It was observed that biological organisms are continually adapting to their environment, getting better and better at surviving over the generations. In terms of optimization theory, the individuals are "optimizing" to their environment. And this method is robust, it works from the deserts of the Sahara, to the mountains in Tibet.

Genetic Algorithms are different that the other techniques: GA's work with a population of individuals, and although a specific individual in the population might be unfit, as a whole, the population becomes fitter. Another difference is that GA's work with an encoding of the solutions, rather than the solutions themselves: just like the phenotype of biological organisms is encoded in the DNA as the genotype. The genetic operators then operate on this encoding.

GAs work as follows: An initial (random) sampling of solutions is chosen to initialize the population. A process of selection is carried out, where only the most fit of the population survive. The genes of these fit individuals are combined through crossover, and mutated to form the next generation. The gene is usually implemented as a bitstring. Crossover is done by selecting a random point on the genomes of the parents, and swapping the subsets. Mutation is a random bit flip. The *Schema Theorem* also know as *The Fundamental Theorem of Genetic Algorithms* shows that for the above situation, above average solutions genes will be represented in exponentially increasing numbers.

For the TSP, if we use the most natural representation, an ordered list of cities, notice that the standard operators of crossover and mutation do not always produce valid tours. One might respond "so, biological organisms do not always produce valid children." But this is very rare: in fact, most of the time, they do. For TSP, the number of possible solutions is N^N, however, the number of valid tours is $N!$. This means that the valid tours are a small subset of all possible solutions. If a GA is to work reasonably well it must weed out all

those invalid tours. This leads to the problem of representation: How does one represent the solution on its genome, such that the genetic operators use information in the parents' genes to produce "better" children with high probability? This has been the focus in GA research with respect to the TSP.

3.1 Goldberg and the PMX Operator

David Goldberg proposed the "Partially-Mapped Crossover" operator [5]: given two parent tours, say

```
A = 9  8  4  5  6  7  1  3  2  10
B = 8  7  1  2  3  10 9  5  4  6
```

the child tours are produce by picking two random numbers and swapping the cities within the bounds. Some clean up is then required because it might be the case that the child has duplicate cities. If a city is represented twice in the child (one originally, and one from the swapped substring), simply replace the first occurrence with a city that got swapped away. For example if the two random numbers are 4 and 6, the substrings to swap are "5 6 7" from A, and "2 3 10" from B, resulting in one child:

```
Child1 = 8  7  1 | 5 6 7 | 9  5  4  6
```

Now swap the other 5 with the 2, the 6 with a 3, and the 7 with 10 to get

```
Child1 = 8  10 1 | 5 6 7 | 9  2  4  3
```

which is a valid tour.

The crossing over of parents does produce children that are, in some sense, similar to their parents, but the swaping at the end introduces lots of randomness. Ideally, we would be able to take the "best" from one parent and combine it with the "best" of the other parent. "Best" for the TSP means subtours that are optimal. However, with PMX, the larger the optimal subtour, the more likely it will be broken up.

Their results described in [5] are inconclusive at best. They only give results of two different runs for a 10-city TSP instance. Their method does give good results, but with a rather large population size of 200.

3.2 Grefenstette

Grefenstette et al present a number of various representations and their results on them [9]. The first is *Ordinal Representation*, in which a tour is described by a list of N integers in

which the ith element can range from 1 to $(N-i+1)$. So, for example, the path `(a c e d b)` corresponds to the ordinal representation `(1 2 3 2 1)`.

The advantage of this representation is that standard (2-point) crossover can be used and will create valid tours. However, the child might be arbitrarily random, depending on the crossover points. The relationship to the parents is missing. Experimental results by Grefenstette show that in most cases, ordinal representation does no better than random search on TSP.

Another representation is Adjacency Representation: there is an edge in the tour t if and only if the allele in position i is j. For example the representation `(1 3 5 4 2)` corresponds to the tour `(3 1 5 2 4)`. Crossover for this representation is done by either alternating edges or by subtour chunking.

This, I believe, is moving in the right direction: ideally, if we knew which subtours of each parent were a part of the optimal tour, then we could form children such that both the subtours of each parent are present. This is important for the *building block hypothesis* which states that GA's try to "build" on the fitness of each parent. Alternating edges, however, are too fine building blocks: any optimal subtour will have a high probability of being broken. Mixing subtours should perform better. Grefenstette's results show alternating edges perform uniformly bad, as expected, but also that subtour mixing performed badly. Their hyperplane analysis showed that "there is generally no significant difference between the mean relative performance of any two competing first order hyperplanes." Which means that for this scheme, better individuals don't dominate the populations.

3.3 Whitley and Edge Recombination

Whitley's representation [8] builds on the adjacency representation of the last section. They believe that since the edges are the important component of TSP, that they should be encoded on the genome instead of the ordering of the cities. In addition, they developed a crossover operator called *Edge Mapped Crossover* which they claim transfers 95-99% of the edges from the parents to the child. The reason that this is desired is that the children of the parents should have some common traits of each parent, be them good or bad. If they are bad, the selection algorithm will weed them out. But if they are good, then the those genes are spread to the next generation.

Their crossover operates as follows: consider to tours `(a b c d e f)` and `(b d c a e f)`. If we "place" both tours on the map together, we will find that each city has between 2 and 4 edges adjacent to it. It has 2 if both tours go through the city in the same way, and 4 if they go through in completely different ways. So the edge map looks like:

```
a : b f c e
b : a c d f
```

```
c : b d a
d : c e b
e : d f a
f : a e b
```

To create the child, start at any node, say b. Now the possible choices are a, c, d, f. Notice that c, d, f all have only 2 edges (after removing the b), while a has 3 edges. So pick a city with the minimum amount of edges, and continue. So `(b c d e a f)` is a child.

The rational for this method is that those subtours that are shared between the parents will have a high probability of being transfered to the parent. The disadvantage of this method is that it is still randomly choosing between an edge of each parent, breaking up larger subtours. Although it might work better than other GA methods, it still does far worse than local search heuristics, even 2OPT. In the next section I describe some experiments where this method was compared with local search heuristics.

4 Experimental Setup

These experiments compared the performance of the Lin-Kerighan heuristic, 2OPT and the GA using edge recombination as described in [8]. Each was coded in standard C and run on Sun Sparcstations. The TSP instances were obtained form the TSPLIB instance package available over the network.

The GA used a simple selection and breeding mechanism: of the individuals in the population at generation t, selection for $t+1$ was done by eliminating the bottom $(1-s)\%$. Crossover was used to generate enough new individuals to fill the population. Selection of the parents was done at random with uniform distribution. And mutation was not introduced.

2OPT and LK were implemented as described in [2] and [1]. It should be noted that this implementation of the Lin-Kernighan did not include refinements as proposed in [2] such as lookahead, or reduction.

Fitness was computed as the sum of the squares of the distances of the cities in the tour. This was done in order to eliminate any roundoff errors that might accumulate from the square root function. The percent off of optimal will, however, be different that that reported in the literature. But since only relative performance is being judged, this is acceptable.

The goal was twofold: first, to determine how a "good" GA compares with other heuristics, and second, to see the effect of combining local search and the GA. Performance was measured in number of tour evaluations. Although this is not as accurate as CPU time, it abstracts away the differences in implementation.

4.1 GA vs. 2OPT vs. LK

The first experiment was run to gauge the relative performance of the GA, 2OPT and the LK heuristic (see Figure 1). Data was collected for the 48-city symmetric euclidean instance, with multiple runs each with different random seeds.

The GA was run with selection criteria $s = .7$, so the top 70% of the individuals from each generation would breed and continue. For each method, a maximum number of evaluations was allowed, after which the method would return its best tour found.

I first tested the GA on a 10 city instance, and found that given a population of around 200 and 200-300 generations, the GA does consistently find the optimal city. I then moved to a 48 city instance, where I tested all three methods.

As can be seen from Figure 1, the GA provides little competition for either the LK heuristic or 2OPT. As the number of evaluations is allowed to increase, the GA does do better and better. But on average, performance is still poor.

One explanation is that GAs do better when the number of generations and population size

is larger. In these graphs, I considered the low range in the number of evaluations. In the experiments by Whitley, a 50 city instance took a population of 600 and around 25,000 evaluations before their results came close to optimal. In larger TSP instances, it might be the case that GA's do better as compared with the pure local search heuristics.

4.2 GALK vs. LK

For the next experiment, I modified the GA to use the LK heuristic at each generation to optimize each individual in the population with local search. Local search was performed until the tour was locally optimal. Recombination then occured with the locally optimal parents – Lamarkian Evolution. The results are shown in Figure 2 for the same 48 city instance. The results show that GA with LK local search is much more competitive, and, in fact, consistently outperforms the random multistart LK.

If the GA did nothing at all interesting, recombining two parent tours and produced random child tours, we would expect that the GA with LK would do no worse than LK. But since it is doing better, it implies that the recombination operator is doing something interesting. Edge recombination attempts to save all edges that are common to the parents. So for a population of locally optimal tours, if there are some common optimal subtours, it is likely that they will be passed to the next generation. All other edges will essentially be shuffled. This is enough randomization to move the child tour far enough away from the parents so the local search can find different local optimal.

This idea is not new. Lin and Kernighan in [2] mentioned a optimization technique that worked very well for them. They called this method "reduction", and it worked as follows: after finding some number of locally optimal tours with random multistart LK, the initial tours would be limited to those containing the common edges of the previous tours. So their method is really doing the same thing as the GA with edge recombination. It would be interesting, to compare LK with reduction to the GALK to see which performs better.

4.3 Varying the amount of LS

After finding some good results for the GA with local search, I wanted to see how the amount of local search effected performance. To that end, I modified the GA with LK, to only perform n local search steps per individual per generation, and varied n and gaged performance on the same 48 city instance. The results are shown in Figure 3 and 4.

Figure 3 shows that as the n increases, the number of evaluations also increases, as is expected. The dotted line is for when an infinite amount of local search is allowed, ie. do local search until locally optimal.

Figure 4 shows the same graph with the z axis representing the cost of the best tour found.

What this says is very surprising: even a small amount of local search helps the GA dramatically, even as little as 2 steps! Of course this is 2 steps of LK local search, which since it is a variable KOPT algorithm, could be transforming the tour radically. But nonetheless, the effect is remarkable. By reducing the amount of local search that the GA does, it can save a substantial percentage of its evaluations, and possibly have more time to search. It should then do even better against the standard LK heuristic.

5 Conclusion

The experiments in the last section show that a GA alone is no match for even simple local search heuristics, but the GA/local search hybrid is much more powerful. Experiment 2 shows the effect of adding the LK heuristic as local search in the GA. Experiment 3 shows that not very much local search is needed to improve the GA dramatically. This implies that a GA with limited local search abilities can be better than either GA or local search alone.

Much more work is needed to adequately support the above claims. For example, how does it scale? What is the tradeoff in terms of population size and number of generations? Why is it that a little local search performs as well as lots of local search? These are questions to which I don't have answers. Much more testing and analysis is needed. But these results do indicate the importance of local search in genetic algorithms.

References

[1] Lawler, Lenstra, Kan and Shmoys, *The Traveling Salesman Problem*, John Wiley & Sons, 1985

[2] S. Lin and B.W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," *Operations Res.* 21(1973), 498-516

[3] David Johnson, "Local Optimization and the Traveling Salesman Problem," *Proc. 17th Colloquium on Automata, Languages and Programming*, Springer-Verlag (1990), pp 446-461

[4] David Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley Publishing Company, 1989

[5] David Goldberg and Robert Lingle, "Alleles, Loci, and the Traveling Salesman Problem" *International Conference on Genetic Algorithms, 1985*

[6] Kenneth De Jong, "Genetic Algorithms: A 10 Year Perspective" *International Conference on Genetic Algorithms, 1985*

[7] Prasanna Jog, Jung Suh and Dirk Van Gucht, "The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem" *International Conference on Genetic Algorithms, 1989*

[8] Darrell Whitley, Tim Starkweather and D'Ann Fuquay "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator" *International Conference on Genetic Algorithms, 1989*

[9] John Grefenstette, Rajeev Gopal, Brian Rosmaita, Dirk Van Gucht, "Genetic Algorithms for the Traveling Salesman Problem" *International Conference on Genetic Algorithms, 1985*

[10] John Holland, *Adaptation in Natural and Artificial Systems*, Mit Press, 1992