**A Study of Evolutionary Algorithms for the Degree-Constrained Minimum Spanning Tree Problem**

Rob Owens

Anthony Bloch

CSCI443

10/29/18

**ABSTRACT**

The degree constrained minimum spanning tree problem (DCMSTP) is an NP-hard problem which seeks to find the minimum spanning tree on a complete graph, whose degree does not exceed the maximum specified degree. Several different solutions are tested, and two different implementations are used, one using Prufer strings and another using an array of vertices each with an array of pointers to connected vertices and associated weights. The different operators are as follows: single mutation, triple mutation, single mutation with Prufer strings, similar alternating crossover, similar random crossover, similar random with a parent copy crossover, n-point crossover, and a greedy algorithm. Generally, the mutation evolutionary algorithms (EA) perform the best with a smaller number of vertices while the greedy algorithm performs the best on graphs with a more than 100 vertices. Of the EAs that are implemented, the mutation version, specifically the single mutation EA, performed best while n-point crossover performs the worst for this problem.

**1 Introduction**

A combinatorial optimization problem can be complex to solve, and, as the problem scales in size, the time required to find the best answer can become impractical. The need to solve such problems has inspired the development of evolutionary algorithms (EAs) which strive to find a good solution in a reasonable amount of time to a problem that would otherwise take too long to solve for a perfect solution. EAs simulate ideas inspired by biology, such as mutation, survival of the fittest, etc. to implement algorithms that can find good solutions to NP-hard problems. Put very simply, an EA generates a population of chromosomes, which are representations of possible solutions, and then operates on those chromosomes to form a new generation of chromosomes. The process is repeated, forming many generations of chromosomes, and if the operators that were used were chosen well, the fitness of the solutions that the chromosomes represent should improve as more generations are run, resulting in a good solution in the end.

Two popular operators are crossover and mutation. Crossover chooses parents from the current generation and, drawing from elements of both the parents, generates an offspring chromosome for the next generation. Mutation chooses a single chromosome from the current generation and makes a small change to generate an offspring chromosome for the next generation. The idea of survival of the fittest is often used to select parents from the current generation that will be operated on, choosing chromosomes with better fitness to be parents. This typically results in a next generation that has better average fitness than the previous generation and can often lead the way to generating a chromosome with a new best overall fitness. These are the main ideas driving EAs, but how they are implemented for a given problem can vary and thus can result in EAs that are better or worse than the next.

The degree-constrained minimum spanning tree problem (DCMSTP) is a NP-hard problem, which will be examined and a variety of EAs have been implemented to solve this problem. The results are analyzed to determine which implementations work best. A greedy non-evolutionary algorithm is also implemented, and its results are analyzed against the results of the EAs.

**2 The Problem**

DCMSTP seeks to find a minimum spanning tree (MST) on a complete graph, whose degree does not exceed a given maximum degree allowed and whose weight is minimized. DCMSTP has been shown to be NP-hard 1. Thus, as the number of vertices in the complete graph increase, the time to find the best answer increases, making DCMSTP a good candidate for EAs.

**3 Representations of the Solutions**

The first element of an EA to consider is what a chromosome looks like. A chromosome needs to represent a potential solution, and it must be possible to evaluate a chromosome for its fitness, i.e. how well it solves the problem. For DCMSTP, chromosomes must represent spanning trees (STs) on the base graph. Two representations of STs are used in the implemented solutions below. The first chromosome is an array of vertices, one for each vertex on the graph. Each vertex is defined to hold an array of pointers to the other vertices it is connected to, and a corresponding array of integers that represent the weight of the edge that connects the given vertex to the vertex whose pointer is held at the same index of the array of vertex pointers. The fitness of a chromosome is found by adding up the weights of all the edges in the ST. This is the structure used to hold the base graph on which the EAs run.

Some of the implementations also use this structure for their chromosomes. The second chromosome used in the following implementations is a Prufer string. For a graph of n vertices, a Prufer string, consisting of integers, can represent ST for those n vertices. Where the number corresponding to a given vertex appears in the Prufer string x times, the degree of that vertex in the ST is . When the fitness of the Prufer string is being evaluated, it can be decoded into a ST structure as described for the first type of chromosome, and its fitness can be evaluated by adding up the weights of all its edges. For example, if a Prufer string is 12141, it represents a tree with vertices 0 through 6. Decoding the string can reveal the structure of the ST. Vertex 0 is connected vertex 1. Vertex 1 is connected to vertices 0, 2, 4, and 6. Vertex 2 is connected to vertices 1 and 3. Vertex 3 is connected to vertex 2. Vertex 4 is connected to vertices 1 and 5. Vertex 5 is connected to vertex 4. Vertex 6 is connected to vertex 1. While the ST structure is used in evaluation of the Prufer string’s fitness, only the Prufer string needs to be held in memory from generation to generation of the EA. In the implementations of the EAs, a Prufer string is defined as a structure containing an array of integers to represent the string itself and an integer representing the fitness of the Prufer string. Thus, its fitness needs only be calculated once, not each time it is referenced.

**4 Non-Evolutionary Heuristic**

A greedy algorithm is implemented to set a baseline against which the EAs are compared. A greedy algorithm is an algorithm that builds its solution one step at a time, making the most beneficial choice it can make at each given step, ignoring what happens in the other steps. For DCMSTP, each step is comprised of several sub-steps. First, the graph is searched for the shortest edge. If there are more than one shortest edge, one is selected randomly. The ST, which should be initialized to be empty before the first step is begun, is checked to determine if adding the selected edge would create a cycle or would violate the degree constraint. If either of these conditions are met, the graph is searched for the next shortest edge. Once a valid edge is found, it is added to the ST and removed from the base graph. This step is performed times where n is the number of vertices in the base graph to form a ST of edges.

**5 EAs: Crossover**

Crossover is one of the two primary operators used in genetic algorithms. Crossover, put simply, involves choosing chromosomes from the parent generation and creating an offspring chromosome for the next generation using elements from each parent. Several forms of the crossover operator itself are implemented for DCMSTP, but the EAs using crossover are similar otherwise. To begin, a population of chromosomes represented by Prufer strings are randomly generated. The next generation is formed by selecting parent chromosomes through a simple tournament selection and then using a crossover operator on the chosen parents to create an offspring. As a chromosome is created, its fitness is compared against the fitness of the current best solution. If it is better, that chromosome is saved as the current best solution. The initial best fitness is set equal to where n is the number of vertices in the base graph and where m is the max weight an edge can be. Each time a new generation is begun, the current best solution is copied as the first chromosome in that generation’s population. This ensures that the current best chromosome of any generation is never worse than the previous generation. The algorithm runs until the fitness of the current best solution becomes stale, i.e. it stops improving over the course of a given number of generations.

Four crossover operators are implemented. The first (crossover 0) examines the parent Prufer strings element by element, copying any element that is similar between the parents to the corresponding element of the offspring. Next, the remaining elements are copying from one parent or the other, alternating which parent is referenced from one element to the next. If adding the given element from the chosen parent would violate the degree constraint, a random number is assigned to that element so that the degree constraint is met. The second crossover operator (crossover 1) is similar to the first. It copies the elements that are similar in the parents to the offspring and then, for the remaining elements, assigns a random number while still observing the degree constraint. The third crossover operator (crossover 2) is similar to the second, but, after creating a new offspring, one of the parent’s chromosomes is copied over as a whole to the new generation. Thus, there are only half as many new chromosomes each generation. The final crossover operator (crossover 3) is n-point crossover which divides the offspring chromosome into p sections where p is the number of parents it has. For each section, the corresponding elements of a given parent are copied. If the degree constraint would be violated, a random number meeting the constraint is assigned instead of copying the parent.

**6 EAs: Mutation**

EAs using mutation are also implemented. The mutation operator involves selecting a chromosome from the parent population and copying its elements to the offspring chromosome, while making a small change to its elements2. The tree structure is used in the implementation of the mutation EA. For this type of chromosome, mutation involves copying the ST of the parent to the offspring, removing a random edge from the offspring’s ST, and then adding a random edge so that the degree constraint is met, no cycles are created, and a connected ST is formed that includes all the vertices from the base graph. Two implementations are used to generate similar EAs that differ only in the number of edges that are changed (single mutation and triple mutation).

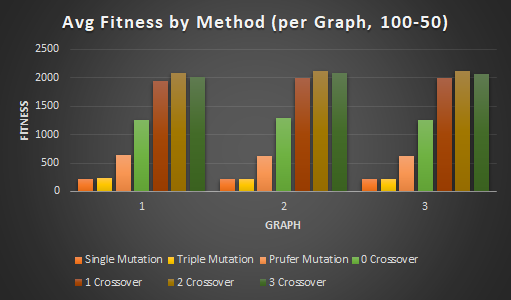
Prufer strings are used for chromosomes in another implementation of a mutation EA. For these, the mutation operator involves copying the elements of the parent Prufer string to the offspring and changing one element to a random number while observing the degree constraint. As in the crossover algorithms, an initial population is generated, and the current best solution is tracked. A parent is chosen through simple tournament selection, and the mutation operator is applied to form the offspring chromosome. The fitness of each new offspring is compared to the current best solution which is updated if a better solution is ever found. To ensure the current best solution of a given generation is never worse than the previous, the best solution is copied as an offspring in each new generation. The mutation EAs run until they become stale.

**7 Testing Procedure and Results**

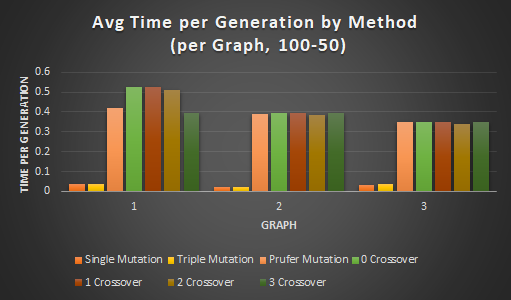
For each of the operators described above, the population sizes are kept the same between each run, with a population size equal to the number of vertices in the graph, and the staleness factor is the same needing 100 generations of having the same best chromosome before ending. Each test is run on a complete graph whose edges’ weights are generated randomly. There are three different levels of vertices tested: 20, 50, and 100. Then at each level, nine different graphs are generated and separated into three sets. The first set has a max degree equal to 10% of the total number of vertices, the second set has a max degree equal to 25% of the number of vertices, and the final set has a max degree equal to 50% of the number of vertices. The methods tested are a single mutation EA, a triple mutation EA, a Prufer mutation EA, an EA for each of the four crossover operators, and finally a greedy algorithm. Each method is run 30 times on each graph to provide ample data and illuminate trends. The following table lists the best scores obtained on each graph by each method and the mutation and crossover operators which provide the best outcome. The run column is formatted (vertices in graph - max degree - graph number) for identification purposes.

|  |  |  |  |
| --- | --- | --- | --- |
| Run | Mutation Best | Crossover Best | Greedy Best |
| 20-3-1 | 45 (Triple) | 204 (0) | 64 |
| 20-3-2 | 49 (Single) | 250 (0) | 68 |
| 20-3-3 | 48 (Triple) | 246 (0) | 54 |
| 20-5-1 | 50 (Single) | 231 (0) | 74 |
| 20-5-2 | 47 (Single) | 243 (0) | 76 |
| 20-5-3 | 51 (Triple) | 206 (0) | 50 |
| 20-10-1 | 49 (Tie S & T) | 254 (0) | 64 |
| 20-10-2 | 44 (Single) | 270 (0) | 89 |
| 20-10-3 | 41 (Single) | 225 (0) | 58 |
| 50-5-1 | 92 (Triple) | 857 (0) | 89 |
| 50-5-2 | 80 (Triple) | 844 (0) | 82 |
| 50-5-3 | 89 (Single) | 815 (0) | 84 |
| 50-12-1 | 93 (Triple) | 809 (0) | 78 |
| 50-12-2 | 90 (Single) | 699 (0) | 83 |
| 50-12-3 | 85 (Single) | 847 (0) | 92 |
| 50-25-1 | 90 (Single) | 838 (0) | 81 |
| 50-25-2 | 79 (Single) | 855 (3) | 91 |
| 50-25-3 | 85 (Single) | 812 (0) | 93 |
| 100-10-1 | 163 (Single) | 1963 (1) | 125 |
| 100-10-2 | 159 (Single) | 1876 (0) | 112 |
| 100-10-3 | 166 (Tie S & T) | 1934 (0) | 120 |
| 100-25-1 | 164 (Single) | 1080 (0) | 112 |
| 100-25-2 | 160 (Triple) | 1061 (0) | 127 |
| 100-25-3 | 163 (Single) | 1063 (0) | 125 |
| 100-50-1 | 161 (Single) | 1113 (0) | 105 |
| 100-50-2 | 159 (Triple) | 1129 (0) | 114 |
| 100-50-3 | 164 (Single) | 1085 (0) | 117 |

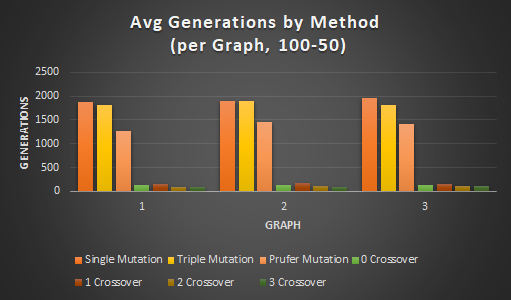
The following charts compare average statistics for each EA, with average ending fitness, average time per generation, and average number of generations run. All graphs are based on data collected from runs with 100 vertices and a max degree of 50.



**Figure 1**: The average best fitness for each EA on three different graphs with 100 vertices.



**Figure 2:** The average time in seconds to run one generation for each EA on three graphs with 100 vertices.

**Figure 3:**  The average number of generations ran by each EA for 3 graphs with 100 vertices.

**8 Analysis**

Overall, the best performing method for mutation based on the best achieved score was the single mutation, having beaten triple mutation and Prufer mutation 17 times out of the 27 instances. On the other end of the spectrum the Prufer form of mutation did not beat either of the tree structure-based implementations. In fact, there was not a single instance where the Prufer implementation beat the tree structure implementation for the best fitness achieved. However, as shown in Figure 2, the average time per generation for the Prufer string mutation was significantly longer than the average time per generation for either of the tree structure mutation EA. In terms of single versus triple mutation, the triple mutation on average took less generations before completing, while producing chromosomes with a similar fitness. A potential strategy to save time and generations as the number of vertices increases may be to run the EA as a triple mutation initially until it hits its end and then change to a single mutation seeded with the chromosomes found by the triple mutation. At the 100 vertices level, this would save about 100 generations at least for graphs with more than 100 vertices. This would potentially save time as the EA will make bigger leaps, however performing three mutations, is still small enough that it doesn’t make radical changes to the tree at each step allowing an offspring to look very similar to its parent.

Of the four crossover operators, crossover 0 produced the best solution, beating the other crossover operators every time. As shown in Figure 1, crossover 0 averaged about 150-200% better than the next crossover operator. In Figure 2, its shown that even though it is far better than the other crossover operators, the time to run each generation was almost on par with all the other crossover operators. The rest of the crossover operators performed very similar to each other, with crossover 1 producing the best results alongside crossover 2 and leaving crossover 3 as the worst. As expected, crossover 1 and crossover 2 generally performed about the same, with crossover 1 having a better average fitness than crossover 2. Shown in Figure 3, all the crossover operators ran a generally small number of generations in comparison with the mutation operators. This could be due to a limited number of potential edges that can be added, or perhaps the crossover operator couldn’t change enough in each generation to explore different potential solutions.

For the debate of crossover versus mutation, these instances show that for DCMSTP, mutation is generally better. Parameter values for mutation and crossover are controlled such as differences in population size, the staleness factor, and the number of runs for each method. The data collected shows that Prufer strings do not make good chromosomes for crossover operators, because of the nature of Prufer strings. Prufer strings comprised of the same numbers, but in different order, can represent very different STs. The crossover operators modify the Prufer strings, often changing many elements, which make larger changes in the resulting ST. Thus, while the parents may be good solutions, the child ST is so different that it is unlikely to be better. Another potential reason that the crossover operators lagged behind the others is that in a Prufer string crossover, what matches may not be the best parts of the tree, and especially early on before the chromosomes have converged, there may be no matches, meaning the whole tree could be randomized at which point there isn’t really any advancement if new trees are just being made randomly.

When compared with the greedy algorithm, the mutation operator generally beat the greedy algorithm at smaller number of vertices. After the number of vertices went above 100, the EAs fell behind the greedy algorithm by a large margin. This trend could be due to mutation being able to search most if not all the possible solutions for smaller numbers of vertices; however, when the number of vertices grew, the EAs were more likely to get stuck at worse points. In the future this could be solved by having a larger population size, increasing the staleness factor, or doing more runs. This would decrease the chance of an EA getting stuck at a chromosome that, compared to those around it, is good but, overall, is bad.

**9 Conclusion**

EAs are implemented to solve DCMSTP for a good solution. As the results show, the mutation operator performed the best of the EAs, and it performed better than a greedy algorithm for graphs with vertices less than 100. For the mutation operators, the single mutation operator achieved the best results, with triple mutation being a close second. For the crossover operators, crossover 0, which copies the similarities between the parents to the child and then copies the remaining elements from a single parent alternating the chosen parent every other element, performed the best by a fair margin, and then similar random and similar random with parent copy, and finally n-point crossover was the worst in this case. Prufer strings do not seem to be good chromosomes for EAs for DCMSTP. DCMSTP seems to lend itself to mutation based EAs and greedy algorithm solutions for general heuristic solutions.

**REFERENCES**

[1] Kai Shi, Qingfeng Song, Sheng Lin, Guangping Xu, and Zhanxu Cao (2016). An Improved Genetic Algorithm for Degree Constrained Minimum Spanning Trees. 2016 28th Chinese Control and Decision Conference (CCDC).

[2] Nacira Ghoualmi-Zine, Rachid Mahmoudi (2010). Crossover and Mutation Based Cloning Parent for Degree Constrained Minimum Spanning Tree Problem (d-MSTP). " *2010 Second International Conference on Engineering System Management and Applications*, Sharjah, 2010,

[3] A.E. Eiben and J.E. Smith (2003). Introduction to Evolutionary Computing. Second Edition. Springer, New York, 2015.