AI Methods for Finding Degree-Constraint Minimum Spanning Trees

CSM6120 Essay by Morgan Jones (mwj7@aber.ac.uk)

# Introduction

This essay presents and compares current AI approaches to solving the Degree-Constrained Minimum Spanning Tree (DCMST) problem. I begin by introducing the problem definition and then proceed to present and compare various AI algorithms that solve it. Finally, I conclude my findings and state, along with motivation, my chosen approach to tackling the problem.

I used Google Scholar to find recent cited articles that define solutions to the DCMST. I read some of these articles and compared the background sections of the papers to piece together a history/timeline of DCMST solutions. I searched for an API for solving DCMST instances and found a GitHub project that referenced papers that solve the problem in different ways, some of which I had already read. I focused on frequently occurring papers and chose the most recent and well cited approaches to compare; one from each type of AI approach. {Ant Colony Optimization, Genetic Algorithms, Simulated Annealing & Multi-Start Hill Climbing}

# The Degree-Constrained Minimum Spanning Tree Problem

The goal of the DCMST problem [1] is to find a Minimum Spanning Tree (MST) [2] of an asymmetric, unweighted and complete graph; such that the MST does not have a degree d 2 on any of its vertices. The degree of a vertex is the number of edges attached to that vertex (incident edges). The problem is NP-Hard as shown by Garey and Johnson (1979) [3] through a reduction to an equivalent symmetric TSP.

Despite being difficult the DCSMT is a problem worth studying because many real-life applications require a connected network that is subject to a degree-constraint. <Examples here..> There is an exact branch and bound solution to the DCMST based on the lagrangean relaxation approach to solving the TSP [4]. DCMST is defined by the following objective function and constraints:

|  |  |  |
| --- | --- | --- |
| Minimise |  | Minimise the sum of edge weights included in the spanning tree |
|  |  | The number of edges on each node is no more than the degree. |
|  |  | The number of edges on each node is no less than one. |
|  |  | The number of edges in the MST is no more than one less than the number of vertices in the MST. |

Notation:

* i,j = Nodes in the graph.
* Bi = Number of incident edges on a vertex
* Cij = The weight of the edge connecting i and j.
* Xij = 1 if the edge eij is included in the MST, 0 otherwise
* V = Set of all vertices in the graph
* N = Set of vertices included in the MST

# AI Approaches for Computing Degree-Constrained Minimum Spanning Trees

## Ant-Based Optimisation (AB-DCST)

An Ant-Based optimisation (ABO) algorithm AB-DCST by Thang N. Bui, Xianghua Deng, and Catherine M. Zrncic [4]. ABO algorithms are inspired by Ant Colony Optimization (ACO) <REF> but differ by only using the ants to narrow the search space to a promising area; that is, instead of find solutions. Then other methods are used to derive a solution from the narrowed search space.

In AB-DCST the pheromone level on edges, as result of the ants’ exploration, are used to find a candidate set of edges from which to construct spanning trees. The constructed trees are then optimised by two optimisation functions: 2-Edge Replacement and 1-Edge Replacement. The adding of the optimisation phase is the reason for the ‘Improved’ AB-DCST as opposed to their original paper [5].

Each iteration of the AB-DCST algorithm is defined by the following phases:

**Initialisation Phase:** An ant is placed on each node and each edge is set an initial pheromone level proportional to its weight such that those edges with smaller weights will have higher initial pheromone. *IP(e)=[(M − w(e)) + (M − m)/3]*

**Exploration Phase:** Ants move in parallel around the graph laying pheromone trails as they go. The ants move for a specified number of steps. For efficiency pheromone trails are updated only after a fixed number of steps.

Ants choose which incident edges to traverse based on a probability proportional to the pheromone level of each edge incident to an ant’s current location. After an edge has been traversed it is scheduled to have its pheromone level increased by its initial pheromone level (If 5 ants traverse an edge its pheromone will be increased by 5 x e.initialPheromone). Therefore, lower cost edges and more frequently traversed edges are more likely to be selected for traversal.

To prevent early convergence on local optima:

* Each ant keeps a Tabu List of recently visited nodes which they are not allowed to revisit.
* Pheromone trails evaporate periodically via a time decreasing evaporation rate.
* Pheromone levels are bounded to the range:  
     
  If the new pheromone level of an edge would take it over a boundary the pheromone is only adjusted by +/- e. initialPheromone.

**Construction Phase:** Edges are sorted in order of decreasing pheromone level and n candidate edges selected. Candidates sorted in order of increasing edge cost and then used to construct a spanning tree using Kruskal’s algorithm [6] with a modification to obey the degree constraint.

**(Local) Optimization Phase:** The algorithm tries to optimise the spanning tree by replacing 2 edges in the tree such that the degree constraint is maintained but the cost of the tree is smaller. The algorithm then tries to do the same for just 1 edge. These optimisation functions are called 2-Edge Replacement and 1-Edge replacement respectively.

After each phase of an iteration is completed there are 5 **post** **processing** steps:

1. The resulting tree and best are compared with the lowest cost tree being kept.
2. Pheromone levels of the edges in the best tree are enhanced by an enhancement factor.
3. The algorithm runs a restart algorithm if stuck in local optimum (no improvement in best tree after set number of iterations).
4. Ants are reset ready for the next iteration half the ants retain their current position with the other half are placed at random vertices.
5. Enhancement factor (γ) & evaporation factor (η) are updated. \*=1.05 and \*=0.95 respectively.

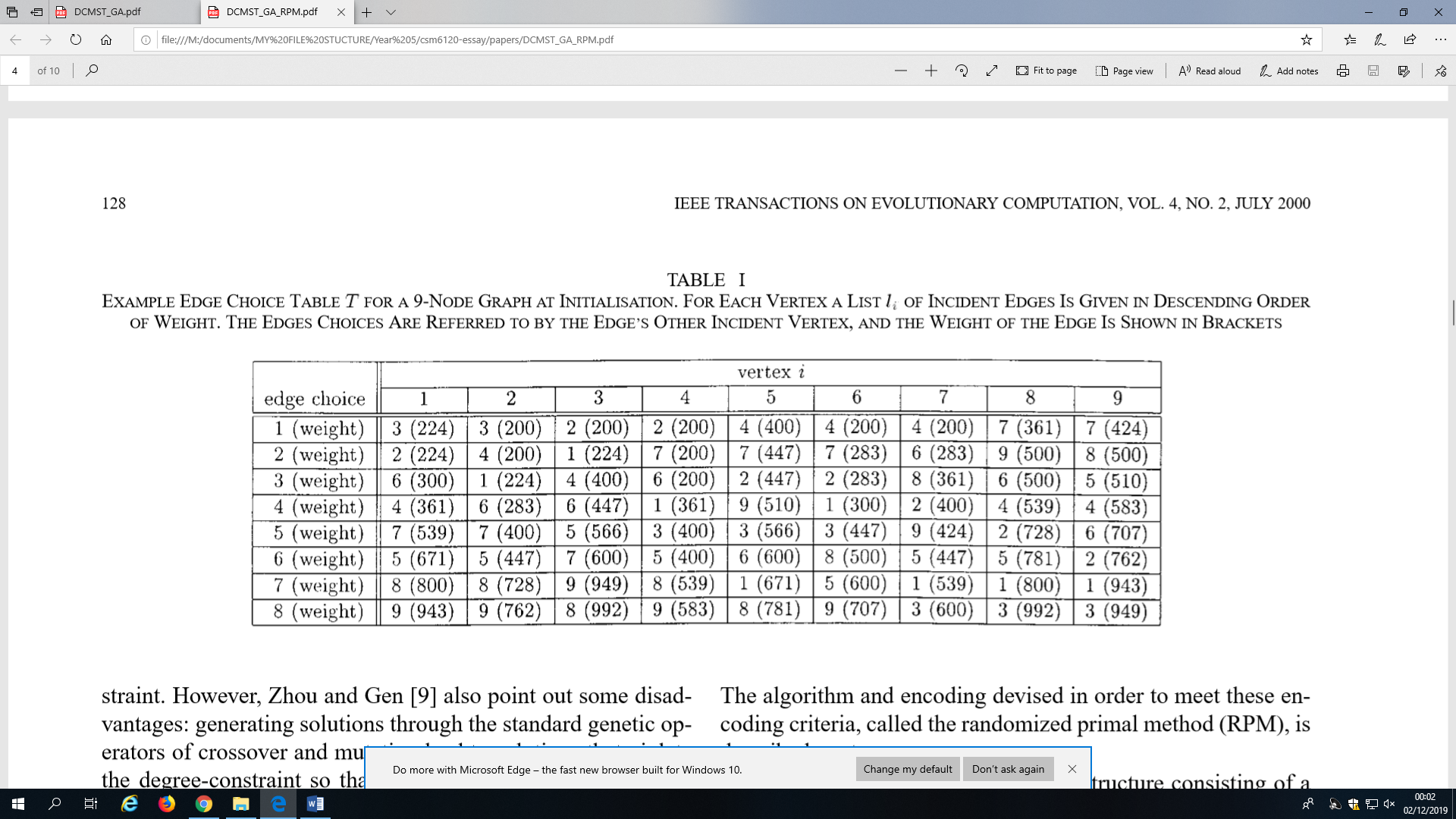
## Genetic Algorithm (RPM)

The original idea to use a Genetic Algorithm (GA) <REF> to solve the DCMST was by Zhou and Gen [7], they used standard µ+λ GA <REF> along with Prüfer numbers [8] to encode the spanning trees. Prüfer encoding is useful because every Prüfer number represents a valid spanning tree and any vertex with degree d will appear in the representation exactly d-1 times, making it easy to spot degree violations.

Prüfer encoding also has drawbacks. Modification of Prüfer numbers is required after the application of genetic operators because crossover and mutation result in solutions that violate degree constraint. Another issue is similar Prüfer numbers can represent vastly different spanning trees resulting in a GA liable to drift rather than converge within the search space.

An improvement to the use of Prüfer encoding is Joshua Knowles‘ and David Corne’s Randomised Primal Method (RPM). RPM is an algorithm for decoding a solution vector chromosome into a valid degree constrained spanning tree. It can be applied to stochastic iterative search techniques, in their paper they use it as the solution representation for three: A Genetic Algorithm, Simulated Annealing and Multi-Start hill climbing.

RPM is a modification to d-Primm’s (Primm’s algorithm [9] for computing an MST but with a modification to not violate degree constraint) that uses a dynamic tabular data structure T. The table T holds an edge-list entry for each vertex in the graph ordered by increasing edge weight.



In Primm’s when choosing the lowest weight edge to add the tree, if using the suggest table data structure, a low-weight edge set L would be created from the first in each vertex edge-list and then the cheapest edge of the set L would be added to the growing spanning tree. This is how Knowles and Corne implemented the d-Primm’s algorithm they used for comparison.

In the RPM instead of choosing the 1st edge from each edge-list to form the low-weight edge set L the th edge is chosen from appropriate entry in T. Where the value for is predetermined by the corresponding value in a tabular (2D) solution vector chromosome that is passed in as input to RMP. Where is the ith vertex and is the current degree at that vertex in the growing tree.

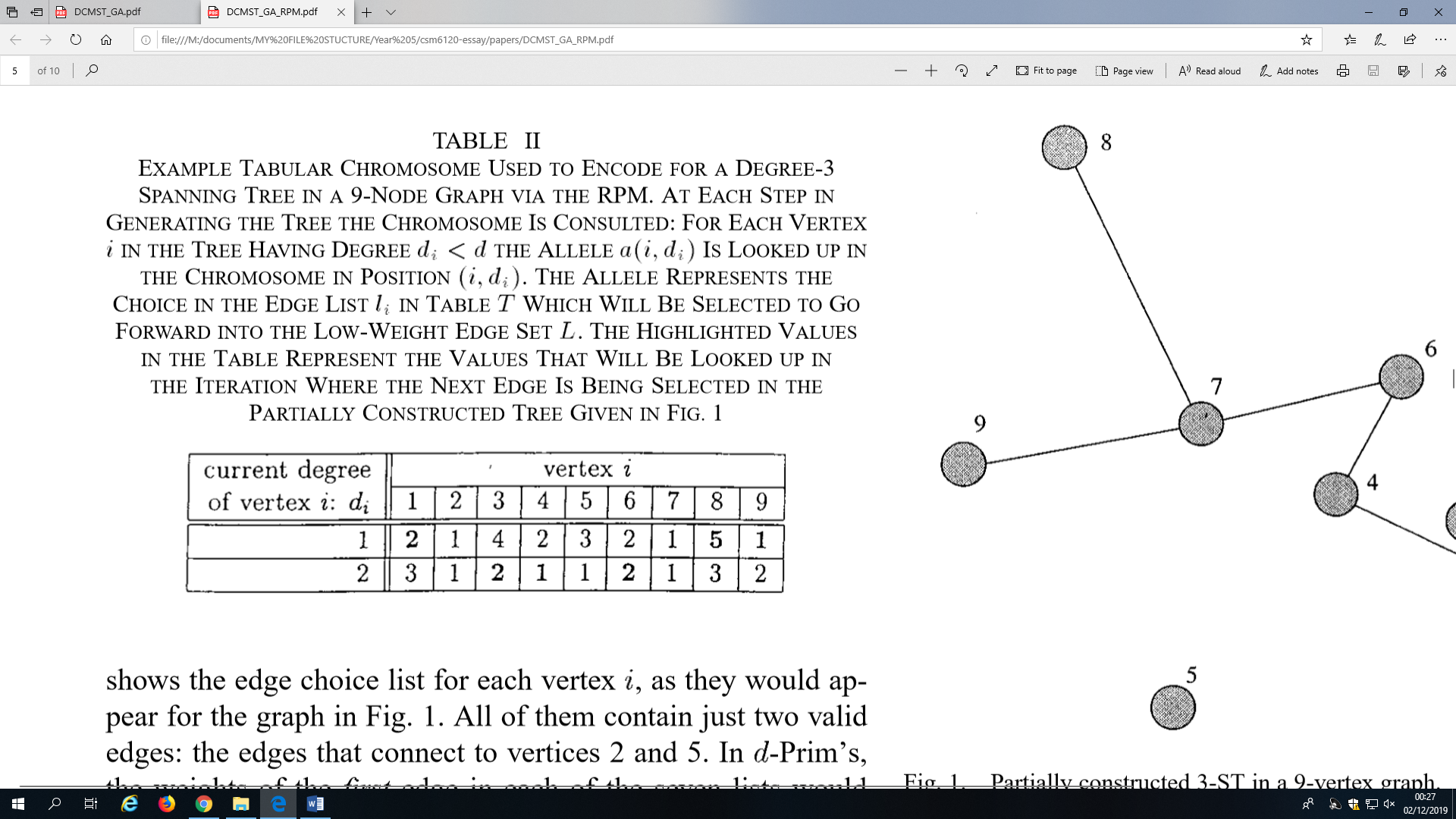


Figure 1 - Example of Tabular Chromosome in RPM

In their implementation the values of (alleles) in the chromosome are randomly initialised at the start of each iterative search algorithm with a bias towards smaller values. Meaning, towards the start of the algorithm the RPM is choosing the lowest edges in the edge-lists nearly all the time.

RPM dominates Prüfer encoding because it does all Prüfer encoding does and more. This is summarised by the following superset of RPM properties:

1. RPM encodes only valid trees;
2. RPM encodes only tress which meet the degree-constraint;
3. Similar representations (solution vectors) correspond to similar trees with common edges.
4. Edge cost information is used so high-cost solutions are rarely generated.

This is also shown in the paper where Knowles and Corne demonstrate that a GA employing RPM is superior to a GA employing Prüfer numbering. They present the results of the 9-node problem instance which Zhou and Gen [7] originally used to demonstrate the quality of their GA.

The optimal solution is 2256 when the degree-constraint is 3. The original GA found the optimal solution 66.7% of the time in 25,000 evaluations. The GA with RPM found the optimal solution 99% of the time in 500 evaluations.

The GA itself is a µ+λ GA with one crossover and one mutation performed per generation. A population size of 255 chromosomes are mapped onto a 2D grid (15 x 15). A local mating selection [10] is used to help prevent early convergence meaning only chromosomes near to each other in the grid can mate. Selection for mating is achieved by first choosing a random grid square. Each parent is then the fittest individual found along a random three step walk, performed each time (k times) from the starting square.

## Simulated Annealing

Simulated Annealing (SA)

## Multi-Start Hill Climbing

Multi-Start Hill Climbing (MHC) is a search method that starts at a random point in search space and greedily moves to the best state in its local neighbourhood. If MHC has failed to improve in a given number of iterations it simply restarts its ‘climb’ from another random point in search space. With the best evaluation found so far being continually stored and returned at the end of the algorithm. The algorithm iterates until some stopping criteria is met. Multiple restarts provide a benefit over the standard Hill Climbing algorithm because they allow MHC to escape from local optima.

# References

|  |  |
| --- | --- |
| [1] | S. C. Narula and C. A. Ho, “Degree-Constrained Minimum Spanning Tree,” *Computers and Operations Research,* vol. 7, pp. 239-249, 1980. |
| [2] | E. Horowitz and S. Sahni, “Minimum spanning trees,” in *Fundamentals of Computer Algorithms*, Potomac, Md, Computer Science Press, 1978, pp. 174-183. |
| [3] | M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, San Francisco, CA: Freeman, 1979. |
| [4] | T. N. Bui, X. Deng and C. M. Zrncic, “An Improved Ant-Based Algorithm for the Degree-Constrained Minimum Spanning Tree Problem,” *IEEE Transactions on Evolutionary Computation,* vol. 16, no. 2, pp. 266-278, 2012. |
| [5] | T. N. Bui and C. M. Zrncic, “An Ant-Bsed Algorithm for Finding Degree-Constrained Minimum Spanning Tree,” *Proceedings of the 8th annual conference on Genetic and evolutionary computation,* pp. 11-18, 2006. |
| [6] | T. H. Cormen, C. E. Leiserson, Rivest, R. L and C. Stein, Introduction to Algorithms, 2nd ed., Cambridge, MA: MIT Press, 2001. |
| [7] | G. Zhou and M. Gen, “A Note on Genetic Algorithms for Degree-Constrained Spanning Tree Problems,” *Networks,* vol. 30, no. 2, pp. 91-95, 1997. |
| [8] | J. W. Moon, “Counting Labelled Trees,” in *Associating sequences with trees*, London, William Clowes and Sons, Limited,, 1970, pp. 4-8. |
| [9] | R. C. Prim, “Shortest Connection Networks And Some Generalizations,” *Bell System Technical Journal,* pp. 1389-1401, 1957. |
| [10] | R. J. Collins and D. R. Jefferson, “Selection in Massively Parallel Genetic Algorithms,” in *4th Int. Conf. Genetic Algorithms*, San Mateo, CA, 1991. |