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 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 algorithms to compare.

# The Degree-Constrained Minimum Spanning Tree Problem

The DCMST is a combinatorial optimization problem within graph theory. 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 [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 formally defined by the following objective function and constraints:

|  |  |  |
| --- | --- | --- |
|  | | 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. |
| Where is the set of all vertices in the graph, is the set of vertices included in the MST, and are vertices in the graph, the number of incident edges on a vertex , the weight of an edge connecting and , is 1 if the edge is included in the MST or 0 otherwise. | | |

# AI Approaches for Computing Degree-Constrained Minimum Spanning Trees

## Ant-Based Optimisation

An Ant-Based optimisation (ABO) algorithm AB-DCST by Thang N. Bui, Xianghua Deng, and Catherine M. Zrncic [5]. ABO algorithms are inspired by Ant Colony Optimization (ACO) [6] 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 [6].

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 [7] 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 an iteration is completed there are 5 **post** **processing** steps:

1. The resulting tree and current 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 & evaporation factor are updated (\*=1.05 and \*=0.95 respectively).

## Genetic Algorithm

The original idea to use a Genetic Algorithm (GA) <REF> to solve the DCMST was by Zhou and Gen [8], they used standard µ+λ GA [9] along with Prüfer numbers [10] 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 that similar Prüfer numbers can represent vastly different spanning trees resulting in a GA liable to drift rather than converge within the search space. The following subsections detail incremental improvements to a GA approach through use of alternate methods for tree representation within the GA.

### Randomised Primal Method

An improvement to the use of Prüfer encoding is Joshua Knowles‘ and David Corne’s Randomised Primal Method (RPM) [11]. 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.

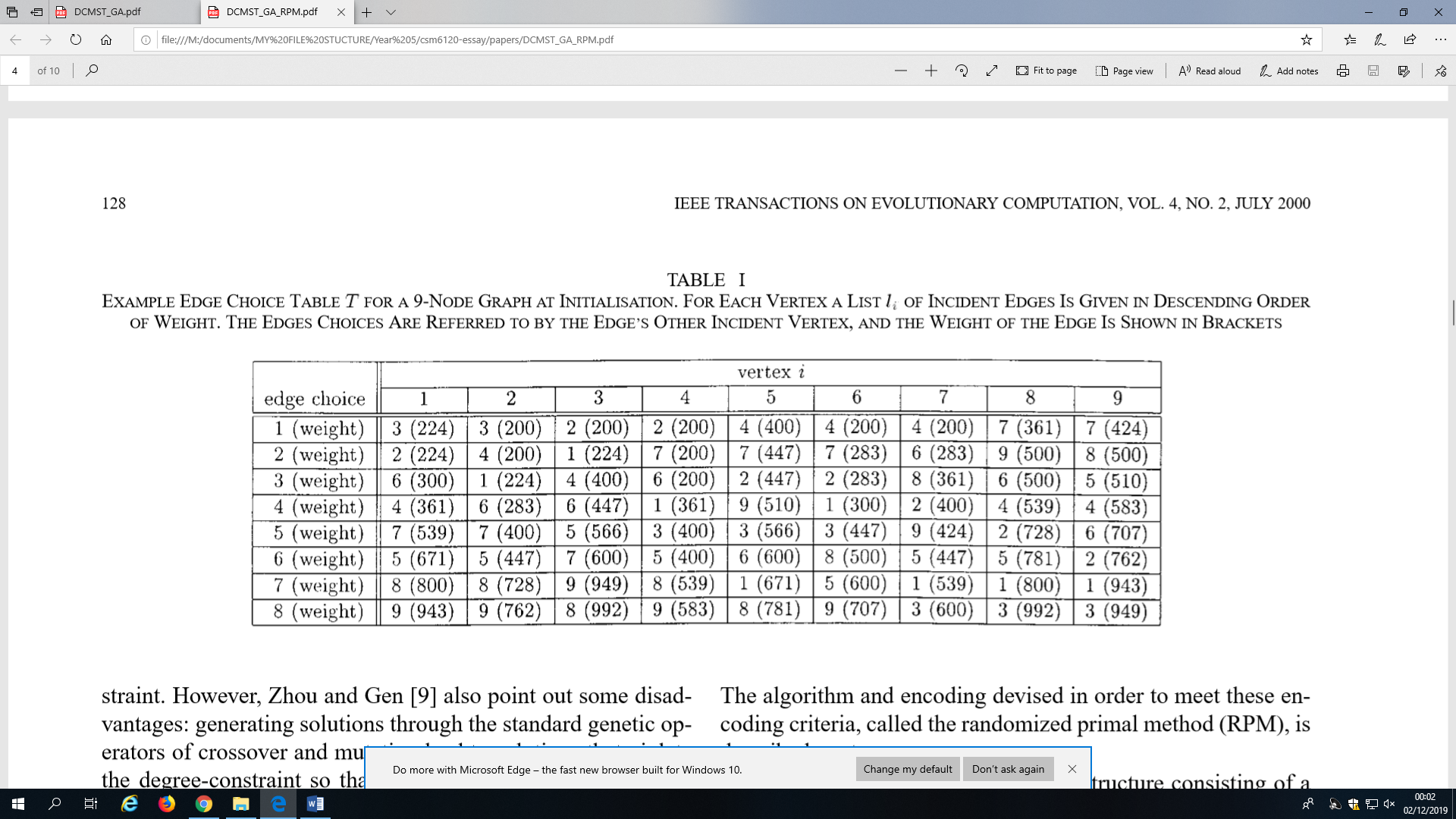
RPM dominates Prüfer encoding because it does all Prüfer encoding does and more. This is summarised by the following superset of RPM properties:

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 [8] 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.

RPM is a modification to d-Primm’s (Primm’s algorithm [12] 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 suggested 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.

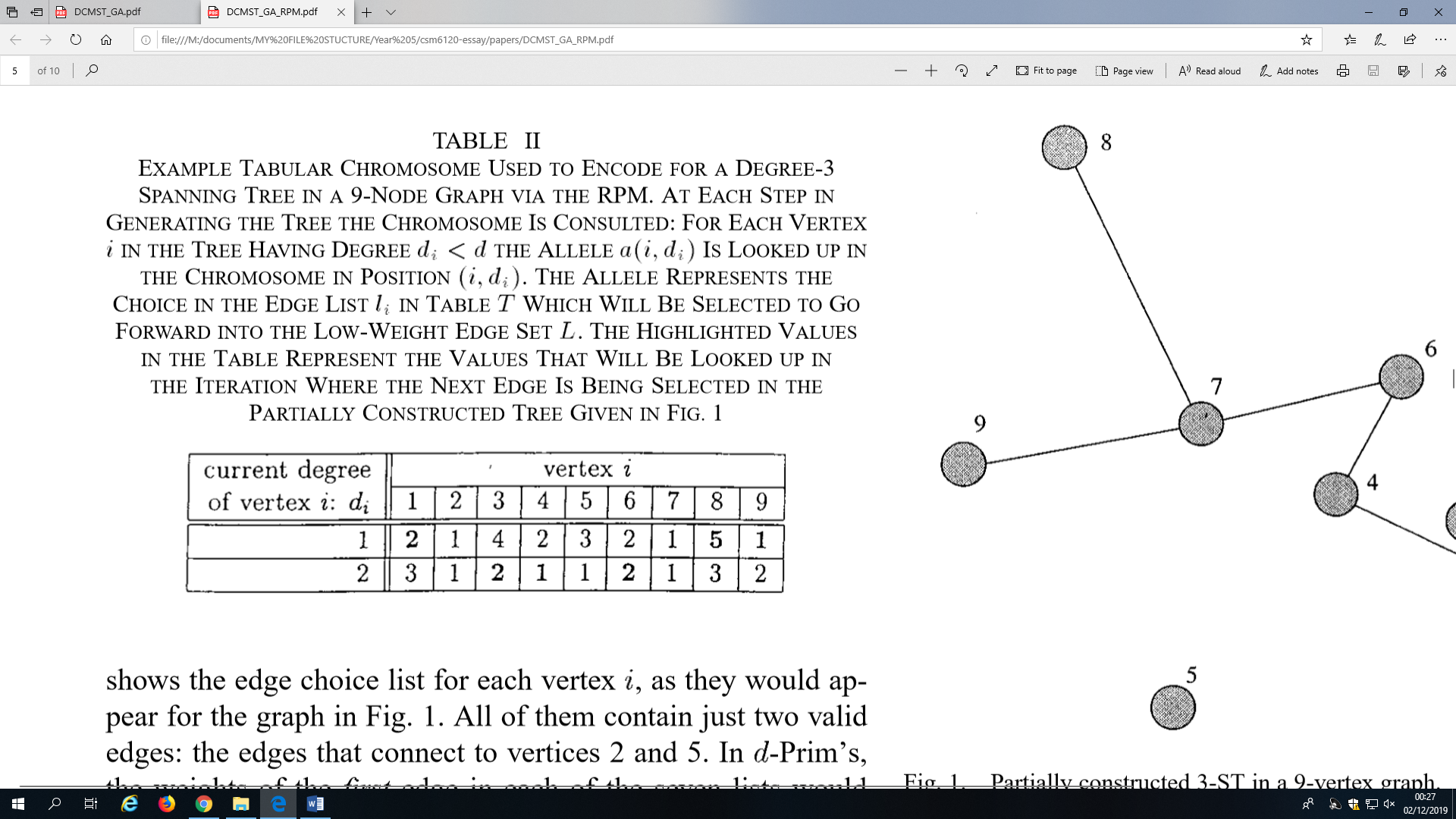


Figure - Example of Tabular Chromosome in RPM

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.

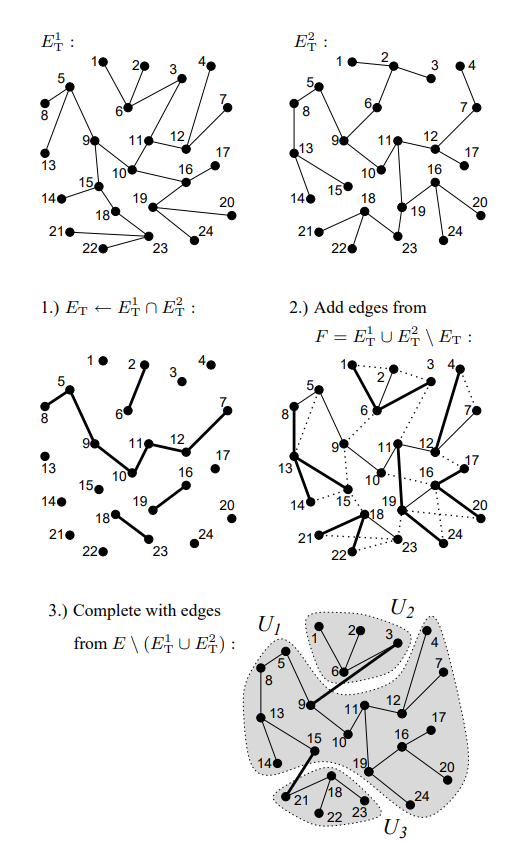
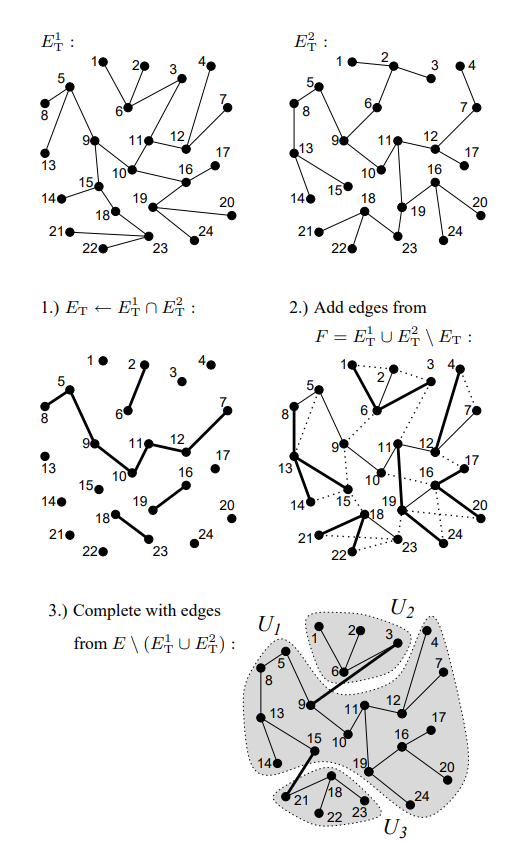
The GA itself is a µ+λ GA with one crossover and one mutation performed per generation. A population size of 255 chromosomes are mapped onto a 2D grid (15 x 15). A local mating selection [13] is used to help prevent early convergence meaning only chromosomes near to each other in the grid can mate. Selection 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.

### Edge-Set Representation

Another approach by Günther Raidl [14] uses an edge-set representation along with specialised genetic operators to achieve an efficient GA solution. The representation being all edges forming the degree-constrained spanning tree (d-ST) implemented for the sake of efficiency in a hash-table storing each pair of connected vertices as a tuple. This simple representation along with specialised initialisation, crossover and mutation operators fulfil all of the criteria for an efficient GA DCMST solution as laid out in his paper [14].

**Initialisation:** The initial population must include only valid d-STs. To ensure this, a procedure derived from Kruskal’s algorithm [15] is used to create d-STs. For efficiency Raidl recommends the use of a union-find data structure [16] when checking if two vertices have some connection while building the d-ST. Given feasible parents new generations of solutions are obtained by the following means.

**Edge Crossover:** The idea behind the edge crossover operator is to produce a d-ST that inherits as many edges as possible from its parents. The operator first ensures that whatever edges are in both parents are included in the child. Then, the edges in one or the other (but not both) are checked for inclusion one after another being added to the d-ST if they do not introduce a degree-constraint or cycle. If a complete d-ST (|V|-1) is constructed then the crossover terminates and it is returned. If not, the partially built d-ST child is partitioned into disjointed sets containing vertices only connected to each other. A single edge is then added to connect each disjoined set to the final d-ST. The edge is added between two vertices, one on each sub tree, that have both not yet met the degree constraint. The computational efficiency of the edge crossover is O(|V|).



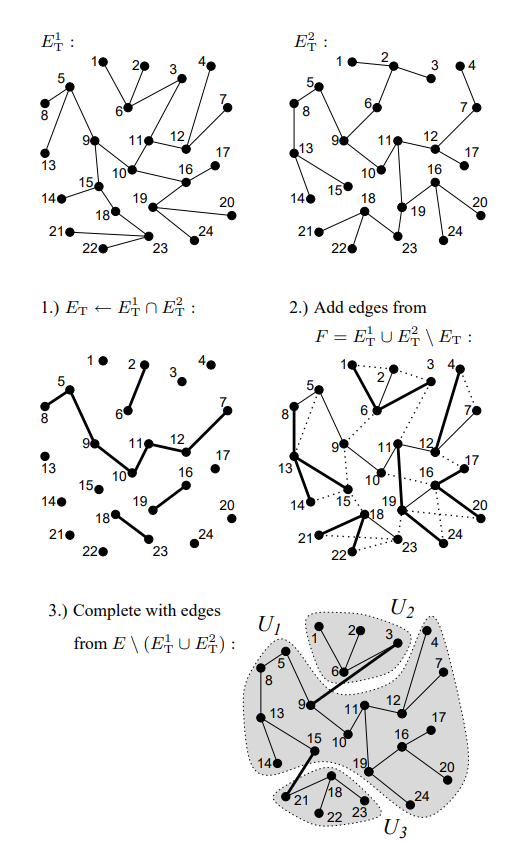


Figure - Visualisation of Edge Crossover (d=3)

**Edge Insertion Mutation**: The mutation operator inserts a random new edge between two vertices and deletes another edge lying on the cycle caused by the insertion. To prevent degree constraint violation the choice of one of the vertices is restricted so that its degree is less than . The set of edges lying on the path of cycle are obtained through a depth-first search performed on a temporary adjacency list representation. One of these edges is deleted after the edge insertion. The edge insertion mutation operator has a complexity of O(|V|).

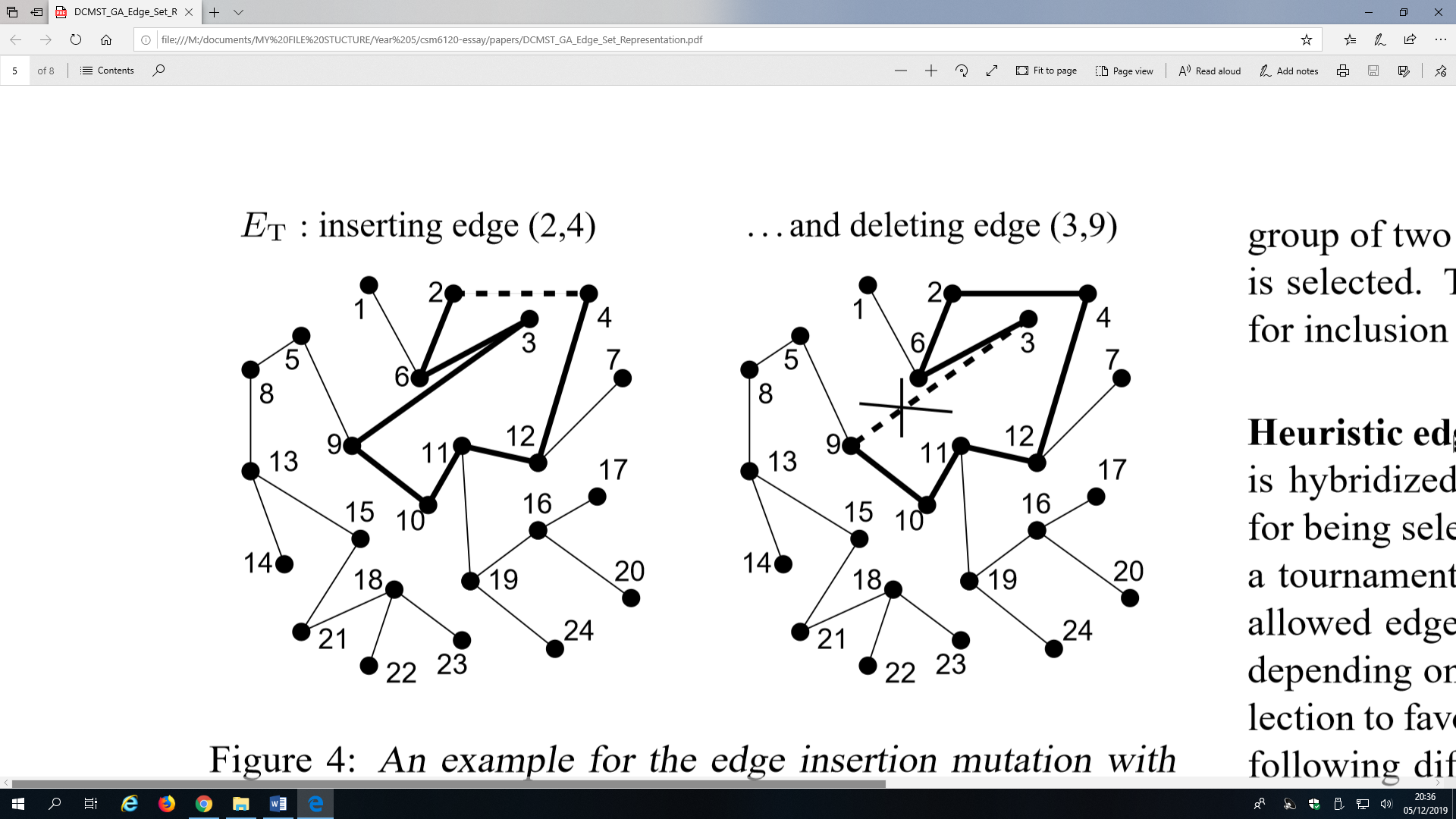


Figure - Visualisation of Insertion Mutation

*A* ***heuristic assumption*** *is incorporated into the procedures that says edges with smaller costs should be preferred over more expensive edges. The* ***edge-cost heuristic*** *is incorporated in the following ways.*

**Heuristic Initialisation:** Instead of trying to add edges in a random order the edges are sorted by increasing cost at the beginning of a run. The first candidate solution of the initial population is created by the Kruskal-like initialisation procedure using this edge ordering. To ensure diversity all proceeding solutions are created with less heuristic bias (denoted by α) by randomly permuting the cheapest k edges to be out of the initial sorted order. Number of shuffled edges increases with time throughout the initialisation phase ().

**Heuristic Edge Crossover:** During crossover when choosing which edges to add in to the growing child tree from edges with lower cost have higher probability to be added in first. Edges are selected via binary tournament selection with the cheaper of the two being chosen.

**Heuristic Edge Insertion Mutation:** To favour low cost the -th cheapest edge is chosen based of a random number () limited by the value of a strategy parameter where a smaller value for results in a stronger bias towards cheaper edges. Tournament selection was not deemed appropriate for mutation because the set of all edges in the whole graph could be huge.

The GA is another steady state algorithm (µ+λ) with a population size of 100. Binary tournament selection is used with the crossover and mutation probability both set to 0.8. New children replace the worst solution in the population with the exception that identical solutions to any already in the population are discarded to preserve diversity.

## Other Noteworthy Algorithms

The search algorithms below are well mentioned in literature as solutions to the DCMST. However, both were compared in the results of Knowles’ and Corne’s paper [12] with the RPM applied and were found to not perform as well as the GA. A description of the two is included for completeness along with the parameters set for their use in the RPM experiments.

### 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.

**Parameters for RPM Experiments:** Evaluations allowed was 10,000. Value of was set at 500.

### Simulated Annealing

Simulated Annealing (SA) [18] is a search method that can escape local optima by accepting transitions to worse solutions based on a control parameter . The value of is decreased over time allowing fewer such transitions to occur. The rate at which is decreased is known as the cooling schedule. The algorithm iterates until some stopping criteria is met.

**Parameters for RPM Experiments:** Evaluations allowed was 10,000. The value of was calculated such that initially () 90% of transitions are allowed and such that finally () 0.1% of transitions are allowed.

# Comparing the Different Approaches

Knowles and Corne show [12] that A GA using the Randomised Primal Method is better than Multi-Start Hill Climbing, Simulated Annealing and a GA using Prüfer encoding. An Edge-Set Representation GA is then shown to be better than an RPM GA by a direct comparison in the results section of Raidl’s paper [15]. My goal for this section is to first summarise the results of both papers and then add my own comparison between the Edge-Set Representation GA with the less associated Ant-Based optimisation algorithm.

## GA Comparisons Summary

Knowles and Corne present their results as a ratio between the best d-MST weight found and the known MST weight of the graph as calculated by Primm’s algorithm. They first use a single 250 node M-graph[[1]](#footnote-1) with a maximum node degree of 10 to compare methods. The three search algorithms (MHC, SA, GA) using RPM were run 20 times each on the graph with each run stopping after 10,000 evaluations. The best result for each algorithm is plotted on the below graph. The d-Primm’s and BF2 algorithms are deterministic; the result of their single run is plotted for comparison.

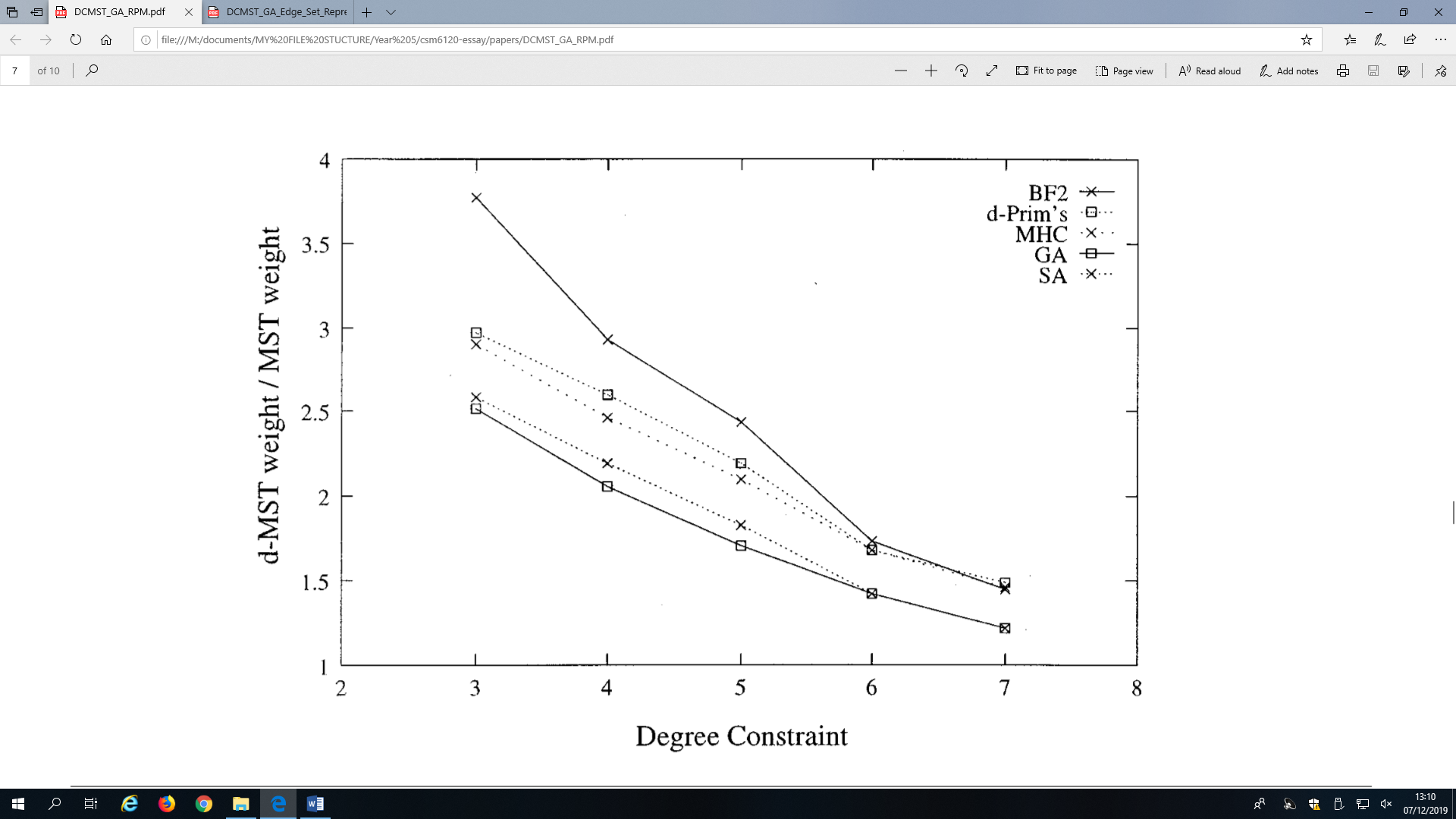


Figure - Comparison of MHC, SA and GA (all using RPM) on a 250-node M-graph.

Nine more M-graphs at each of 3 sizes (50, 100 and 200 nodes) are used for comparison. Each RPM method being run again 20 times for 10,000 evaluations each with the degree-constraint set at 5. It is clear from these results that the GA employing RPM finds lower cost solutions than any of the other algorithms.

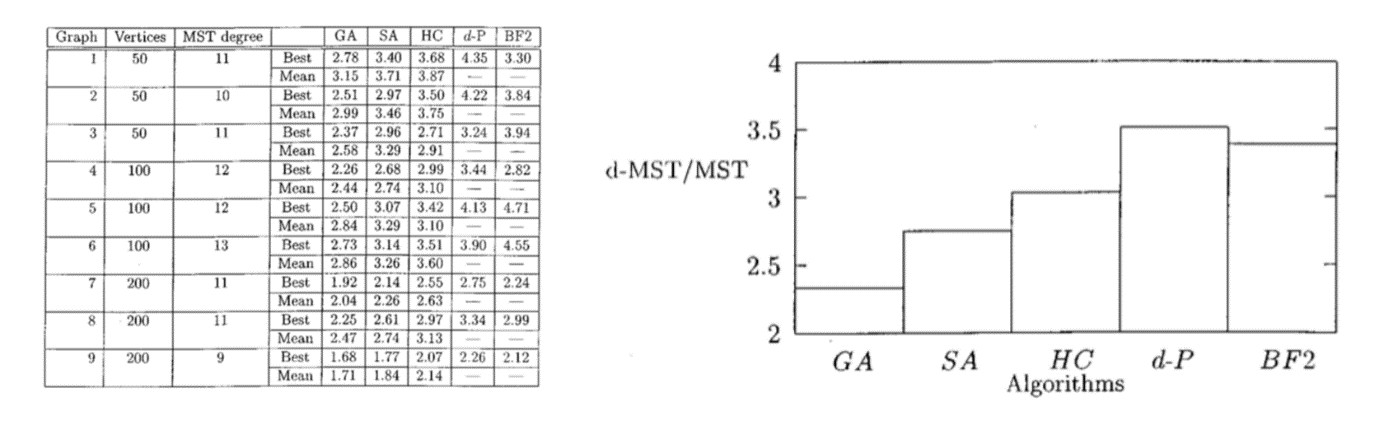


Figure - Results of runs on 9 M-Graphs. Mean of best solutions visualised in bar chart.

Raidl presents the results of each AI search algorithm as the % gains over the deterministic d-Primm’s algorithm. Therefore, using d-Primm as a reference the quality of (fitness) improvement is calculated for each approach and displayed in the results.

The first table of results compare previous search algorithms on SHRD (structured hard) dataset examples. SHRD is a set of particularly demanding problem instances defined by Krishnamoorthy *et al.* [19]*.* 20 runs were done for each problem instance for each of the degree-constraints with 10,000 evaluations in each run. Note that the first 5 algorithms results are taken for comparison from Krishnamoorthy’s paper on comparison of DCMST solutions [19]. This table shows Prüfer Encoding (F-EA, P-EA), Simulated Annealing (SA) alongside Edge-Set representation (S-EA) so I believe it is worth including as further confirmation of the inferiority of these former algorithms.

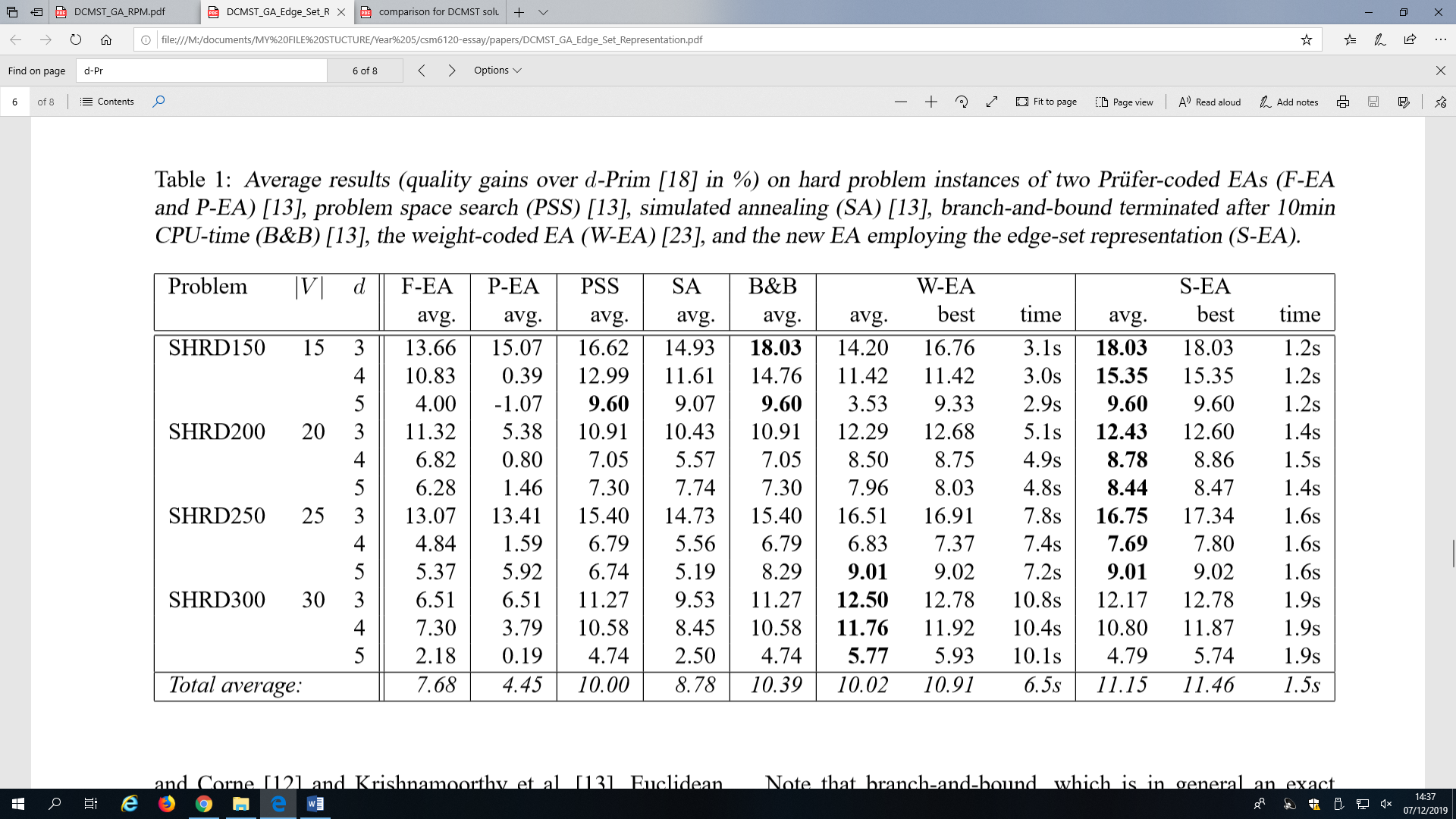


Figure - Average % quality gains over d-Primm on hard problem instances.

The second table of results directly compares Edge-Set representation (S-EA) to RPM (K-EA) through experiments on the same nine misleading problem instances (M-graphs) as used by Knowles and Corne. In addition, three larger M-graphs (up to 500 vertices) are generated to expand the results.

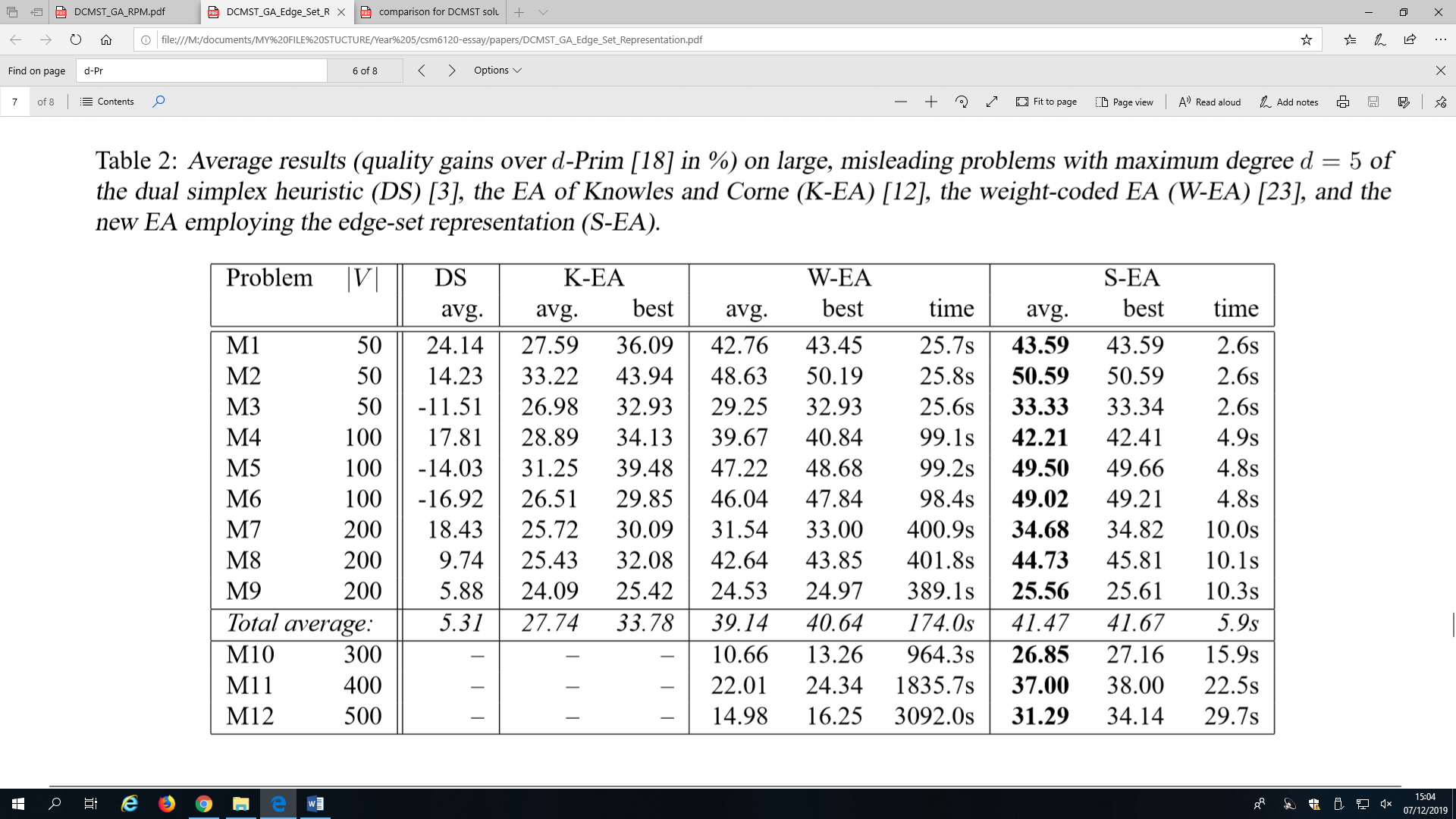


Figure - Average % quality gains over d-Primm on misleading problem instances.

From the above table it is clear that Edge-Set Representation (S-EA) has higher quality gains over d-Primm’s then the Randomised Primal Method (K-EA) in the best and average case on every problem instance. This is not unexpected because RPM uses d-Primm to decode a chromosome into a new solution. This is computationally expensive because the complexity of d-Primm is O(|V|2log|v|). In edge-set representation the process of creating and evaluating a new candidate solution is always achieved in O(|V|) time.

Raidl compares algoritms using gaisn over d-primms where as Knowles amnd corne compare alogorutms using how close the algorithms’ best solutions are to the actual graph MST

## Ant-Based vs Edge-Set Representation

* Summarise the comparisons done by the genetic algorithms people (chain of solution encoding improvement)
* Add my own comparison between ant colony and the best GA
* NOTE might need to read different techniques for generating random graphs in RPM because its used in edge-set aswell I think to generate test insatcnes

# 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] | A. Volgenant, “A Langrangean Approach to the Degree Constrained Minimum Spanning Tree Problem.,” *European Journal of Operational Research,* no. 39, pp. 325-331, 1989. |
| [5] | 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. |
| [6] | 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. |
| [7] | T. H. Cormen, C. E. Leiserson, Rivest, R. L and C. Stein, Introduction to Algorithms, 2nd ed., Cambridge, MA: MIT Press, 2001. |
| [8] | 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. |
| [9] | H.-G. Beyer and H.-P. Schwefel, “Evolution strategiesc A comprehensive introduction.,” *Natural Computing,* vol. 1, no. 1, pp. 3-52, 2002. |
| [10] | J. W. Moon, “Counting Labelled Trees,” in *Associating sequences with trees*, London, William Clowes and Sons, Limited,, 1970, pp. 4-8. |
| [11] | J. Knowles and C. D, “A New Evolutionary Approach to the Degree-Constrained Minimum Spanning Tree Problem,” *IEEE Transactions On Evolutionary Computation,* vol. 4, no. 2, pp. 125-134, 2000. |
| [12] | R. C. Prim, “Shortest Connection Networks And Some Generalizations,” *Bell System Technical Journal,* pp. 1389-1401, 1957. |
| [13] | R. J. Collins and D. R. Jefferson, “Selection in Massively Parallel Genetic Algorithms,” in *4th Int. Conf. Genetic Algorithms*, San Mateo, CA, 1991. |
| [14] | G. R. Raidl, “An Efﬁcient Evolutionary Algorithm for the Degree-Constrained Minimum Spanning Tree Problem,” in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00*, Vancouver, 2000. |
| [15] | J. B. Kruskal, “On the shortest spanning subtree of a graph and the traveling salesman problem.,” *Proceedings of the American Mathematics Society,* vol. 7, no. 1, pp. 48-50, 1956. |
| [16] | M. Krishnamoorthy, A. T. Ernst and Y. M. Sharaiha, “Comparison of Algorithms for the Degree Constrained,” *Journal of Heuristics,* pp. 587-611, 2001. |

1. An “M-Graph” is a misleading graph aimed at deceiving greedier algorithms, they are generated using a modification to a procedure originally defined by Boldon *et al.* [20]for generating biased random complete graphs. The procedure for M-graph generation is defined in section IV of Knowles’ and Corne’s paper. [↑](#footnote-ref-1)