Evolving Hard Instances for Graph Colouring Problems

Simon Bowly

February 28, 2013

The following report details the application of a genetic algorithm to the generation of data to aid in algorithm selection for graph colouring. The genetic algorithm is used to evolve instances of the graph colouring problem which are hard for certain heuristic algorithms to solve. This set of instances is then used to better define a space in which graphs can be grouped according to which heuristic will provide the best solution. The results demonstrate that this method can be used to generate a set of graphs which challenge the capabilities of available heuristic algorithms and cover a large range of the space defined by graph features.

Contents

Postal Address: 111 Barry Street
c/- The University of Melbourne
Victoria 3010 Australia

Email: enquiries@amsi.org.au
Phone: +61 3 8344 1777
Fax: +61 3 9349 4106
Web: www.amsi.org.au

1 Introduction

1.1 Graph Colouring

Graph colouring is the process of assigning a colour to each node in a graph such that any two nodes connected by an edge do not have the same colour. An *optimal colouring* is achieved when the number of colours required to complete a colouring without conflicts is minimised. The number of colours required to do this for a given graph is called the chromatic number.
The graph colouring problem is classified as NP-complete, indicating that there is no known, fast way to solve the problem optimally for a given graph. Therefore a number of heuristic algorithms are used to achieve the best solution possible in reasonable time. Given the classification as NP-complete, the only way to confirm optimality of such a solution is to check all possible solutions to the problem, which is an unreasonably slow method. As such it is important to select a heuristic algorithm which will give an optimal solution, or at least the best possible solution from a portfolio of heuristics.

1.2 Motivation

The *No Free Lunch Theorem* [1] states that all heuristic optimisation algorithms have an equal mean performance over the set of all possible optimisation problems. Therefore for any subset of graph colouring problems for which an algorithm performs well, there must be a corresponding subset for which the algorithm performs poorly. The aim of this project is to produce data which will help define a space in which the set of all graph colouring problems can be separated into subsets on which the available heuristics perform well. The separation of this space is termed the *algorithm selection hypothesis*, which would enable a computer to decide quickly which algorithm to use for a given graph problem. The graphs are to be characterised by a set of features which can be easily visualised using a scatterplot. The *performance footprint* of a given heuristic is defined as a region within some space of graph problems (defined by features) where that heuristic outperforms other available methods.

1.2.1 Heuristic Algorithms

The results generated for this report are from a comparison of two heuristic algorithms for graph colouring: *DSATUR* (Degree Saturation) and *MAXIS* (Maximal Indepen-

dent Set). [1]
The DSATUR algorithm uses dynamic ordering of vertices to colour them one by one. An initial ordering is chosen based on the degree of each vertex, and a colour is assigned to each from the existing colour set as long as this does not create a conflict. If none of the available colours can be used, a new colour class is created. The vertex order is updated as the algorithm progresses according to the degree saturation of the vertices (number of connected colours).
The MAXIS algorithm searches for the largest set of vertices within the graph which share no connections (are independent). The first colour is assigned to this group, and the algorithm progresses the find a new maximal independent set from the remaining vertices, continuing until all vertices are coloured.
A third algorithm, TABU, uses a local improvement search from a known valid colouring in order to converge on an optimal result. A list of unsuccessful improvement attempts (TABU list) is kept in order to prevent backward steps. This algorithm is not tested here.

1.2.2 Graph Features

Graph features which meet the requirements for this project should be fast to calculate, and provide useful information to aid in algorithm selection. The following graph properties are calculated and used to define positions of graphs in a feature space:

Density defines the number of edges present in a graph as a fraction of the maximum number of edges which could be present

Node Degree number of nodes with which a given node shares an edge

Average Path Length average length of all the shortest paths existing between nodes

Clustering Coefficient measures the degree to which nodes are clustered together

Eigenvector Centrality measures the importance of nodes in a network, calculated from the number of connections from a node to other important nodes

Spectrum set of eigenvectors of the adjacency matrix of a graph

Algebraic Connectivity eigenvalues of the Laplacian matrix

[1] C++ implementations of these colouring programs were provided open source by Joseph Culberson, University of Alberta [2]

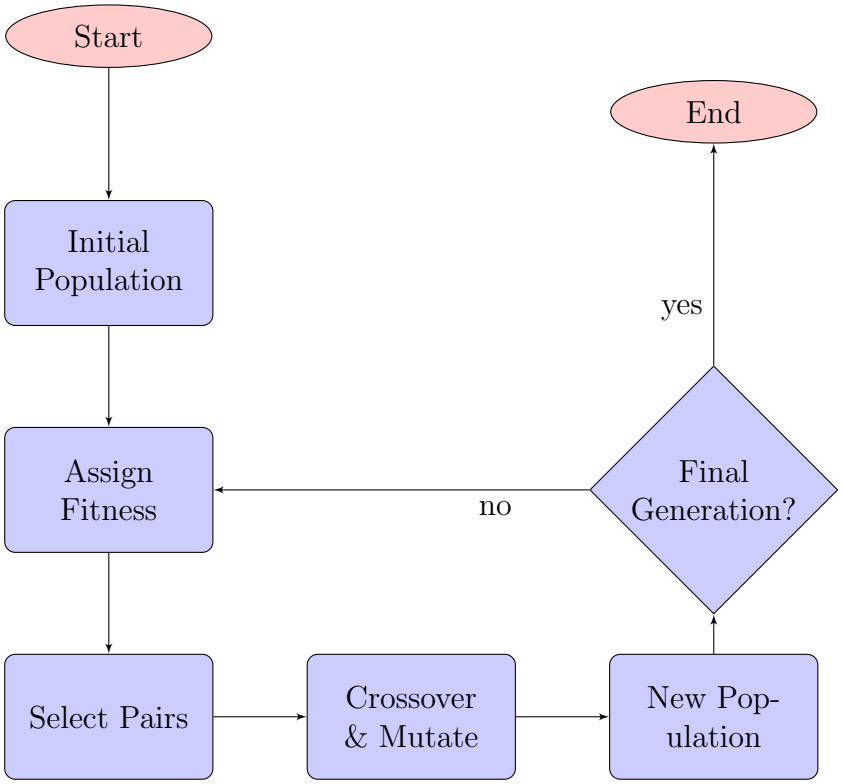


Figure 1: Genetic Algorithm operation

1.3 Evolving Hard Instances

It has been found that randomly generating sets of graphs does not provide enough useful data to determine an effective separation of graph problems according to algorithm performance [3]. Therefore it is necessary to find a method for generating instances of the graph colouring problem which are difficult for a given algorithm. Furthermore it is desired to design a method for finding graphs in particular locations within a feature space, so that regions which are not explored by random graph generation can be used to further test the algorithms.

A *Genetic Algorithm* is used to achieve these goals. This is a population based search of the set of possible instances which aims to improve their fitness by selecting for and combining the best instances in the population. A fitness value is assigned to each instance in the population and selection is made with a higher probability of selection placed on the fitter instances. Selected instances undergo crossover and mutation (defined later) to produce a new population.

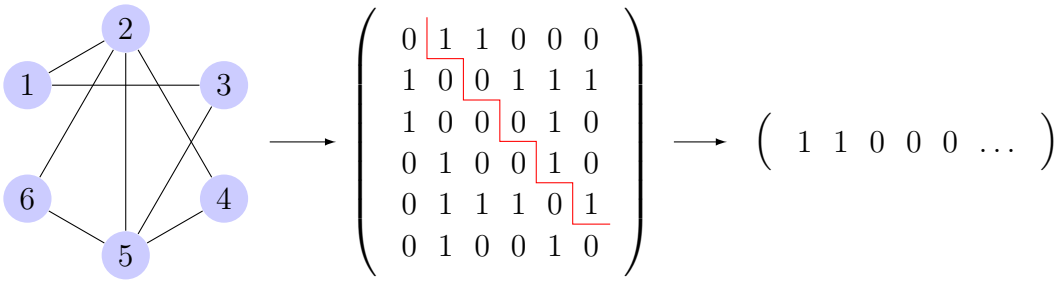


Figure 2: Encoding of a graph for storage in the GA population

2 Genetic Algorithm Design

The design of a Genetic Algorithm (GA) requires definition of methods for encoding solutions, evaluating their fitness values, selecting fitter instances and performing crossover and mutation on instances. In the case of this GA, a solution is some graph G(V,E), which defines a set of vertices and the edges connecting them. The fitness function is designed to favour instances which are harder for one of the algorithms described in Section 1.2.1 and have target properties as listed in Section 1.2.2. The algorithm is coded in MATLAB.

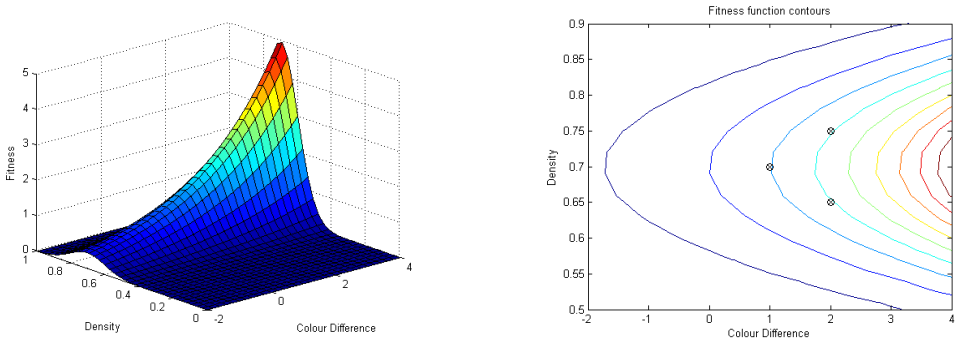
2.1 Encoding

The most common method for describing a graph mathematically is to use an adjacency matrix. For an n-vertex graph, an $n \times n$ matrix is used, where a one in column i, row j indicates that an edge connects node i to node j. All other elements are zero, including the main diagonal. Graph colouring is only applicable to undirected graphs, and therefore the adjacency matrices used here are symmetric. For storage simplicity, the upper triangular part of the adjacency matrix is unrolled into a row vector. This allows a population of N graphs with E total possible edges to be stored in an $N \times E$ matrix. Figure 2 shows the process of encoding a graph in the GA.

2.2 Fitness Function and Selection

The fitness function (or objective function in optimisation theory) has two goals for this application:

1. Generate hard graphs



(a) Surface (b) Contours

Figure 3: Representation of the fitness function

2. Target required features

Difficulty of a graph colouring problem is measured by a pairwise comparison of two algorithms from the portfolio: algorithm 1, for which the instance is to be hard, and algorithm 2, to be used as a comparison. A graph is defined as hard to colour for algorithm 1 if it requires more colours to complete than algorithm 2. There is no measurement of absolute difficulty since the optimal colouring is not known. The difference in number of colours required by the two algorithms is taken as the parameter in this case, and placed on an exponential scale so that a higher positive value (hard for algorithm 1) produces a larger value of the difficulty score.
Feature targeting is performed by multiplying the difficulty score by a standard normal distribution with its mean at the required feature value and a standard deviation chosen for an appropriate level of feature selectivity. The composite function is shown as a surface plot in Figure 3a. Higher fitness values have a higher chance of being selected, so it is clear that the GA will optimise towards graphs of greater difficulty for algorithm 1 with the desired target property. Thus the equation used to calculate this composite fitness function is given by:

$$\begin{aligned} N(F,\mu,\sigma)\cdot\exp(\alpha\cdot(\#clrs_{alg1}-\#clrs_{alg2})) \\ F = graph\ feature\ value \\ \mu = feature\ target\ value \end{aligned} \tag{1}$$

The tradeoff between difficulty and required properties is indicated by the contour plot in Figure 3b. The three points on the same contour have indentical values of the fitness function for different parameters. With the selected scaling parameters σ and

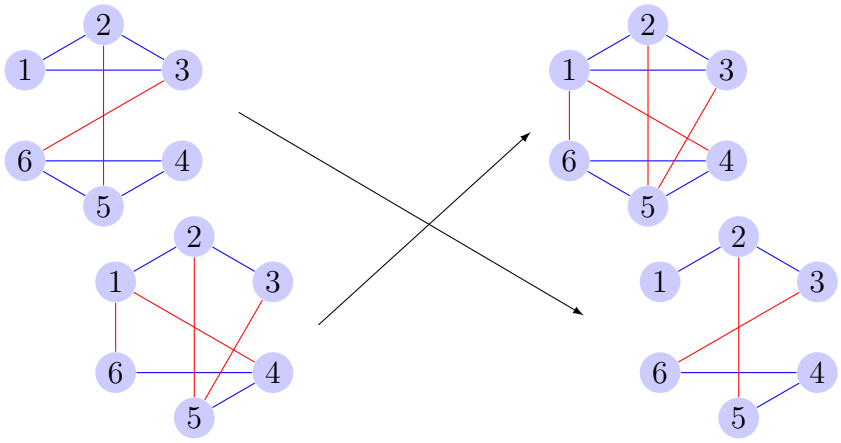


Figure 4: Crossover of two parent graphs to produce two children

α the GA may allow the graph density to vary as much as 0.15 from the target in either direction, if it results in an increase in colour difference by 1, and still obtain an improvement in the fitness function value. Hence the selection of scaling parameters is important in determining how much the algorithm will focus on targeting graph properties against obtaining hard instances. The GA selects instances to be carried on to the next generation using a weighted random selection known as Roulette Wheel Selection. The fitness function value of each instance is used to weight each graph's selection chances. Instances are selected in pairs to undergo crossover and mutation before being passed on to the next generation.

2.3 Crossover and Mutation

Crossover of two selected graphs is performed by combining their adjacency matrices by quadrants. Both matrices are split into four quadrants at a random point along the main diagonal. One child is produced using the diagonal quadrants matrix 1, combined with off-diagonal quadrants of matrix 2. The other child is produced using the opposing quadrants. This method splits the vertices of a graph into groups, keeping the connections between vertices within the groups (the diagonal quadrants) and bringing in cross group connections from the other graph (off-diagonal quadrants). Figure 4 shows the effect of this crossover. This method was selected because it has a physical meaning for the graphs and preserves the interconnectivity of groups of nodes, which has a strong effect on colouring.

Following crossover, mutation is performed on the child instances in order to main-

tain diversity, which helps the GA avoid getting stuck in local optima. This mutation is performed by altering a random number of the possible edges in the graph, removing some that exist and adding some that were not previously present.

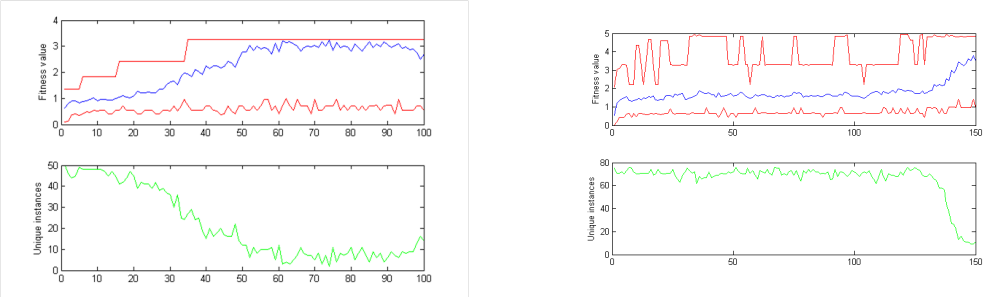
2.4 Parameter Tuning

Some experimentation with the parameters of the GA was required to ensure effective operation. The main decisions to be made were the mutation rate and the use of *elitism*. Elitism is performed before the random selection process, and passes the two fittest instances on to the next generation unchanged. This prevents the crossover and mutation processes from destroying a potential good solution before the GA has a change to improve it further. The convergence characteristics of the GA search are shown in Figure 5. These plots show the minimum, maximum and mean values of the fitness function, and the diversity (number of unique instances present) in the population as the generations progress.

Figure 5a shows convergence of the GA when elitism is used. The algorithm shows a steady improvement in fitness values of the population (consistent with finding harder instances) and a steady decline in diversity until it reaches a point where it can make no further improvements. This was found to be a very effective method for generating hard instances, however it does produce clustered results. The instances selected by the elitism process eventually take over the population with many identical or almost identical copies. As a result this process is not very effective at finding a wide ranging data set.

Figure 5b shows convergence of the GA without elitism. The diversity in the population remains at a maximum for a much larger number of generations. From the fitness function plot it is clear that improvements are being made to instances within the population but the algorithm only pursues improvements to these graphs for a short period, after which the fittest instances are mutated and become less fit. This method does not find as hard instances as elitism (colour difference is not as high) but it is effective at returning results with a wide range of features. As a result this is the method used.

Experimentation with mutation parameters indicated that quite a high mutation rate (up to 10%) was required to ensure the maintenance of diversity within the population. This was coupled with a cap of 2% of all edges in a single graph which could be altered in a mutation process. Higher values did not preserve any of the characteristics of instances which were mutated and resulted in many potentially good solutions losing fitness and being bred out too early.



(a) With Elitism (b) Without Elitism

Figure 5: Monitoring convergence of the Genetic Algorithm

It was found that the exponential scaling factor α worked best at a value of 0.4, giving a 50% increase to the selection chances of an instance with a 1 higher colour difference. When elitism was used, this value was decreased to 0.3, to compensate for the fact that the best instances were already selected by default.

3 Research Approach

The generation of a data set which would aid in separating hard instances within a feature space required the following stages:

1. Generate graphs with a wide range of features
   - Run GA with a range of target densities, with high standard deviation σ on the property bell curve
   - Complete several runs at each density, using different starting populations
   - Use non-elitist selection, so the algorithm finds many local optima
   - Save the best results, based on fitness, with a condition on uniqueness of instances
2. Visually test for combinations of features which show good separation
   - From the results found, assign graphs as hard for algorithm 1, hard for algorithm 2, or equally difficult
   - Test features in pairs on a simple X-Y axis, plotting each graph as a point, to find a 2D space where the sets can be separated

- Apply logistic regression to find a decision boundary for algorithm selection hypothesis

3. Target positions in this new feature space to verify hypothesis
   - Select a point in the chosen 2D feature space
   - Use the GA to target both features (penalise graphs not near the chosen point)
   - At a point where hard graphs for algorithm 1 are expected, optimise towards hard graphs for algorithm 2 (and vice versa)
   - Generate random graphs at a chosen point in the space (not selecting for difficulty, just features) and examine whether they are hard for either algorithm

4 Results

The method detailed in Section 3 was applied to generate an algorithm selection hypothesis for the DSATUR and MAXIS algorithms. Graphs used have 50 vertices, and densities in the range 0.35 to 0.75.

4.1 Initial Results Generation

Figure 6 shows the initial results generated for this pair of algorithms. Each graph is indicated as a point in a 2D space defined using density and eigenvector centrality mean (see Section 1.2.2 for explanation) as coordinates, as these parameters gave the best visual separation of instances. Graphs were defined as hard for an algorithm if they required at least 2 more colours than the competing algorithm. Four target densities were chosen for the initial runs, from 0.4 through to 0.7, generating a wide range of feature combinations. The resulting values of eigenvector centrality mean were in the range 0.138 to 0.141. A uniqueness condition was applied that no two graphs with less than 10 edges different would be shown on the plot. In the case of a conflict the graph with a higher fitness value was used. Parameters for the GA are a 10% mutation rate, 75% crossover rate and exponential scaling value α of 0.4.

Given that the results obtained by a GA are have some measure of random influence (the algorithm starts from a randomised initial population and undergoes mutation), it is reasonable to conclude that the data produced reflects which graphs are more common. Since the data shows a reasonable separation of the two classes of graphs in

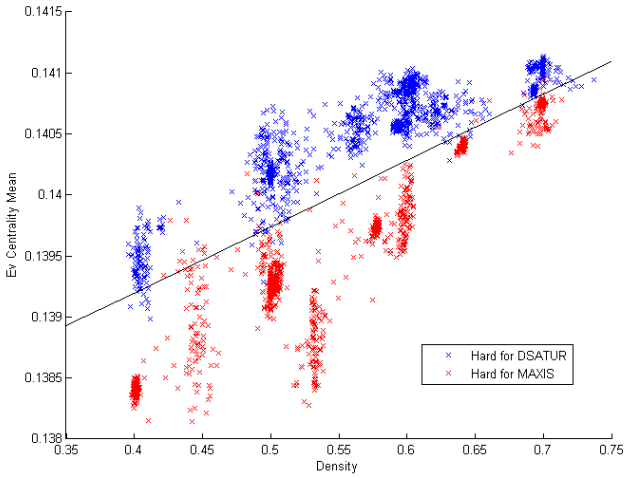
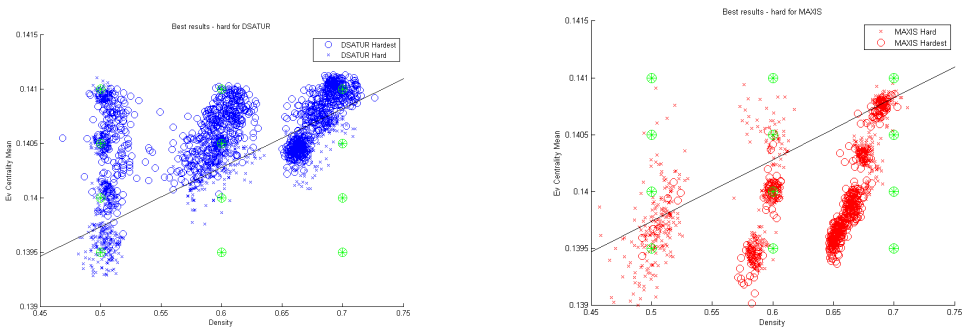


Figure 6: Results for comparison of DSATUR and MAXIS with 50 vertices

the 2D space, it is assumed then that a selection hypothesis generated from this data would give the correct result in most cases. The decision boundary found using linear regression is indicated on the plot and gives a selection prediction with less than 10% error on the same data set.
Graphs in this set seem to exist in only a narrow band of feature values. Since only density was used as a target criteria in this case (eigenvalue centrality was not controlled), it is reasonable to conclude that there are some combinations of density and eigenvector centrality mean which cannot be obtained.

4.2 Testing the Selection Hypothesis

Figure 7 shows the result of using the GA to target a grid of 12 points (shown in green) in the 2D space, finding hard instances for both DSATUR and MAXIS. The decision boundary determined from the original results is shown on both plots. For these figures, graphs are separated into *hard* (2 colour difference) and *hardest* (more than 2 colour difference) instances for each algorithm.
The results indicate that the selection method is not 100% reliable in the space defined by density and eigenvector centrality mean. For each algorithm, the GA was able to find hard instances in regions where the selection hypothesis indicated good performance. However, the data does indicate the following:



(a) Hard for DSATUR (b) Hard for MAXIS

Figure 7: Targeted search results

- Relatively fewer graphs were found which contradicted the hypothesis than those which supported it
- The GA found less hard instances as it moved further from the decision boundary towards the region where they were not expected
- Grouping of *hardest* instances tended to be on the correct side of the decision boundary
- The extreme target points only produced hard instances when the algorithm drifted a significant distance from its target (the gain from difficulty in the fitness function was enough to offset penalties from distance to the target)

Again, based on the interpretation of GA results as indicative of the most common instances, it can be seen that the decision boundary does provide some useful information to contribute to a selection hypothesis. It is assumed that graphs near the boundary will then require another feature to separate them more concretely into hard graphs for either heuristic. It is also possible that both heuristics are performing poorly on these borderline instances, and a third heuristic will be more effective in this region of the space.

Figure 8 shows results from runs of the GA using the same target points, with no weighting on graph difficulty in the fitness function. Therefore the GA is essentially generating random instances with desired graph features. A set of completely randomised graphs (for example an initial population) is indicated, and appears to be confined to a narrow band. However the GA is capable of evolving instances away

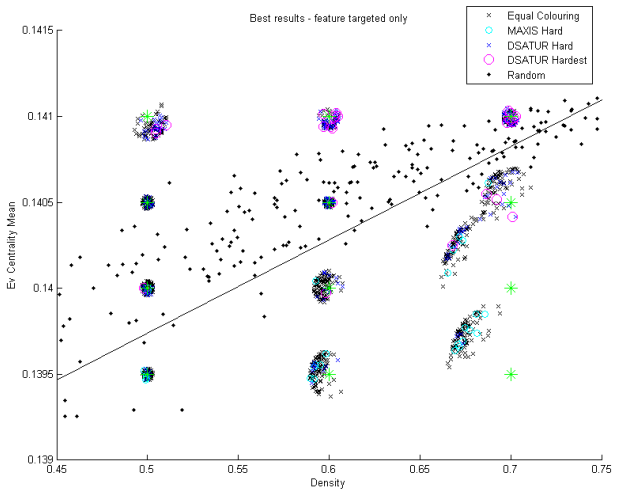


Figure 8: Targeted search results with no weighting on difficulty

from the band towards the more extreme target points.
The results indicate that at points further from the centre of this band (where the decision boundary exists) are less common. Furthermore, random graphs found in these regions further from the decision boundary support the selection hypothesis, e.g. some graphs in the bottom right region are found to be hard for MAXIS, despite the fitness selections not being based on difficulty. This observation indicates that the selection hypothesis is likely to be effective for many randomly generated and real world graphs, since these are not generated with a difficulty requirement.

4.3 Further Results

Figure 9 shows graphs produced when comparing the same two algorithms, DSATUR and MAXIS, with 100 vertices. The same features are used for comparison, and there is still evidence of separation in this 2D space. However, this separation does not occur at the same values of the features. Therefore it would be necessary to perform some feature scaling in order to apply the decision boundary obtained to a wider range of graph results.

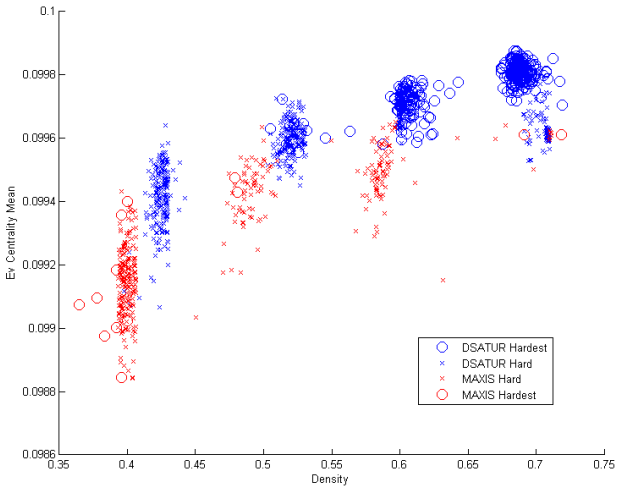


Figure 9: Results for comparison of DSATUR and MAXIS with 100 vertices

5 Conclusion

The results serve as a proof of concept that a Genetic Algorithm with an appropriate fitness function can be used to generate an effective data set for use in determining an algorithm selection hypothesis. The composite fitness function, combining a difficulty measure with feature targeting, is effective at finding graphs with specific properties and testing whether such graphs can be hard for some heuristic algorithm. It can also demonstrate whether a particular type of graph is either rare or does not exist.

The decision boundary calculated using logistic regression from the hard instance data provides a useful algorithm selection hypothesis. Distance from the dividing line gives a measurement of confidence that the selection will be the correct one. The result indicates that when attempting to colour a graph, finding its position in the space defined by density and eigenvector centrality mean is an effective indicator of which algorithm should be used. For example, for a graph found to exist below the line, hard instance data shows that MAXIS will perform poorly. Therefore the DSATUR heuristic should be selected to colour such a graph.

Results from the test of the DSATUR and MAXIS heuristics indicate that a strict algorithm selection hypothesis cannot be generated on the basis of only two features. However, density and eigenvector centrality mean do provide some useful information to make a choice of algorithm, since a linear decision boundary can correctly split the

graphs based on these features in more than 90% of cases.

5.1 Further Work

The following tests are recommended to extend the applicability of these results and generate a larger and more useful data set:

- Produce data for graphs with different numbers of vertices, to determine a feature scaling method which would make results applicable to all graphs
- Use different machine learning methods and more graph features to produce a more accurate selection hypothesis
- Explore whether real world problems obey the same measured rules as randomly generated instances
- Compare different pairs of algorithms from the portfolio

References

[1] Ben Harwood, *Evolving Instances for the Evaluation of Graph Colouring Algorithms*. Monash University, 2012.

[2] Joseph Culberson, Graph Colouring Programs. University of Alberta. *http://webdocs.cs.ualberta.ca/ joe/Coloring/index.html*

[3] Brendan Wreford, *The Impact of Graph Features on the Performance of Graph Colouring Heuristics*. Monash University, 2010.

Acknowledgements

Monash University Supervisors
Professor Kate Smith Miles
Dr Davaatseren Baatar

Australian Mathematical Sciences Institute