Graph Coloring

Doncean Șerban-Gabriel, Oleniuc Iulian, Panaite Doru-Răzvan

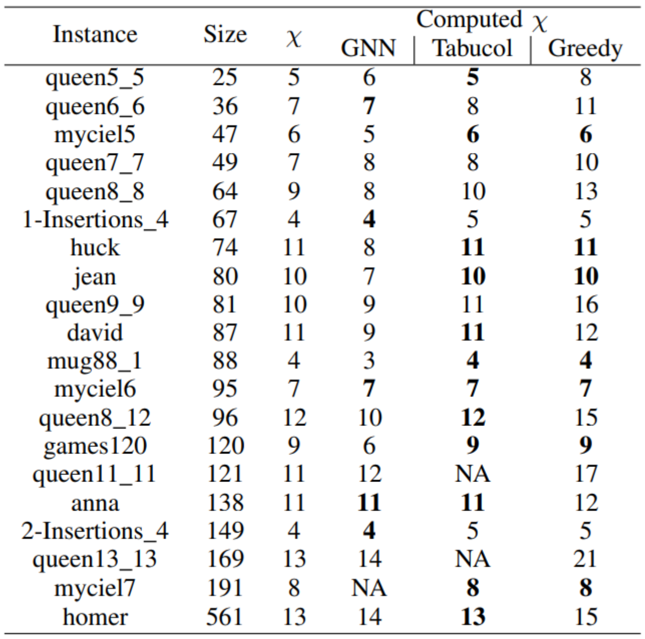
14 March 2024

1 SOTA

"Graph Colouring Meets Deep Learning: Effective Graph Neural Network Models for Combinatorial Problems" [1] This paper explores the intersection of graph coloring problems and deep learning techniques, focusing on the effectiveness of Graph Neural Network (GNN) models.

The model initially assigns the same randomly initialized embedding in R^d to all vertices in the graph. This embedding becomes a learned parameter as the model trains. To facilitate communication between neighboring vertices and between vertices and colors, the model requires both a vertex-to-vertex adjacency matrix and a vertex-to-color adjacency matrix. After initialization, adjacent vertices and colors communicate and update their embeddings over several iterations. The resulting vertex embeddings are then fed into a Multi-Layer Perceptron (MLP) to compute a logit probability, indicating whether the graph accepts a C-coloration (corresponding to the model's prediction of the answer to the decision problem: "does the graph G accept a $C-coloration?$"). The Graph Neural Network (GNN) model learns message functions, implemented as MLPs, to translate color embeddings into messages understandable by a vertex update function ($C : R^d \to R^d$), and to translate vertex embeddings into messages ($V : R^d \to R^d$). In addition, it learns functions (RNNs) for updating vertices ($V_u : R^{2d} \to R^d$) and colours ($C_u : R^{2d} \to R^d$) given their hidden states and received messages. The model uses 64-dimensional embeddings for both vertices and colours. The message computing MLPs consist of three layers (64, 64, 64) with ReLU nonlinearities, except for the linear activation in the output layer. The RNN cells used are basic LSTM cells with layer normalisation and ReLU activation. Training involves Stochastic Gradient Descent using TensorFlow's Adam optimizer for the message computing MLPs and RNNs.

The loss function is the binary cross-entropy between the model's final prediction and the ground truth, which is a Boolean indicating whether the graph accepts the target colourability in a given Graph Coloring Problem (GCP) instance.



Instance	Size	χ	Computed χ		
			GNN	Tabucol	Greedy
queen5_5	25	5	6	**5**	8
queen6_6	36	7	**7**	8	11
myciel5	47	6	5	**6**	**6**
queen7_7	49	7	8	8	10
queen8_8	64	9	8	10	13
1-Insertions_4	67	4	**4**	5	5
huck	74	11	8	**11**	**11**
jean	80	10	7	**10**	**10**
queen9_9	81	10	9	11	16
david	87	11	9	**11**	12
mug88_1	88	4	3	**4**	**4**
myciel6	95	7	**7**	**7**	**7**
queen8_12	96	12	10	**12**	15
games120	120	9	6	**9**	**9**
queen11_11	121	11	12	NA	17
anna	138	11	**11**	**11**	12
2-Insertions_4	149	4	**4**	5	5
queen13_13	169	13	14	NA	21
myciel7	191	8	NA	**8**	**8**
homer	561	13	14	**13**	15

Figure 1: Comparative results between Model, Tabucol and Greedy

The model performs better than Tabucol and greedy for some instances, although for some instances the model tends to underestimate its responses, erring on the minimum number of colours by predicting a lower number.

"Solving the graph coloring problem via hybrid genetic algorithms" [2] This paper proposed a hybrid genetic algorithm based on a local heuristic called DBG to give approximate values for the cromatic number of the graph. It achieves highly competitive results comparable with the best existing algorithms, but has a high runtime.

The method described in the article integrates a genetic algorithm (GA) with a local search heuristic known as DBG (Douiri and Elbernoussi, 2011) to tackle the graph coloring problem (GCP) in more detail. Initially, the algorithm starts by generating an initial population of individuals, where each individual represents a potential coloring configuration for the vertices of the graph. The number of colors initially used is often set to an arbitrary value, typically equal to the upper bound of the chromatic number of the graph. The objective function of the algorithm evaluates the quality of each individual solution by measuring the number of conflicts between adjacent vertices that have been assigned the same color. During the selection phase, individuals are chosen for reproduction based on their fitness, which is determined by the objective function. Higher fitness individuals are more likely to be selected for reproduction. The crossover operator takes two parent individuals from the population and combines parts of their

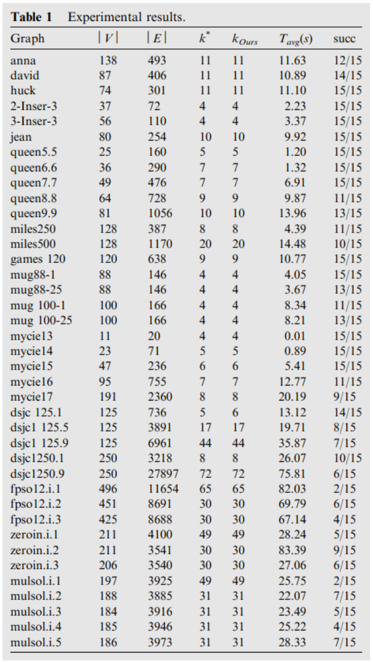


Table 1 Experimental results.

Graph	$\lvert V\rvert$	$\lvert E\rvert$	k^*	k_{Ours}	$T_{avg}(s)$	succ
anna	138	493	11	11	11.63	12/15
david	87	406	11	11	10.89	14/15
huck	74	301	11	11	11.10	15/15
2-Inser-3	37	72	4	4	2.23	15/15
3-Inser-3	56	110	4	4	3.37	15/15
jean	80	254	10	10	9.92	15/15
queen5.5	25	160	5	5	1.20	15/15
queen6.6	36	290	7	7	1.32	15/15
queen7.7	49	476	7	7	6.91	15/15
queen8.8	64	728	9	9	9.87	11/15
queen9.9	81	1056	10	10	13.96	13/15
miles250	128	387	8	8	4.39	11/15
miles500	128	1170	20	20	14.48	10/15
games 120	120	638	9	9	10.77	15/15
mug88-1	88	146	4	4	4.05	15/15
mug88-25	88	146	4	4	3.67	13/15
mug 100-1	100	166	4	4	8.34	11/15
mug 100-25	100	166	4	4	8.21	13/15
mycie13	11	20	4	4	0.01	15/15
mycie14	23	71	5	5	0.89	15/15
mycie15	47	236	6	6	5.41	15/15
mycie16	95	755	7	7	12.77	11/15
mycie17	191	2360	8	8	20.19	9/15
dsjc 125.1	125	736	5	6	13.12	14/15
dsjc1 125.5	125	3891	17	17	19.71	8/15
dsjc1 125.9	125	6961	44	44	35.87	7/15
dsjc1250.1	250	3218	8	8	26.07	10/15
dsjc1250.9	250	27897	72	72	75.81	6/15
fpso12.i.1	496	11654	65	65	82.03	2/15
fpso12.i.2	451	8691	30	30	69.79	6/15
fpso12.i.3	425	8688	30	30	67.14	4/15
zeroin.i.1	211	4100	49	49	28.24	5/15
zeroin.i.2	211	3541	30	30	83.39	9/15
zeroin.i.3	206	3540	30	30	27.06	6/15
mulsol.i.1	197	3925	49	49	25.75	2/15
mulsol.i.2	188	3885	31	31	22.07	7/15
mulsol.i.3	184	3916	31	31	23.49	5/15
mulsol.i.4	185	3946	31	31	25.22	4/15
mulsol.i.5	186	3973	31	31	28.33	7/15

Figure 2: Results of the Hybrid GA Approach

solutions at a randomly chosen crossover point to create offspring individuals. These offspring solutions undergo adjustments to ensure that they adhere to the constraints of the problem. Mutation introduces small changes to individual solutions to explore new regions of the solution space. In this method, mutation involves changing the color of a vertex in an existing solution, guided by the objective function to retain only beneficial changes. The genetic algorithm iterates through selection, crossover, and mutation operators, continually evaluating the fitness of resulting individuals. Additionally, the DBG local search heuristic is applied at various stages to further improve solution quality by reducing conflicts. The termination condition of the algorithm is met when a satisfactory solution with minimal conflicts (ideally, no conflicts) is found, or after reaching a predefined maximum number of iterations. By combining genetic algorithms with the DBG local search heuristic, the method efficiently explores the solution space of the graph coloring problem, aiming to find high-quality solutions within reasonable computational time.

The article assesses this new method's performance against existing algorithms on standard benchmark graphs. The results likely show it achieving competitive results, potentially finding the minimum number of colors for most graphs. This highlights the effectiveness of the combined genetic algorithm and local search heuristic. However, the efficiency of genetic algorithms can be sensitive to parameters, so for some complex graphs the solution might take longer or get stuck on suboptimal colorings. Overall, the research suggests this hybrid approach has promise for solving graph coloring problems, with further exploration needed to optimize its performance on various graph structures.

"A Discrete Firefly Algorithm Based on Similarity for Graph Coloring Problems" [3] The Firefly Algorithm, inspired by the flashing behavior of fireflies, is an optimization technique where fireflies in a search space represent potential solutions, attracted to brighter ones while adjusting their positions iteratively, aiming to find optimal or near-optimal solutions to various optimization problems. This paper proposes a novel non-hybrid discrete firefly algorithm for solving planar graph coloring problems. It discretizes the firefly algorithm using the *similarity* metric:

$$\text{similarity}_{ij} = 1 - \frac{H(x_i, x_j)}{n},$$

where x_i and x_j are two colorings.

The firefly algorithm is compared with HDPSO (Hamming Distance Particle Swarm Optimization) and HDABC (Hamming Distance Artificial Bee Colony). The proposed DFA obtains an excellent success rate. When applied to relatively small graphs, say $n = 90$, the success rate of DFA is higher than HDPSO and nearly equal well with DABC at $d = 2.5$ (where d is the constraint density), which is the most difficult problem. However, as the size of graph increases, DFA brings obvious advantages not only at $d = 2.5$, but also at $d = 2$ and $d = 3$. For example, the succcess rate of DFA is 78% when the graph size is 150 and $d = 2.5$. This is much higher than DABC's 20% and HDPSO's 1%. On the other hand, its evaluation times of solving the most difficult problems are much larger than HDPSO and DABC.

2 Benchmark instances

In order to test our methods, we will used a number of 79 test instances from various sources presented below representing different graph typologies. Of these, for 56 the minimum number of colours required for colouring is known, for the rest an optimal value has not yet been determined. The test instances have as origin:

- DSJ (from David Johnsom) - 15 instances: random graphs, where DSJC (12 instances) are standard random graphs, DSJR (2 instances) are geometric graphs and DSJR.c (1 instances) are complements of geometric graphs. For all this the optimum solution is not found.
- CUL (from Joe Culberson) - 6 instances: quasi-random coloring problems
- REG (from Gary Lewandowski) - 14 instances: Problem based on register allocation for variables in real codes.

- LEI (from Craig Morgenstern) - 12 instances: Leighton graphs with guaranteed coloring size.
- SCH (from Gary Lewandowski) - 2 instances: Class scheduling graphs, with and without study halls.
- LAT (from Gary Lewandowski) - 1 instance: Latin square problem.
- SGB (from Michael Trick and Donald Knuth) - 24 instances: these can be divided into:
  - Book Graphs - 5 instances: Given a work of literature, a graph is created where each vertex represents a character. Two vertices are connected by an edge if the corresponding characters encounter each other in the book. Knuth creates the graphs for five classic works: Tolstoy's Anna Karenina (anna), Dicken's David Copperfield (david), Homer's Iliad (homer), Twain's Huckleberry Finn (huck), and Hugo's Les Misérables (jean).
  - Game Graphs - 1 instance: A graph representing the games played in a college football season can be represented by a graph where the vertices represent each college team. Two teams are connected by an edge if they played each other during the season. Knuth gives the graph for the 1990 college football season.
  - Miles Graphs - 5 instances: These graphs are similar to geometric graphs in that vertices are placed in space with two vertices connected if they are close enough. These graphs, however, are not random. The vertices represent a set of United States cities and the distance between them is given by by road mileage from 1947. These graphs are also due to Kuth.
  - Queen Graphs - 13 instances: Given an n by n chessboard, a queen graph is a graph on n^2 vertices, each corresponding to a square of the board. Two vertices are connected by an edge if the corresponding squares are in the same row, column, or diagonal. Unlike some of the other graphs, the coloring problem on this graph has a natural interpretation: Given such a chessboard, is it possible to place n sets of n queens on the board so that no two queens of the same set are in the same row, column, or diagonal? The answer is yes if and only if the graph has coloring number n. Martin Gardner states without proof that this is the case if and only if n is not divisible by either 2 or 3. In all cases, the maximum clique in the graph is no more than n, and the coloring value is no less than n.
- MYC (from Michael Trick) - 5 instances: Graphs based on the Mycielski transformation. These graphs are difficult to solve because they are triangle free (clique number 2) but the coloring number increases in problem size.

3 Methods

3.1 Hybridized Genetic Algorithm

3.1.1 Representation of solutions

The representation of a solution is in the form of an integer array, the size of which is influenced by the number of vertices in the graphs (one value for vertex). Thus, for each verticle its color is retained, which has values between 0 and $no\_colors - 1$, where no_colors is the number of colors for which a solution is attempted. Also, for each solution we additionally retain its fitness.

3.1.2 Binary Search

The first part of this approach is a binary search on the result of the number of colours needed to colour the graph. This uses two parameters: *lower_bound* (the minimum number of colours for which a solution may exist) and *upper_bound* (the minimum number for which we know so far that a solution exists). Thus, we know that the value we are looking for is in this range. Initially, *upper_bound* is equal to *no_vertices* (the number of vertices), which is the worst case in which the graph is complete, and *lower_bound* is equal to 1, representing the best case in which the graph contains only isolated vertices (we assume that no input instance is the null graph, in which case the number of colors would be 0). For the number of colors at a given step generated by the binary search, an attempt will be made to create a solution using the following two parts.

3.1.3 Genetic Algorithm

The main part of our method is the genetic algorithm. It starts from an initial randomly generated population and applies for a predefined number of generations 3 operations inspired by nature:

1. Selection: involves choosing from the current population which solutions to keep and which to discard. In our approach, this is done by eliminating half of the population each generation, keeping only the highest quality solutions (quality determined by fitness function).

2. Crossover: involves choosing two individuals which genes are combined to produce a new one. The selection of the two individuals is done by randomly choosing 4 individuals and keeping the best 2, this assuring that the best genes are carried in future generation and also avoiding reaching a local minimum.

3. Mutation: involves changing the genes of a newly created individual with a certain probability (in the experiments we used 0.7). In the case of a mutation, we try to modify the solutions so that the quality of the solution increases as follows: for each vertex, if it has neighbours of the same colour, we try to change its colour to one not used by any of the neighbours. If this is not possible, the vertex is not changed.

The fitness function we use is a simple one, i.e. its value is given by the number of edges in the graph connecting two vertices of the same colour. Thus, the goal is to minimize it, an individual with fitness 0 being a solution.

3.2 GNN

This method was inspired from the paper called "Graph Coloring with Physics-Inspired Graph Neural Networks" [4] and it consists în implementing and testing the first version of the model proposed by the authors, the PI-GCN, a graph convolutional neural network. To determine the solution of the problem, a similar binary search to the solution before is used, and for each number of colors provided, the neural network is trained to find a solution.

3.2.1 Representation of solutions

To store each instance of a problem, an adjacency matrix is used, this being a binary matrix that contains on row i and column j True if there is an edge in the graph between vertexes i and j, or False otherwise. This method is very efective for checking if for a color asigned to a vertex if there any other adiacent vertexes with the same color, but it is also costly regarding the memory needed.

3.2.2 GNN

For the neural network part, we are using a Graph Convolutional Network as described in this paper [4], changing only the output layer in order to predict a soft assignment of colors for each vertex in the graph. At the end, this soft assignment is transformed into a hard one to produce the problem solution. Unfortunatelly, because at each iteration of the binary search the number of colors changes, so the network structure changes and a retraining of the entire model is needed. One improvement we can use in the future is using the already trained network weights from an iteration to another, thus may reducing the time needed for the retraining. Because the network convergence is determimed by the random assignment of the weights for now, we are using 3 tries to determine if there is a solution for the current number of colors, this also being time consumming.

3.2.3 Loss function

The loss function used in the paper is based on the Potts model from statistical physics. The energy function for a graph G with n nodes and q colors is defined as:

$$E(\mathbf{x}) = \sum_{(i,j)\in E} \delta(x_i, x_j)$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a color assignment, E is the set of edges, and $\delta(x_i, x_j)$ is the Kronecker delta function.

The corresponding loss function for the graph neural network (GNN) is:

$$\mathcal{L} = \sum_{(i,j)\in E} \sum_{c=1}^{q} p_{i,c} p_{j,c}$$

where $p_{i,c}$ is the probability of node i being assigned color c.

3.2.4 Hyperparameters

For our experiments, we tried several combinations of hyperparameters to find the combination that produces the best results, finally using a learning rate of $1e-4$, a hidden layer size of 64 neurons, a dropout rate of 0.1 and a number of 100000 epochs. To increase the speed of the model, we added two early stop conditions: the first is to stop training when a solution is found and continue with a smaller number of colors, and the second is to introduce a patience parameter that signifies the number of epochs since no significant change in the loss function has occurred. Thus, if the loss function does not decrease by more than $1e-3$ (tolerance) during 20000 epochs (patience), the training stops and continues with a higher number of colours. The number of colours is another hyperparameter of the network, provided by the binary search presented above.

4 Improvements

4.1 Genetic Algorithm

To improve the results of the genetic algorithm, we looked for ways to combat blocking in local minima by making the following modifications in the late stages of the algorithm (after the fitness score of one individual drops under 5):

- Selection: We select the individual with the highest fitness score.
- Crossover: We do not perform crossover at this stage at all.
- Mutation: For the selected individual, we choose a random vertex and we change its color to a random one, different than the one already assigned.

If the fitness score increses above 5, the algorithm returns to its normal execution.

In addition to this, we have added a refresh step to increase the explorability of the algorithm by using a threshold representing the number of generations in which the algorithm must find a better solution than the existing one. If an improvement is not found, three quarters of the individuals in the order of scores are eliminated (thus keeping the best solutions).

One other improvement we added is the Wisdom of Artificial Crowds component. This part occurs if the genetic algorithm has failed to find an optimal solution and works in the following way: it extracts the best individual from the population and tries to replace the color of each vertex that is adjacent to it with a vertex of the same color with the color that occurs most often in that position in the half of the population with the best fitness. This takes advantage of the fact that knowing a group is better than knowing each individual in the group and can lead to finding an optimal solution.

We also tried to implement a wheel of fortune selection, but it did not improve our results.

4.2 GNN

A major improvement we have made to this method is to address the problem of needing to retrain the model from scratch for each instance, a problem addressed above which significantly increases the time required for convergence. In this regard, we have two types of transfer learning that we use:

- Local Transfer: During the training for an instance with the previous version, we needed to train a model for each number of colors provided by the binary search from scratch because we always changed the output layer of the network (dependend on the number of colors used). The improvement we added was that for each iteration of the binary search, we use an already trained model from a previous iteration, changing the output layer and reinitializing it, thus reducing the time needed for the convergence by a lot.

- Global Transfer: During the training for multiple instances, we trained models from scratch because not just the output layer changed, but the input as well, but this can be changed because the problem we need to solve is the same no matter the instance. To solve this, similar to the local transfer part, we use a model trained from previous instances but we change and reinitialize the input and the output layer, thus taking advantage from multiple instances training and improving the time needed to find a solution for a new instance.

Another component we added that provided a significant boost in results is adding residual connections in our model, thus combating the vanishing gradient problem and allowing us to try deeper architectures. We also added a scheduler to encourage exploration at the beginning of the train (when the loss is high) and exploitation in the late stages (to prevent overshooting the local best).

One more tehnique we considered adding was to add another algorithm at the end of the network training (similar to the Wisdom of Artificial Crowds at the end of the GA) to try to reach the solution (for example adding a HillClimbing), but we remained to this stage to properly view our model efficiency.

5 Results and Comparison

We performed the experiments in the following way: for GA, we performed for each instance a binary search on the result to determine the number of colors, and for each color we ran the algorithm 5 times to determine if there is any possible valid coloring. For GNN, we worked in a similar way, running the algorithm 3 times for each color due to the resource constraints we faced. Our results are summarized in the tables below, both in terms of performance and resource efficiency.

5.1 Base Results and Time Comparison

Graph	Optimal	Prediction GA	Time GA	Prediction GNN	Time GNN
anna	11	11	$\approx 1s$	21	$\approx 30min$
david	11	11	$\approx 1s$	17	$\approx 2min$
huck	11	11	$\approx 1s$	15	$\approx 2min$
jean	10	10	$\approx 1s$	10	$\approx 1min$
myciel4	5	5	$\approx 1s$	5	$\approx 1min$
myciel6	7	7	$\approx 1s$	7	$\approx 3min$
myciel7	8	8	$\approx 1s$	9	$\approx 8min$
queen5_5	5	5	$\approx 1s$	5	$\approx 3min$
queen8_8	9	11	$\approx 10min$	14	$\approx 60min$
queen9_9	10	13	$\approx 13min$	17	$\approx 80min$
queen11_11	11	14	$\approx 40min$	19	$\approx 3h$
queen13_13	13	17	$\approx 70min$	24	$\approx 5h$
fpsol2.i.3	30	33	$\approx 20min$	37	$\approx 90min$
inithx.i.2	31	31	$\approx 1s$	31	$\approx 1min$
mulsol.i.2	31	31	$\approx 1s$	31	$\approx 2min$
mulsol.i.4	31	31	$\approx 1s$	31	$\approx 2min$
mulsol.i.5	31	31	$\approx 1s$	31	$\approx 3min$
zeroin.i.2	30	30	$\approx 1s$	30	$\approx 2min$
zeroin.i.3	30	30	$\approx 1s$	30	$\approx 3min$
DSJC250.5	28(?)	36	$\approx 40min$	47	$\approx 4h$
DSJC500.1	12(?)	16	$\approx 20min$	31	$\approx 3h$
DSJR500.5	122(?)	144	$\approx 90min$	212	$\approx 10h$
flat300_28_0	28	34	$\approx 20min$	46	$\approx 50min$
miles1500	73	73	$\approx 1s$	73	$\approx 5min$

Table 1: Graph Coloring Results Base Methods

5.2 Final Results and Time Comparison

Graph	Optimal	Prediction GA	Time GA	Prediction GNN	Time GNN
anna	11	11	$\approx 1s$	11	$\approx 1s$
david	11	11	$\approx 1s$	11	$\approx 1s$
huck	11	11	$\approx 1s$	11	$\approx 1s$
jean	10	10	$\approx 1s$	10	$\approx 1s$
myciel4	5	5	$\approx 1s$	5	$\approx 1s$
myciel6	7	7	$\approx 1s$	7	$\approx 1s$
myciel7	8	8	$\approx 1s$	8	$\approx 30s$
queen5_5	5	5	$\approx 1s$	5	$\approx 1min$
queen8_8	9	9	$\approx 25min$	9	$\approx 30min$
queen9_9	10	10	$\approx 30min$	11	$\approx 50min$
queen11_11	11	12	$\approx 60min$	14	$\approx 80min$
queen13_13	13	14	$\approx 90min$	17	$\approx 2h$
fpsol2.i.3	30	30	$\approx 40min$	32	$\approx 80min$
inithx.i.2	31	31	$\approx 1s$	31	$\approx 1s$
mulsol.i.2	31	31	$\approx 1s$	31	$\approx 1s$
mulsol.i.4	31	31	$\approx 1s$	31	$\approx 1s$
mulsol.i.5	31	31	$\approx 1s$	31	$\approx 10s$
zeroin.i.2	30	30	$\approx 1s$	30	$\approx 1s$
zeroin.i.3	30	30	$\approx 1s$	30	$\approx 3s$
DSJC250.5	28(?)	31	$\approx 90min$	34	$\approx 2h$
DSJC500.1	12(?)	13	$\approx 35min$	15	$\approx 60min$
DSJR500.5	122(?)	122	$\approx 110min$	122	$\approx 2h30min$
flat300_28_0	28	29	$\approx 30min$	31	$\approx 50min$
miles1500	73	73	$\approx 1s$	73	$\approx 30s$

Table 2: Graph Coloring Results Final Methods

References

[1] H. Lemos, M. Prates, P. Avelar, and L. Lamb, “Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems,” 2019.

[2] S. M. Douiri and S. Elbernoussi, “Solving the graph coloring problem via hybrid genetic algorithms,” *Journal of King Saud University-Engineering Sciences*, vol. 27, no. 1, pp. 114–118, 2015.

[3] K. Chen and H. Kanoh, “A discrete firefly algorithm based on similarity for graph coloring problems,” in *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2017, pp. 65–70.

[4] M. J. Schuetz, J. K. Brubaker, Z. Zhu, and H. G. Katzgraber, “Graph coloring with physics-inspired graph neural networks,” *Physical Review Research*, vol. 4, no. 4, p. 043131, 2022.