Heuristic Colour Assignment Strategies for Merge Models in Graph Colouring

István Juhos[1], Attila Tóth[2], and Jano I. van Hemert[3]

[1] Dept. of Computer Algorithms and Artificial Intelligence, Univ. of Szeged, Hungary
[2] Department of Computer Science, Univ. of Szeged (JGYTFK), Hungary
[3] Centre for Emergent Computing, Napier University, Edinburgh, UK

Abstract. In this paper, we combine a powerful representation for graph colouring problems with different heuristic strategies for colour assignment. Our novel strategies employ heuristics that exploit information about the partial colouring in an aim to improve performance. An evolutionary algorithm is used to drive the search. We compare the different strategies to each other on several very hard benchmarks and on generated problem instances, and show where the novel strategies improve the efficiency.

1 Introduction

The problem class known as the graph k-colouring problem [1] is defined as follows. Given a graph $G = \langle V, E \rangle$, where $V = \{v_1, \ldots, v_n\}$ is a set of nodes and $E = \{(v_i, v_j) | v_i \in V \wedge v_j \in V \wedge i \neq j\}$ is a set of edges. The objective in the graph k-colouring problem is to colour every node in V with one of k colours such that no two nodes connected with an edge in E have the same colour. Such a colouring is called a *valid colouring.* The smallest number of colours k used to achieve a valid colouring of G is called the *chromatic number* of G, which is denoted by χ.

Most algorithms searching for a solution of a graph k-colouring problem do so by incrementally assigning a colour to a node. Consequently, at every node visited a decision must be made which colour to assign. This choice may prove of vital importance to achieving a valid colouring. Quite a number of strategies for making this decision exist, and each one comes with its own rationale and benefits [2–Chapter 5]. From recent theoretical developments [3] we know that algorithms for finites sets of problems under permutation closure also cannot escape the No Free Lunch Theorem [4], which makes it more important to link properties of problems with algorithms [5].

A great deal of study is devoted to hybrid algorithms, as these have proven to be successful approaches to solving difficult constrained optimisation problems. Popular methodologies are meta-heuristics [6] and, more recently, hyper-heuristics [7] and hybrid meta-heuristics, where the idea is to use combine several heuristics to get a more successful algorithm. This success is measured in both effectiveness and efficiency, i.e., accuracy in finding solutions and time complexity. Here, we examine the combination of a powerful representation for graph colouring with five different heuristic strategies for colour assignment.

In the next section we explain how solutions of graph colouring problems can be represented using merge models. Then, in Section 3 we explain different heuristic strategies for colour assignment. These strategies are used in an evolutionary algorithm explain in Section 4. The different heuristic strategies are benchmarked in Section 5 together with other algorithms. Finally in Section 6 we draw some conclusions.

2 The Binary Merge Model Representations

Graph colouring algorithms make use of adjacency checking during the colouring process, which has a large influence on the performance. Generally, when assigning a colour to a node, all adjacent nodes must be scanned to check for potential violations. Thus, a number of constraint checks, i.e., checks for equal colours, need to be performed. The exact number of constraint checks performed is bounded by the current number of coloured neighbours and by $|V| - 1$. Using the Binary Merge Model approach, explained next, the number of constraint checks lies between one and the number of colours used up to this point. These bounds arise from the model-induced hyper-graph structure, which guarantees that the algorithms usually performs better .

The *Binary Merge Model* (BMMs) implicitly uses hyper-nodes and hyper-edges (see Figure 1). A hyper-node is a set of nodes that all have the same colour, i.e., they are folded into one node. A hyper-edge connects nodes of which at least one node is a hyper-node. Such a hyper-edge essentially forms a collection of regular edges, i.e., constraints. A hyper-edge only exists if and only if its corresponding nodes are connected by at least two "normal" edges. The BMM concentrates on operations on hyper-nodes and normal nodes, by trying to merge normal nodes with normal nodes and hyper-nodes. Within the context of search effort, which in constraint satisfaction is measured by counting the number of constraint checks, we can save effort if at least one of the nodes is a hyper-node. Then, the number of adjacency checks, i.e, constraint checks, can be reduced as these are performed along hyper-edges instead of normal edges, because one constraint check on a hyper-edge saves at least one, but possible more, constraint checks on the normal edges it incorporates. A more detailed explanation is given in [8], where the model was introduced.

The current colouring of a graph $\langle V, E\rangle$ is stored in a Binary Merge Table (BMT) (for an example, see Figure 2). Every cell (i, j) in the table is binary. The columns refer to the nodes and the rows refer to the colours. A value in cell (i, j) is zero if and only if node $j \in V$ cannot be assigned colour i because of the edges E in the original graph $\langle V, E\rangle$. The initial BMT is the adjacency matrix of the graph, hence a different colour is assigned to each node.

If the graph is not a complete graph, then it might be possible to reduce the number of necessary colours. This corresponds to reducing rows in the BMT. Rows can be reduced by repeatedly using a Binary Merge Operation, which attempt to merge two rows. If a merge is possible, i.e., no violations are introduced, the number of colours is decreased by one. Otherwise, the number of colours remains the same. A merge is successful only when two nodes are not connected by a normal edge or a hyper-edge. An example of both a successful and unsuccessful merge is shown in Figures 1 and 2.

Definition 1. *The Binary Merge Operations* $\cup$ *merges an initial row* r_j *into an arbitrary (initial or merged) row* r_i *if and only if* $(j, i) = 0$ *(i.e. the hyper-node* x_j *is not connected to the node* x_i*) in the* BMT*. If rows* r_i *and* r_j *can be merged then the result is the union of them.*

Formally, let I be the set of initial rows of the BMT and R be the set of all possible $|V|$ size rows, i.e. binary vectors. Then an merge operation is defined as

$$\begin{aligned} \cup &: R \times I \to R \\ r'_j &:= r_j \cup r_i, \quad r'_j, r_j \in R, \quad r_i \in I, \text{ or by components} \\ r'_j(l) &:= r_j(l) \vee r_i(l), \quad l = 1, 2, \ldots, |V| \end{aligned}$$

With regard to the time complexity of the binary operation, it is proportional to a binary OR operation on a register of l bits. If l is the number of bits in one such operation and, under assumption that the time complexity of that operation is one, the merge of two rows of length n by l length parts takes $\lceil n/l \rceil$ to complete. If k is the number of rows left in the BMT, then the number of merge operations is $|V| - k$, where $k \in \{\chi, \ldots, |V|\}$.

2.1 Permutation Merge Model

Finding a minimal colouring for a graph k-colouring problem using the binary merge table representation requires finding the sequence of merge operations that leads to that colouring. This can be represented as a sequence of candidate reduction steps using the greedy approach described above. The permutations of this representation form the *Permutation Merge Model* [8].

The result of colouring a graph after two or more merge operations depends on the order in which these operations were performed. Consider the hexagon in Figure 1(a) and its corresponding BMT in Figure 2. Now let the sequence $P_1 = 1, 4, 2, 5, 3, 6$ be the order in which the rows are considered for the merge operations and consider the following merging procedure. Take the first two rows in the sequence, then attempt to merge row 4 with row 1. As these can be merged the result is $1 \cup 4$ (see Figure 1(b)). Now take row 2 and try to merge this with the first row, i.e. $(1 \cup 4)$. This is unsuccessful, so row 2 remains unaltered. The merge operations continue with the next rows 5 and 3, and finally, with 6. The allowed merges are $1 \cup 4$ and $2 \cup 5$. This sequence of merge operations results in the 4-colouring of the graph depicted in Figure 1(c). However, if we use the sequence $P_2 = 1, 4, 2, 6, 3, 5$ then the result will be only a 3-colouring, as shown in Figure 1(e) with the merges $1 \cup 4$, $2 \cup 6$ and $3 \cup 5$. The defined merge is

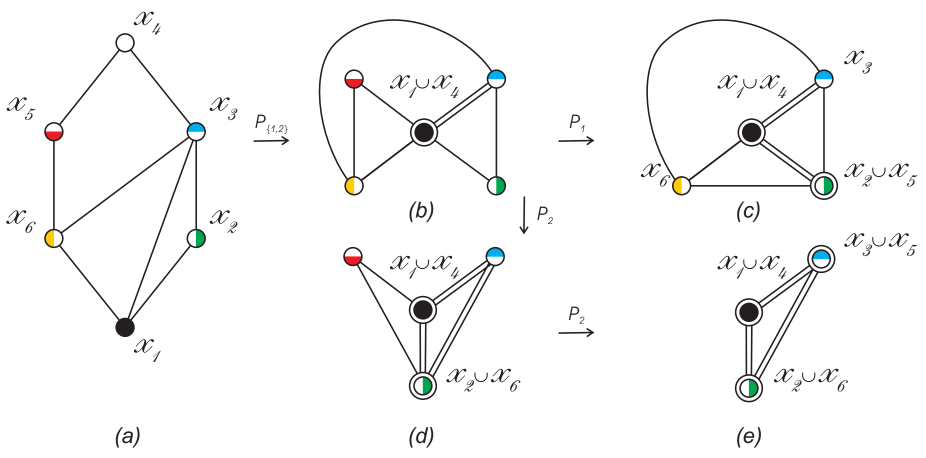


Fig. 1. Examples of the result of two different merge orders $P_1 = 1, 4, 2, 5, 3, 6$ and $P_2 = 1, 4, 2, 6, 3, 5$. The double-lined edges are hyper-edges and double-lined nodes are hyper-nodes. The P_1 order yields a 4-colouring (c), but with the P_2 order we get a 3-colouring (e).

greedy, i.e. it takes a row and tries to find the first row from the top of the table that it can merge. The row remains unaltered if there is no suitable row. After performing the sequence P of merge operations, we call the resulting BMT the *merged* BMT.

(a)	x_1	x_2	x_3	x_4	x_5	x_6
r_1	0	1	1	0	0	1
r_2	1	0	1	0	0	0
r_3	1	1	0	1	0	1
r_4	0	0	1	0	1	0
r_5	0	0	0	1	0	1
r_6	1	0	1	0	1	0

(b)	x_1	x_2	x_3	x_4	x_5	x_6
$r_1 \cup r_4$	0	1	1	0	1	1
r_2	1	0	1	0	0	0
r_3	1	1	0	1	0	1
r_5	0	0	0	1	0	1
r_6	1	0	1	0	1	0

(c)	x_1	x_2	x_3	x_4	x_5	x_6
$r_1 \cup r_4$	0	1	1	0	1	1
$r_2 \cup r_5$	1	0	1	1	0	1
r_3	1	1	0	1	0	1
r_6	1	0	1	0	1	0

(d)	x_1	x_2	x_3	x_4	x_5	x_6
$r_1 \cup r_4$	0	1	1	0	1	1
$r_2 \cup r_6$	1	0	1	0	1	0
r_3	1	1	0	1	0	1
r_5	0	0	0	1	0	1

(e)	x_1	x_2	x_3	x_4	x_5	x_6
$r_1 \cup r_4$	0	1	1	0	1	1
$r_2 \cup r_6$	1	0	1	0	1	0
$r_3 \cup r_5$	1	1	0	1	0	1

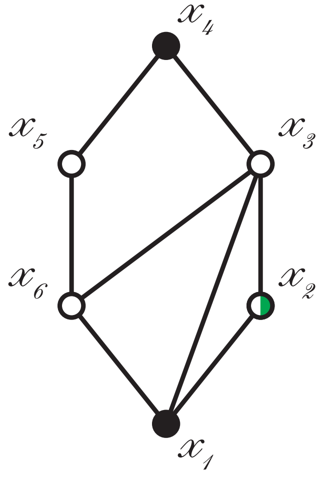
Fig. 2. Binary Merge Tables corresponding to the graphs in Figure 1.

In practice the graphs start out uncoloured, the colouring is then constructed by colouring the nodes in steps. We deal with the sub-graphs of the original graph defined by the colouring steps. The related binary merge tables contain partial information about the original one. Let the original graph with its initial BMT be defined by Figure 3(a) on which the colouring will be performed. Taking the $x_1, x_4, x_2, x_6, x_3, x_5$ order of the nodes into account for colouring G, then using the ordering $P_1 = 1, 4, 2, 6, 3, 5$, an attempt will be made to merge rows. After the greedy colouring of the nodes x_1, x_4, x_2 there is a related partial or sub-BMT along with the (sub-)hyper-graph. These are depicted in Figure 3(b). The 1st and the 4th row are merged together, but the 2nd cannot be merged with the $1 \cup 4$ merged row, thus the 2nd row remains unaltered in the related sub-BMT.

From here on, we concentrate on how to extract valuable information from the sub-structures to get an efficient colour assignment strategy for the the nodes, which takes into account the current state of the colouring. This as opposed to the usually greedy manner, which is blind in this sense, i.e., it does not consider the current environment.

(a)

	x_1	x_2	x_3	x_4	x_5	x_6
r_1	0	1	1	0	0	1
r_2	1	0	1	0	0	0
r_3	1	1	0	1	0	1
r_4	0	0	1	0	1	0
r_5	0	0	0	1	0	1
r_6	1	0	1	0	1	0



(b)

	x_1	x_2	x_3	x_4	x_5	x_6	
$r_1 \cup r_4$	0	1	1	0	1	1	4
r_2	1	0	1	0	0	0	2

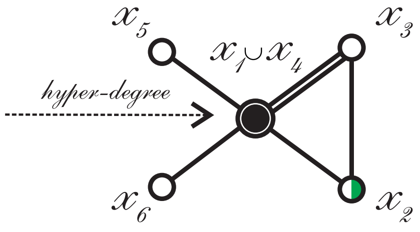


Fig. 3. The left side shows the partial colouring of the G graph according to the x_1, x_4, x_2 greedy order and the adjacency matrix of the graph. The right one shows the partial or sub-BMT related to this colouring with its co-structures and sub-BMT induced hyper-graph.

3 Heuristic Strategies for Colour Assignment

Finding the appropriate node order is important, which will be left to the evolutionary algorithm described in Section 4. However, the choice of which colour to assign also has much influence on the success of the final algorithm. The greedy strategy finds the first available colour for assigning it to the node currently being coloured. It does so by trying to merge the corresponding $r_{P(i)}$ row of the BMT to the previously merged rows following the natural order $1, 2, \ldots, i-1$.

Instead of using a simple greedy procedure, we will employ a number of more sophisticated ways for assigning colours. The way in which the merge model reduces the number of colours leaves open the way in which the colours are assigned to various groups of nodes. We take advantage of this by employing different heuristic strategies for the assigning colours to nodes. Formally, let $x_{P(i)}$ be the next node to be coloured, where i is the index of the node in the permutation P. Next, we provide several strategies for assigning a colour to $x_{P(i)}$ using the information provided by the current merge model. The first two strategies use only information about the already coloured structure, i.e., about the sub-BMT not dealing with the current node. The remaining, novel strategies, use information about the current node and its context so far, and then try to exploit this information by using it to avoid getting stuck later on.

3.1 Hyper-node Cardinality

This strategy attempts to merge row i with its preceding rows by favouring hyper-nodes with high cardinality. A hyper-node's cardinality is defined by the number of normal nodes it encompasses. The strategy consists of first colouring the hyper-node with the highest cardinality. In other words, choosing the colour that colours the most nodes and gives valid colouring. Although this strategy has a greedy component to it, its ability to use knowledge on-line, i.e., while searching for a solution, may give it an edge over the simple greedy method.

3.2 Hyper-node Constrainedness

While the previous heuristic supposes that the hyper-node cardinality determines its constrainedness, this one expresses it in a direct way examining the context of the considered hyper-node. With this heuristic, we favour the most constrained hyper-nodes. The intuition is to avoid the possibility that the least constrained nodes pick up too many irrelevant nodes. This method also works for the BMT, where we count all the connecting hyper-edges, so calculating the hyper-degree (see Figure 3). This information can easily be obtained by summarising the rows, i.e., by making the first order norm $||r_{P(i)}||_1$ of them in the sub-BMT.

3.3 Suitable Matches

Up until now, we did not consider the characteristics of the structure of $x_{P(i)}$. The previous strategy used only the constrainedness of hyper-nodes, and here we shall use the $x_{P(i)}$ constraints as well to find a suitable match for merging rows. We say that the m-th vector of the sub-BMT $r'(m)$ is the most suitable candidate for merging with $r_{P(i)}$ if they share the most constraints. The dot product of two vectors provides the number of shared constraints. Thus, by reverse sorting all the sub-BMT vectors on their dot product with $r_{P(i)}$, we can reduce the number of colours by merging $r_{P(i)}$ with the most suitable match.

These approach include implicitly the least constraining value heuristic [2–Chapter 5], but provide additional one. Try to find that hyper-nodes (group of the nodes) which has the most number of common neighbors. Thus, reducing implicitly the structure of the graph in a way which is explicit described in [9].

3.4 Topological Similarity

The dot product, as described above, provides a measure for the similarity of the vectors. If we normalise these vectors by their length, the result is a measure for similarity in a topological sense. As the normalised dot products gives the cosines of the angles of the vectors, higher cosines corresponds to vectors located nearer. This strategy exploits this idea by collecting vectors that are spatially near to each other. By performing merge operations on these collections we get convex combinations of vectors. Thus, the result of a merge remains in the span of the merged vectors. The idea behind this is to carefully combine similar groups of colours, thereby building up a solid colouring of the graph that leaves enough room for further merging of groups, i.e., rows in the BMT.

4 Evolutionary Algorithm to Guide the Models

We have two goals. The first is to find a successful order of the nodes and the second is to find a successful order for assigning colour. While the order of the node can be represented by a fix length permutation, the order for colour assignment needs a variable length representation. We turn to the heuristics described above to guide the colour assignment dynamically. For the first goal, we must search the permutation search space of the model described in Section 2.1, which is of size $n!$. Here, we use an evolutionary algorithm to search through the space of permutations. The genotype consists of the permutations of the nodes, i.e., rows of the BMT. The phenotype is a valid colouring of the graph after using a colour assignment strategy on the permutation to select the order of the binary merge operations.

An intuitive way of measuring the quality of an individual p in the population is by counting the number of rows remaining in the final BMT. This equals to the number of colours $k(p)$ used in the colouring of the graph, which needs to be minimised. When we know that the optimal colouring is χ then we may normalise this fitness function to $g(p) = k(p) - \chi$. This function gives a rather low diversity of fitnesses of the individuals in a population because it cannot distinguish between two individuals that use an equal number of colours. This problem is called the fitness granularity problem. We address it by introducing a new fitness, which relies on the heuristic that one generally wants to avoid highly constraint nodes and rows in order to have a higher chance of successful merges at a later stage. It works as follows. After the final merge the resulting BMT defines the colour groups. There are $k(p) - \chi$ over-coloured nodes, i.e., merged rows. Generally, we use the indices of the over-coloured nodes to calculate the number of nodes that need to be minimised. But these nodes are not necessarily responsible for the over-coloured graph. Therefore, we choose to count the nodes that are in the smallest group of nodes with the same colour. In the context of the merge model, this corresponds to hyper-nodes with the smallest cardinality. To cope better with the fitness granularity problem we should also deal with the constraints causing high constrainedness. The final fitness function is then defined as follows. Let $\zeta(p)$ denote the number of constraints, i.e., ones, in the rows of the final BMT that belong to the $k(p)-\chi$ hyper-nodes having the smallest cardinality. The fitness function becomes $f(p) = (k(p) - \chi)\zeta(p)$.

Note that here the cardinality of the problem is known, and used as a stopping criterium ($f(p) = 0$) to determine the efficiency of the algorithm. For the case where we do not know the cardinality of the problem, this approach can be used by leaving out the normalisation step.

5 Empirical Comparison

We use a generational model with 2-tournament selection and replacement, where we employe elitism of size one. The stop condition is that either an individual p exists with $f(p) = 0$ or that the maximum number of generations of 6 000 generations is reached (twice as many as in a previous study due to the harder problems here). The latter means that the run is unsuccessful, i.e., the optimal colouring is not found. This setting is used in all experiments. The initial population is created with 100 random individual. Two variation operators are used to provide offsprings. First, the 2-point order-based crossover (OX2) [10–in Section C3.3.3.1] is applied. Second, the other variation operator is a simple swap mutation operator, which selects at random two different items in the permutation and then swaps. We use simple operators to make sure that any success gained, stems from the heuristic strategies for colour assignment. The probability of using OX2 is set to 0.4 and the probability for using the simple swap mutation is set to 0.6. These values are take from a previous study [8].

5.1 Means of Comparisons

The performance of an algorithm is expressed in its effectiveness and its efficiency in solving a problem instance. The first is measured using the success ratio, which is the amount of runs where an algorithm has found the optimum divided by the total number of runs. The second is measured by keeping track of how many constraint checks are being performed on average, for a successful run. This measure is independent of hardware and programming language as it counts the number of times an algorithm requests information about the problem instance, e.g., it checks if an edge exists between two nodes in the graph. This check, or rather the number of times it is performed, forms the largest amount of time spend by any constraint solver. A *constraint check* is defined for each algorithm as checking whether the colouring of two nodes is allowed (satisfied) or not allowed (violated). An evolutionary algorithm is of stochastic nature. Therefore, we always perform ten independent runs with different random seeds for each problem instance. Results are averaged over these runs and, where appropriate, over multiple instances with equal characteristics.

5.2 Benchmarks

We compare the five different strategies on a number of benchmark problems from the "The Second DIMACS Challenge" [11], which is a standard competition repository. For demonstration purposes we choose the extremely difficult Leighton graphs [12]. In a previous study [13], these graphs took 10–20 hours to be solved by specialised evolutionary solvers, due to the structure induced during their creation. Also, the failure of the well-known heuristic DSATUR of Brélaz [14] confirms the difficulty of these problem. They are random graphs on $n = 450$ nodes with an edge density of 0.25. A graph is constructed by first generating cliques of varying sizes in such a way that the pre-specified value of χ is not violated. Here we use $\chi = 15$. They are identified by `le450-15x`, where x is one of a, b, c, or d, to differentiate the individual instances.

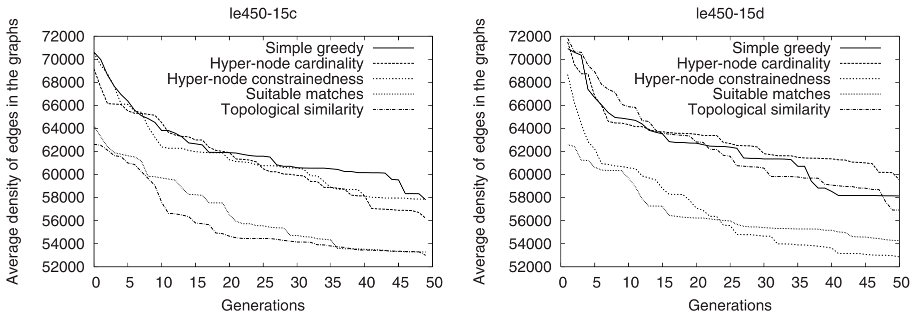


Fig. 4. Convergence graphs of the fitness for all five strategies on two hard problems

In Figure 4, the fitness of the best individual in the population is presented for each generation. These results are averaged over ten runs. This provides insight into the convergence when employing the different heuristic strategies. Not much difference exists between the simple greedy strategy and both the hyper-node cardinality and hyper-node constrainedness. However, we notice a significant faster convergence for both the suitable matches and the topological similarity. For the graph `le450-15c`, the convergence of the topological similarity is slightly faster than that of the suitable matches. Observing the starting phase for the `le450-15d`, the topological similarity shows large improvements, however after the good starting the suitable matches catches it up.

5.3 In the Phase Transition

Using the well known graph k-colouring generator of Culberson [15], we generate a test suite of 3-colourable graphs with 200 nodes. The edge density of the graphs is varied in a region called the phase transition. This is where hard to solve problem instances are generally found, which is shown using the typical easy-hard-easy pattern. The graphs are all equipartite, which means that in a solution each colour is used approximately as much as any other. The suite consists of nine groups where each group has five instances, one each instance we perform ten runs and calculate averages over these 50 runs. The connectivity is changed from 0.020 to 0.060 by steps of 0.005 over the groups.

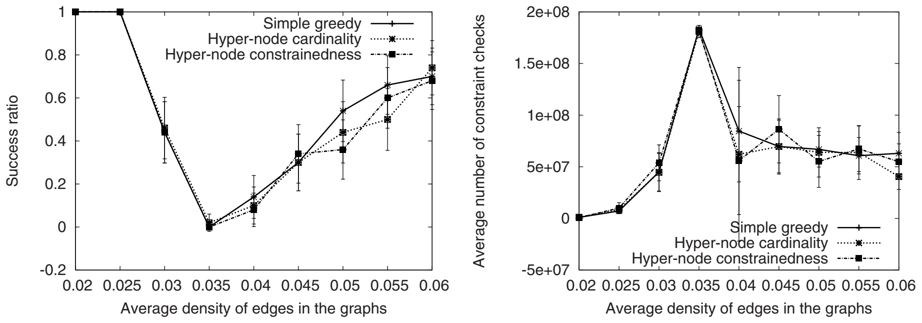


Fig. 5. Success ratio and average constraint checks to a solution for the simple greedy strategy, the hyper-node cardinality strategy, and the hyper-node constrainedness strategy (with 95% confidence intervals; for 0.035, the success ratio is too low thus we include all runs)

Figure 5 shows the performance measured by success ratio and by average constraint checks performed for the simple greedy strategy and the two strategies that restrict their on-line heuristics to the current node under consideration for colouring. No significant improvement is made over the simple greedy method.

The two novel strategies that employ knowledge over the colouring of the graph made so far are shown in Figure 6 together with the simple greedy strategy. Here we clearly notice an improvement in both efficiency and effectiveness over the simple greedy strategy. Especially, the search effort needed for denser graphs is much lower. Furthermore, the confidence intervals for this range are small and non-overlapping. These two approaches give a much robuster algorithm for solving graph k-colouring.

6 Conclusions

We have combined a powerful representation for the graph k-colouring problem, called the permutation merge model, with several heuristic strategies for colour

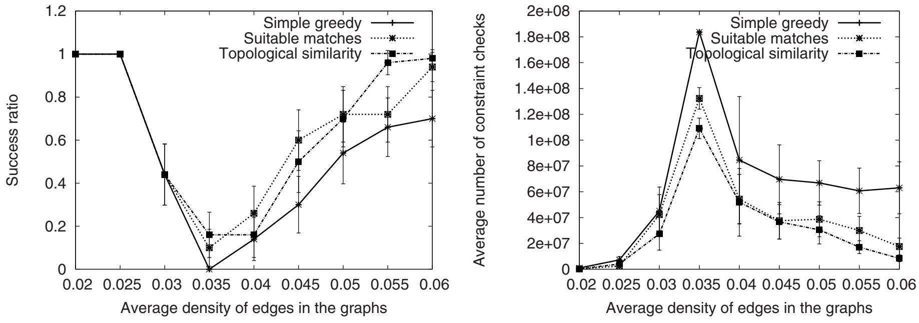


Fig. 6. Success ratio and average constraint checks for the simple greedy strategy, the suitable matches, and the topological similarity (with 95% confidence intervals for 0.035, the success ratio is too low thus we include all runs)

assignment. Four novel strategies use information about the current state of the colouring of the graph to infer where problems can be expected in a future stage of the colouring process. The aim of exploiting this knowledge is to improve performance by increasing the efficiency and effectiveness of the evolutionary algorithm that uses the permutations merge model.

By comparing the different strategies on several hard to solve problems, we showed how employing on-line heuristics improves the convergence speed of the evolutionary algorithm. Furthermore, the two novel strategies, by exploiting the suitability of matches and the topological similarity, showed more potential then the two strategies that restrict to using knowledge about the current node only.

In order to get a strong comparison, we compared all the strategies on a suite of generated problem instances that encompasses the phase transition. This way we ensure a comparison on very hard to solve problems. This confirmed the results on the benchmarks, as the two novel strategies are more effective, i.e., had a higher success ratio, on the right side of the phase transition. Also, they were far more efficient, and more consistent in their efficiency.

Acknowledgements

This work was supported by the Hungarian National Information Infrastructure Development Program through High Performance Supercomputing as the project CSPAI/1066/2003-2004. The third author is supported by a TALENT-Stipendium awarded by the Netherlands Organization for Scientific Research (NWO).

References

1. Jensen, T., Toft, B.: Graph Coloring Problems. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc (1995)

2. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. second edn. Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, New Jersey (1995)
3. Schumacher, C., Vose, M., Whitley, L.: The no free lunch and problem description length. In Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E., eds.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Morgan Kaufmann (2001) 565–570
4. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1** (1997) 67–82
5. Culberson, J.: On the futility of blind search: An algorithmic view of "No Free Lunch". Evolutionary Computation **69** (1998) 109–128
6. Glover, F., Kochenberger, W., Gary, A.: Handbook of Metaheuristics. Volume 57 of International Series in Operations Research and Management Science. Kluwer (2003)
7. Burke, E., Kendall, G., Soubeiga, E.: A tabu-search hyper-heuristic for timetabling and rostering. Journal of Heuristics **9** (2003) 451–470
8. Juhos, I., Tóth, A., van Hemert, J.: Binary merge model representation of the graph colouring problem. In Gottlieb, J., Raidl, G., eds.: Evolutionary Computation in Combinatorial Optimization. (2004) 124–134
9. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the Really Hard Problems Are. In: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia. (1991) 331–337
10. Bäck, T., Fogel, D., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. Institute of Physics Publishing Ltd, Bristol and Oxford University Press (1997)
11. Johnson, D., Trick, M.: Cliques, Coloring, and Satisfiability. American Mathematical Society, DIMACS (1996)
12. Leighton, F.T.: A graph colouring algorithm for large scheduling problems. J. Res. National Bureau Standards **84** (1979) 489–503
13. Dimitris Fotakis, Spyros Likothanassis, S.S.: An evolutionary annealing approach to graph coloring. In: Proceedings of Applications of Evolutionary Computing, EvoWorkshops 2001. (2001) 120–129
14. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM **22** (1979) 251–256
15. Culberson, J.: Iterated greedy graph coloring and the difficulty landscape. Technical Report TR 92-07, University of Alberta, Dept. of Computing Science (1992)