Improving Graph Colouring Algorithms and Heuristics Using a Novel Representation

István Juhos[1] and Jano I. van Hemert[2]

[1] Department of Computer Algorithms and Artificial Intelligence,
University of Szeged, Hungary, P. O. Box 652., Hungary
juhos@inf.u-szeged.hu
[2] Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Vienna, Austria
jano@vanhemert.co.uk

Abstract. We introduce a novel representation for the graph colouring problem, called the Integer Merge Model, which aims to reduce the time complexity of an algorithm. Moreover, our model provides useful information for guiding heuristics as well as a compact description for algorithms. To verify the potential of the model, we use it in DSATUR, in an evolutionary algorithm, and in the same evolutionary algorithm extended with heuristics. An empiricial investigation is performed to show an increase in efficiency on two problem suites , a set of practical problem instances and a set of hard problem instances from the phase transition.

Keywords: graph colouring, representation, heuristics, merge model.

1 Introduction

The Graph Colouring Problem (GCP) plays an important role in graph theory. It arises in a number of applications—for example in time tabeling and scheduling, register allocation, and printed circuit board testing (see [1–3]). GCP deals with the assigment of colours to the vertices of an undirected graph such that adjacent vertices are not assigned the same colour. The primary objective is to minimize the number of colours used. The minimum number of colours necessary to colour the vertices of a graph is called the chromatic number. Finding it is an NP-hard problem, but deciding whether a graph is k-colourable or not is NP-complete [4]. Thus one often relies on heuristics to compute a solution or an approximation.

Graph colouring algorithms make use of adjacency checking during colouring, which plays a key role in the overall performance (see [5–7]). The number of checks depends on the problem representation and the algorithm that uses it. The Integer Merge Model (IMM) introduced here directly addresses the issues mentioned above. Generally, there are two main data structures used to represent graphs: the adjacency matrix and the adjacency list. In [6] a novel graph representation for the colouring problem called the Binary Merge Model (BMM) is introduced. IMM is a generalization of BMM, which is a useful and efficient representation of the GCP ([6, 7]). IMM preserves BMM's beneficial feature of improving upon efficiency. Moreover, it provides useful information about the graph structure during the colouring process, which enables one to define more sophisticated colouring algorithms and heuristics with a compact description. To demonstrate its potential, IMM is embedded in the DSATUR algorithm [8]—a standard and effective heuristic GCP solver—and in a meta-heuristic environment driven by an evolutionary meta-heuristic. On standard problem sets, we compare the effectiveness and efficiency of these three algorithms, with and without the use of IMM.

2 Representing the Graph k-Colouring Problem

The problem class known as the graph k-colouring problem is defined as follows. Given a graph $G(V, E)$ which is a structure of nodes and edges, where $V = \{v_1, ..., v_n\}$ 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 edges define the relation between the nodes ($V \times V \rightarrow E$). 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. The smallest such k is called the chromatic number, which will be denoted here by χ.

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 or coloured nodes must be scanned to check for equal colouring, so constraint checks need to be performed. The number of constraint checks performed lies between two bounds, the current number of coloured neighbours and $|V| - 1$. With the IMM approach the number of checks is greater than zero and less than the number of colours used up to this point. These bounds arise from the model-induced hyper-graph structure and they guarantee that the algorithms will perform better under the same search.

2.1 Integer Merge Model

The *Integer Merge Model* (IMM) implicitly uses hyper-nodes and hyper-edges (see Figure 1). A hyper-node is a set of nodes that have the same colour. A hyper-edge connects a hyper-node with other nodes, regardless of whether it is normal or hyper. A hyper-node and a normal-node or hyper-node are connected by a hyper-edge if and only if they are connected by at least two normal edges. IMM concentrates on the operations between hyper-nodes and normal nodes. We try to merge the normal nodes with another node, and when the latter is a hyper-node, a reduction in adjacency checks is possible. These checks can be performed along hyper-edges instead of normal edges, whereby we can introduce significant savings. This is because the initial set of normal edges is folded into hyper-edges. The colouring data is stored in an Integer Merge Table (IMT) (see Figure 2). Every cell (i, j) in this table has non-negative integer values. The columns refer to the nodes and the rows refer to the colours. A value in cell (i, j) is greater than zero if and only if node j cannot be assigned a colour i because of the edges in the original graph $\langle V, E \rangle$. The initial IMT is the adjacency matrix of the graph, hence a unique colour is assigned to each of the nodes. If the graph is not a complete graph, then it might be possible to reduce the number of necessary colours. This corresponds to the reduction of rows in the IMT. To reduce the rows we introduce an Integer Merge Operation, which attempts to merge two rows. When this is possible, the number of colours is decreased by one. When it is not, the number of colours remains the same. It is achievable only when two nodes are not connected by a normal edge or a hyper-edge. An example of both cases is found in Figures 1 and 2.

Definition 1. *The Integer Merge Operation* $\cup$ *merges an initial row* r_i *into an arbitrary (initial or merged) row* r_j *if and only if* $(j, i) = 0$ *(i.e., the hyper-node* x_j *is not connected to the node* x_i*) in the* IMT*. 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 IMT and R be the set of all possible $|V|$ size integer-valued rows (vectors). Then an integer merge operation is defined as

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

A merge can be associated with an assignment of a colour to a node, because two nodes are merged if they have the same colour. Hence, we need as many merge operation as the number of the nodes in a valid colouring of the graph, apart from the nodes which are coloured initially and then never merged, i.e., a colour is used only for one node. If k number of rows left in the IMT (i.e., the number of colours used) then the number of integer merge operations was $|V| - k$, where $k \in \{\chi, \ldots, |V|\}$.

With regard to the time complexity of a merge operation, we can say that it uses as many integer additions as the size of the operands. In fact, we just need to increment the value in the row r_j, where the corresponding element in the row r_i is non-zero (i.e., has a value of one), that is $d(x_i)$ number of operations. The number of all operations are at most $\sum_i d(x_i) = 2|E|$ for a valid colouring. This occurs when a list based representation of the rows is applied in an implementation. Using special hardware instructions available on modern computers, merge operations can be reduced to one computer instruction. For instance, a merge operation can be performed as one VDOT operation on a vector machine [9].

When solving a graph colouring while using the original graph representation for checking violations, approximately $|V|^2$ constraint checks are required to get to a valid colouring and the IMM supported scheme uses at most $|V| \cdot \kappa$ ($\kappa \approx \chi$ the number of colors in the result coloring) number of checks, because each node ($|V|$ items) has to be compared at most the number of existing hyper-nodes/colours, which is not more than κ and χ if a solution is found. Hence, their quotient is the improvement of an IMM supported colouring, which is proportional to the $|V|/\kappa$ ratio.

2.2 Permutation Integer Merge Model

The result of colouring a graph after two or more integer merge operations depends on the order in which these operations were performed. Consider the hexagon in Figure 1(a) and its corresponding IMT 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 integer merge operations, i.e., for the colouring.

This sequence of merge operations results in a 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 $3 \cup 4$. The defined merge is 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 IMT the *merged* IMT.

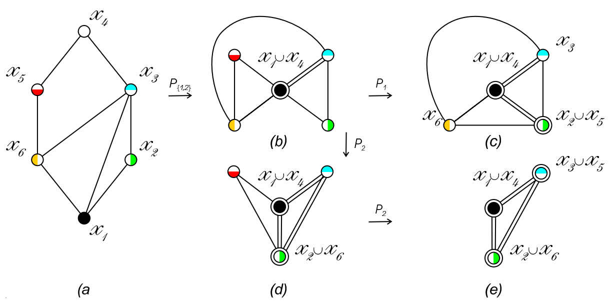


Fig. 1. Examples of the result of two different merge orders $P_1 = 1, 4, 2, 5, 3, 6$ and $P_2 = 1, 4, 2, 6, 3, 5$. The double-lined edges are hyper-edges and double-lined nodes are hyper-nodes. The P_1 order yields a 4-colouring (c), but with the P_2 order we get a 3-colouring (e).

Finding a minimal colouring for a graph k-colouring problem using the IMT representation and integer merge operations comes down to 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 Integer Merge Model (PIMM). It is easy to see that these operations and the colouring are equivalent.

2.3 Extracting Useful Information: Co-structures

The IMM can be incorporated into any colouring algorithm that relies on a construction based form of search. The hyper-graph structure introduced can save considerable computational effort as we have to make only one constraint check along a hyper-edge instead of checking all the edges it contains. Besides this favourable property, the model gives incremental insight into the graph structure

(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	2	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	2	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	2	0	1	1
$r_2 \cup r_6$	2	0	2	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	2	0	1	1
$r_2 \cup r_6$	2	0	2	0	1	0
$r_3 \cup r_5$	2	1	0	2	0	2

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

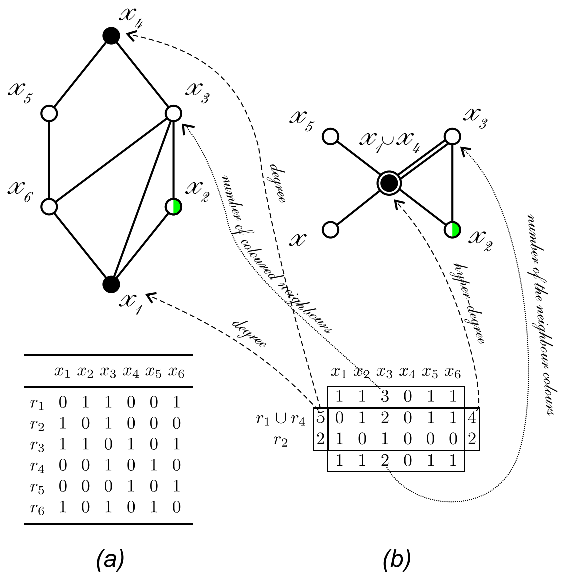
with the progress of the merging steps. This information can be used in a beneficial way, e.g., for defining colouring heuristics.

In this section, the co-structures are defined. These structures contain information about some useful graph properties obtained during the merging process. How this information is used precisely is explained in Sections 3 and 4, where we describe the two algorithms in which we have embedded the Integer Merge Model.

In practice the initial graphs are uncoloured, the colouring being performed by colouring the nodes in steps. Here, we deal with the sub-graphs of the original graphs defined by the colouring steps. The related merge tables contain partial information about the original one. For example, let the original graph with its initial IMT be defined by Figure 2.3(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 $P_1 = 1, 4, 2, 6, 3, 5$ ordered merges of the IMT rows will be performed. After the greedy colouring of the x_1, x_4, x_2 nodes there is a related partial or sub-IMT along with the (sub-)hyper-graph. These are depicted in Figure 2.3(b). The 1st and the 4th rows 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-IMT. The left, top, right and bottom bars are defined around the sub-IMT to store the four co-structures (see Figure 2.3(b)).

The left and top co-structures are associated with the original graph and contain the sum of the rows and the columns of the current IMT, respectively. The sum of the cell values of a row is equal to the sum of the degree of the nodes associated with the row (merged or initial), while the sum of the elements of the j-th column provides the coloured degree of the node x_j, i.e., the number of coloured neighbours.

The right and the bottom co-structures supply information about the hypergraph represented by the sub-IMT. They are calculated by counting the number of non-zero values in the rows and columns in the order described. The bottom bar value is the colour degree, i.e., the number of adjacent colours of a node.



	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

		x_1	x_2	x_3	x_4	x_5	x_6	
		1	1	3	0	1	1	
$r_1 \cup r_4$	5	0	1	2	0	1	1	4
r_2	2	1	0	1	0	0	0	2
		1	1	2	0	1	1	

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-IMT related to this colouring with its co-structures and sub-IMT induced hyper-graph.

The right bar gives the hyper-degree value of the nodes, which is especially interesting in case of hyper-nodes. The hyper-degree tells us how many different normal nodes are connected to the hyper-node being examined. This counts a node once even though it is connected to the hyper-node in question by more than one normal edge folded in a hyper-edge.

By extending the IMT we are able to describe efficient heuristics in a compact manner. To demonstrate this we will formulate two effective heuristic using the Integer Merge Model to get novel colouring algorithms. Two kinds of implementations of the two heuristic algorithms are considered during the experiments, when IMM is used and when it is not used.

3 The DSATUR Heuristic

This algorithm of Brélaz's [8] uses a heuristic to dynamically change the ordering of the nodes and it then applies the greedy method to colour them. It works as follows. One node with the highest saturation degree (i.e., number of adjacent colours) is selected from the uncoloured subgraph and is assigned the lowest indexed colour that still yields a valid colouring (first-order heuristic). If there exist several such nodes, the algorithm chooses a node with the highest degree (second-order heuristic). The result can also be a set of nodes. If this is the case, we choose the first node in a certain order (third-order 'heuristic'). The top and bottom co-structures are used to define the DSATUR heuristic (see Figure 4). Let us denote the top co-structure by τ_t (i.e., the number of coloured neighbours) and the bottom co-structure by τ_b (i.e., saturation degree). In our terminology the highest saturated node is the node which has the largest τ_b value. Here, τ_t is used in the second order heuristic.

```
Procedure DSATUR_IMM
1. Find those uncoloured nodes which have the highest saturated value
   $S = \{v | \tau_b(v) = \max_u(\tau_b(u)), v, u \in V\}$
2. Choose those nodes from S that have the highest uncoloured-degree
   $N = \arg\max_v(d(v) - \tau_t(v))$
3. Choose the first row/node from the set N
4. Merge it with the first non-neighbor hyper-node
5. If there exists an uncoloured node then continue with Step 2
```

Fig. 4. The DSATUR heuristic is defined by the IMM top (τ_t) and bottom (τ_b) co-structures. Here, d is the degree of a node.

A backtracking algorithm is used to discover a valid colouring [10]. It achieves either an optimal solution or a near optimal solution when the maximum number of constraint checks is reached. For comparison purposes, two algorithms were implemented using this heuristic. The first one, DSATUR$_{IMM}$ is based on the IMM structures, while the second one DSATUR$_{pure}$, uses the traditional colouring scheme, where we only make use of the adjacency matrix.

4 Evolutionary Algorithm to Guide the Models

We have two goals with this meta-heuristic. The first is to find a successful order of the nodes (see Section 2.2) and the second is to find a successful order for assigning colours. This approach differs from DSATUR, where a greedy color assignment is used. For the first goal, we must search the permutation search space of the model described in Section 2.2, 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., the rows of the IMT. The phenotype is a valid colouring of the graph after using a colour assignment strategy on the permutation to select the order of the integer merge operations. The colour assignement strategy is a generalization of the one introduced in [7]. We say that the c-th vector of the sub-IMT $r'(c)$ 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-IMT 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. Here, the dot product operates on integer vectors instead of binary ones, thus generalize that.

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 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 modify the fitness function introduced in [7] so that to use Integer Merge Model structures instead of the appropriate binary one. This fitness 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, commonly called a succeed-first strategy. It works as follows. After the final merge the resulting IMT 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 (see $g(p)$ above). But these nodes are not necessarily responsible for the over-coloured graph. Therefore, we choose to count the hyper-nodes that violates the least constraints in the final hyper-graph. To cope better with the fitness granularity problem we should modify the $g(p)$ according to the constraints of the over-coloured nodes discussed previously. The final fitness function is then defined as follows. Let $\zeta(p)$ denote the number of constraints, i.e., non-zero elements, in the rows of the final IMT that belong to the over-coloured nodes, i.e., the sum of the smallest $k(p) - \chi$ values of the right co-structure. The fitness function becomes $f(p) = g(p)\zeta(p)$. Here, the cardinality of the problem is known, and used as a stopping criterium ($f(p) = 0$) to determine the efficiency of the algorithm. If χ is unknown, we can use the worst approximation which is $\chi' = 0$. We must modify the stop condition to, reaching a time limit or to fitness ≤ 0 due to under-approximation ($\chi' \leq \chi$) or over-approximation ($\chi' > \chi$). Alternatively, the normalisation step can be left out, but this might seriously effect the quality of the evolutionary algorithm in a negative way.

```
Procedure EA$_{IMM}$
 1. population = generate initial permutations randomly
 2. while stop condition allows
      – evaluate each p permutation {
      – – merge $p_j$ – th uncoloured node into c – th hyper-node by $c = \max_j \left\langle r'_j, r_{p_i} \right\rangle$
      – – calculate $f(p) = (k(p) - \chi)\zeta(p)$   }
      – population$_{xover}$ = xover(population, prob$_{xover}$)
      – population$_{mut}$ = mutate(population$_{xover}$, prob$_{mut}$))
      – population = select$_{2-tour}$(population $\cup$ population$_{xover}$ $\cup$ population$_{mut}$)
 3. end while
```

Fig. 5. The EA$_{imm}$ meta-heuristic uses directly the IMM structure

We use a generational model with 2-tournament selection and replacement, where we employ elitism of size one. This setting is used in all experiments. The initial population is created with 100 random individuals. Two variation operators are used to provide offsprings. First, the 2-point order-based crossover (OX2) [11, 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. The probability of using OX2 is set to 0.3 and the probability for using the simple swap mutation is set to 0.8. These parameter settings are taken from the experiments in [7].

5 Experiments

The goal of these experiments are twofold. First, to show the improvement in efficiency possible when adding the Integer Merge Model to an existing technique. Second, to show further improvement possible in the evolutionary algorithm by adding heuristics that are based on the additional bookkeeping in the form of the co-structures.

5.1 Methods of Comparison

How well an algorithm works depends on its effectiveness and efficiency in solving a problem instance. The first is measured by determining the ratio of runs where the optimum is found, this ratio is called the success ratio; it is one if the optimum, i.e., the chromatic number of the graph, is found in all runs. The second is measured by counting the number of constraint checks an algorithm requires to find the optimum. A *constraint check* is defined equally for each algorithm as checking whether the colouring of two nodes is allowed or not. This measurement is independent of the hardware used and is known to grow exponentially with the problem size for the worst-case.

5.2 Definition of the Problem Suites

The first test suite consists of problem instances taken from “The Second DIMACS Challenge” [12] and Michael Trick’s graph colouring repository [12]. These graphs originate from real world problems, with some additional artificial ones.

The second test suite is generated using the well known graph k-colouring generator of Culberson [13]. It consists 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 nineteen 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 0.010 to 0.100 by steps of 0.005 over the groups. To characterize better the area of the phase transition, a simplification technique is used introduced by Cheeseman et al in [14]. This three steps node reduction removes the 0.010–0.020 groups, and simplify the graphs in the other groups to get the core of the problems.

5.3 Results

In this section, the results of the three kinds of algorithms are presented with and without using the Integer Merge Model, i.e., DSATUR, EA which uses the introduced fitness f and colour choosing heuristics and EA$_{noheur}$ which does not apply these heuristics, it uses a greedy colouring with the fitness g. Each algorithm was stopped when they reached an optimal solution or 150 000 000 number of constraint checks. DSatur with backtracking is an exact solver, it tries to explore the search space systematically by its heuristics. Thus, only one run is enough to get its result. Because of the stochastic nature of EAs, we use ten independent runs.

We can summarise the results on test suite one found in Table 1 as follows,

- The performance of an algorithm improves significantly if it employs the IMM framework.
- The evolutionary algorithms perform better than DSATUR, even after improving the efficiency of the latter with IMM.
- Adding heuristics to the evolutionary algorithms is useful to improve upon the efficiency for harder problem instances.
- All algorithms find a solution for almost every problem within the maximum number of constraint checks, except for the extremely hard queen8_8 and r75_5g_8 problems.

Table 1. Number of constraint checks required for test suite one using DSATUR and EA with and without IMM (latter is denoted by pure). Ten runs are averaged with different random seeds for EA-s. Prefix-indices show the success ratios if it is not one.

GRAPH	$\|V\|$	$\|E\|$	χ	DSATUR$_{imm}$	DSATUR$_{pure}$	EA$_{imm}$	EA$_{imm}^{noheur}$	EA$_{pure}$	EA$_{pure}^{noheur}$
fpsol2.i.2	451	8691	30	3059091	40527833	3414	4541	42022	56027
fpsol2.i.3	425	8688	30	2660498	32683629	3174	4988	39151	61015
homer	561	1629	13	2085103	75198957	2455	3672	57586	171641
inithx.i.1	864	18707	54	22305812	345876238	4328	5456	120348	142315
inithx.i.2	645	13979	31	6030391	95778467	2606	3680	84603	112000
inithx.i.3	621	13969	31	5762200	86482594	2480	3804	79458	124508
miles500	128	1170	20	147922	1046162	9066	46276	10366	75445
miles750	128	2113	31	204871	1121864	120051	693403	145459	5103811
miles1000	128	3216	42	244886	1249001	57934	559636	116054	1120068
miles1500	128	5198	73	329361	1500956	5436	14584	7032	19550
mulsol.i.5	186	3973	31	472872	2750261	1221	1370	7916	8905
myciel6	95	755	7	27807	624340	283	331	1499	2146
myciel7	191	2360	8	134956	4810974	901	1350	5602	11163
queen5_5	25	160	5	1665	12408	678	1777	906	2488
queen7_7	49	476	7	1176441	9106599	1092455	6675813	2793682	25332278
queen8_8	64	728	9	–	–	${}_{0.6}87482316$	${}_{0.4}102517235$	${}_{0.2}125298157$	–
r75_5g_8	75	1407	13	35693383	–	18668080	${}_{0.2}122257875$	${}_{0.9}29609833$	${}_{0.2}129031499$

Figure 6 shows the performance measured by success ratio and by average constraint checks performed of the algorithms on test suite two where 50 independent runs are used for every setting of the density. Both evolutionary algorithms show a sharp dip in the success ratio in the phase transition (see Figure 6), which is accompanied with a rise in the average number of constraint checks. IMM has significant influence on the performance, the improvement lies in between 6 and 48 times on average (see Figure 6). DSatur provides good results on the whole suite. Both the low target colour and the sparsity of the graphs are favourable terms for the heuristics it employs. Furthermore, the order of the graphs does not imply combinatorial difficulties for the backtracking algorithm. Beside these facts, the suite is appropriate to get valuable information about the behaviour of the algorithms. Even if the DSATURS perfom well on the problem sets, the EA, using the IMM abilities, can outperform the pure version of DSATUR in the critical region. In the phase transition it is 50% better on average. In practice, increasing the size of the graph leads to better performance of the EAs as opposed to the two exact DSATUR algorithms. By employing EA heuristics, i.e., the fitness function f and the colour choosing strategy, we clearly notice an improvement in both efficiency and effectiveness over the simple greedy colouring strategy with the simple fitness g. Furthermore, the confidence intervals for this range are small and non-overlapping. These two approaches give a much robust algorithm for solving graph k-colouring.

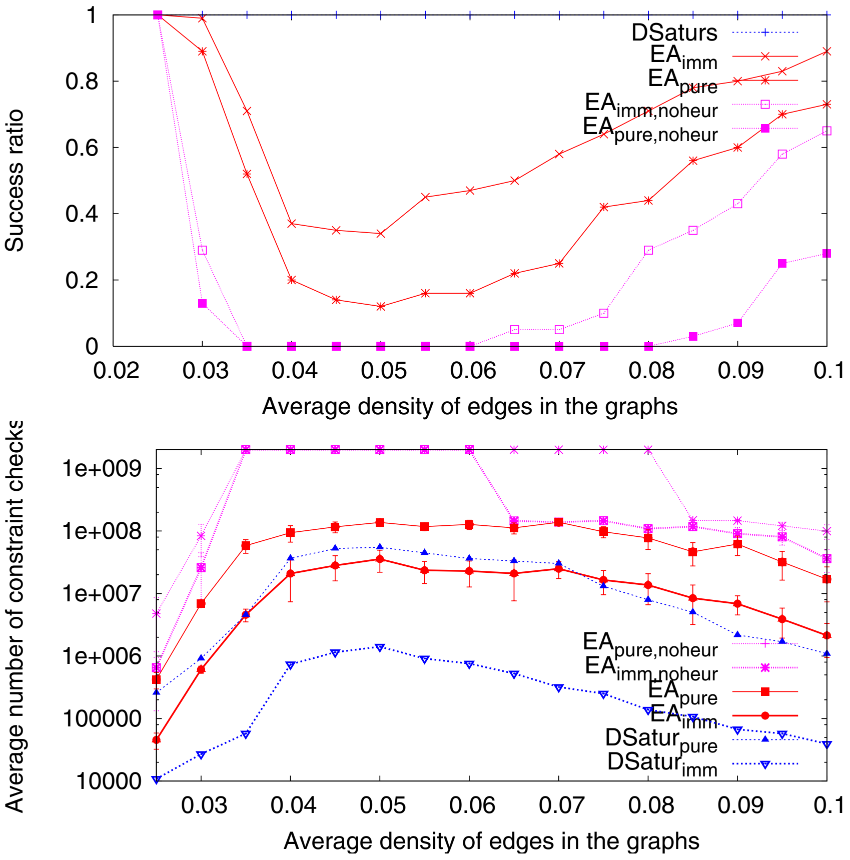


Fig. 6. Success ratio (DSATUR is always one) and average constraint checks to a solution for the DSATUR variants, the EAs with and without heuristics (with 95% confidence intervals)

6 Conclusions

In this paper, we introduced the Integer Merge Model for representing graph colouring problems. It forms a good basis for developing efficient graph colouring algorithms because of its three beneficial properties, a significant reduction in constraint checks, the availability of useful information for guiding heuristics, and the compact description possible for algorithms.

We showed how the popular DSATUR can be described in terms of the Integer Merge Model and we empirically investigated how much it can benefit from the reduction in constraint checks. Similarly, we showed how an evolutionary algorithm can be made more effective by adding heuristics that rely on the Integer Merge Model. Here we have shown a significant increase in both effectiveness and effectivity.

Further studies may include incorporating the Integer Merge Model in other algorithms, including more heuristics. Also, other constraint problems may be considered.

References

1. de Werra, D.: An introduction to timetabling. European Journal of Operations Research **19** (1985) 151–162
2. Briggs, P.: Register allocation via graph coloring. Technical Report TR92-183 (1998)
3. Garey, M., Johnson, D., So, H.: An application of graph colouring to printed circuit testing. IEEE Transaction on Circuits and Systems **CAS-23** (1976) 591–599
4. Garey, M., Johnson, D.: Computers and Instractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco, CA (1979)
5. Craenen, B., Eiben, A., van Hemert, J.: Comparing evolutionary algorithms on binary constraint satisfaction problems. IEEE Transactions on Evolutionary Computation **7** (2003) 424–444
6. Juhos, I., Tóth, A., van Hemert, J.: Binary merge model representation of the graph colouring problem. In Raidl, G., Gottlieb, J., eds.: Evolutionary Computation in Combinatorial Optimization. Volume 3004 of LNCS., Springer (2004) 124–134
7. Juhos, I., Tóth, A., van Hemert, J.: Heuristic colour assignment strategies for merge models in graph colouring. In: Evolutionary Computation in Combinatorial Optimization. Volume 3448 of LNCS., Springer (2005) 132–143
8. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM **22** (1979) 251–256
9. CNET: Xbox specs revealed (http://cnet.com/xbox+specs+revealed/2100-1043_3-5705372.html) (2005) Accessed: Nov. 10, 2005.
10. Golomb, S., Baumert, L.: Backtrack programming. ACM **12** (1965) 516–524
11. Bäck, T., Fogel, D., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. Institute of Physics Publishing Ltd, Oxford University Press (1997)
12. Johnson, D., Trick, M.: Cliques, Coloring, and Satisfiability. American Mathematical Society, DIMACS (1996)
13. Culberson, J.: Iterated greedy graph coloring and the difficulty landscape. Technical Report TR 92-07, University of Alberta, Dept. of Computing Science (1992)
14. Cheeseman, P., Kenefsky, B., Taylor, W.: Where the really hard problems are. In: Proceedings of the IJCAI'91. (1991) 331–337