Increasing the efficiency of graph colouring algorithms with a representation based on vector operations

István Juhos
Department of Computer Algorithms and Artificial Intelligence, University of Szeged, Hungary
E-mail: juhos@inf.u-szeged.hu

Jano I. van Hemert
National e-Science Institute, University of Edinburgh, United Kingdom
E-mail: 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 graph colouring algorithms. Moreover, this model provides useful information to aid in the creation of heuristics that can make the colouring process even faster. It also serves as a compact definition for the description of graph colouring algorithms. To verify the potential of the model, we use it in the complete algorithm DSATUR, and in two version of an incomplete approximation algorithm; an evolutionary algorithm and the same evolutionary algorithm extended with guiding heuristics. Both theoretical and empirical results are provided investigation is performed to show an increase in the efficiency of solving graph colouring problems. Two problem suites were used for the empirical evidence: a set of practical problem instances and a set of hard problem instances from the phase transition.**

Index Terms— **graph colouring, representation, node merging, colouring strategies, evolutionary algorithm, DSATUR**

I. Introduction

The Graph Colouring Problem (GCP) plays an important role in graph theory. It arises in a number of applications—for example in time tabling and scheduling, register allocation, and printed circuit board testing (see [1]–[3]). GCP deals with the assignment of colours to the vertices of an undirected graph such that adjacent vertices are not assigned the same colour. The primary objective is to minimise 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 for large problem instances.

Graph colouring algorithms make use of *adjacency checking*, during colouring, which plays a key role in the overall performance (see [5]–[7]). In the more general context of constraint satisfaction problems, one speaks about *constraint checks*, which is defined as the process of verifying whether a constraint is either satisfied or violated under the current partial assignment of values to variables. The number of these checks applied to solve a problem instance, depends on the problem representation and how the algorithm then uses this representation. The Integer Merge Model (IMM) introduced here directly addresses these issues. Generally, there are three main data structures used to represent graphs: the adjacency matrix, the incidence matrix, and the adjacency list. For graph colouring the adjacency matrix is commonly used, although a few specialised representations exist that depend on specific problem properties, for instance, on sparse graphs [8]. In [6], a novel graph representation for the colouring problem called the Binary Merge Model (BMM) is introduced. IMM is a generalisation of BMM, which is a useful and an efficient representation of the GCP (see [6], [7]). IMM preserves BMM's beneficial feature of improving upon efficiency, i.e., decreasing the number of adjacency checks. Moreover, it provides useful information about the graph structure during the colouring process, which enables one to define more sophisticated colouring algorithms and heuristics, and to describe these with a compact description.

To demonstrate the potential of IMM, it is embedded in the DSATUR algorithm [9]—a standard and effective complete GCP solver—and in a meta-heuristic environment driven by an evolutionary algorithm. Of the latter, two variants are used, one with and one without heuristics that make use of the additional information provided by IMM. On two problem sets, we compare the effectiveness and efficiency of these three algorithms, by testing them with and without using IMM.

II. Representing the Graph k-Colouring Problem

The problem class known as the graph k-colouring problem is defined as follows. Given a graph $G(V, E)$,

This paper is based on "Improving graph colouring algorithms and heuristics using a novel representation," by I.Juhos and J.I.van Hemert, which appeared in J.Gottlieb and G.Raidl, editors, Evolutionary Computation in Combinatorial Optimization, LNCS-3906, pages 123–134. © 2006, Springer-Verlag.

which is a structure of nodes and edges, where $V = \{x_1, ..., x_n\}$ is a set of nodes and $E = \{(x_i, x_j)|x_i \in V \wedge x_j \in V \wedge i \neq j\}$ is a set of edges. The edges define the relationship between the nodes ($V \times V \to E$). 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 for which this task can be achieved, 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. In the context of constraint satisfaction, one says that constraint checks are performed. For graph colouring, the number of constraint checks performed to test whether an assigned colour is valid, 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 explained next, and using these one can guarantee that the algorithms will perform better under the assumption it does the same search.

A. 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, which will be assigned the same colour as any pair of nodes in that set is never connected. Hyper-node can be generalized for normal nodes by considering a one element colour set, i.e., one colour..

Definition 1 (Hyper-node): Given two coloured nodes $\langle x_1, c_1\rangle$ and $\langle x_2, c_2\rangle$ we create a hyper-node $\{x_1, x_2\}$ iff $x_1 \neq x_2$ and $c_1 = c_2$. Generally, $H = \bigcup\{x_i\}_{i\in I}$, where $I \subseteq [1, n]$ is an index set having at least two elements, can form a hyper-node, when $\{x_i\}_{i\in I} \times \{x_i\}_{i\in I} \cap E = \emptyset$.

Here, a hyper-edge can connect only two hyper-nodes if and only if they are connected by at least two normal edges. We can generalize hyper-edges to normal edges as well, if we allow a hyper-edge to consist of one edge only.

Definition 2 (Hyper-edge): Let $I_1, I_2 \subseteq \{1, n\}$ distinct a non-empty index sets, i.e. $I_1 \cap I_2 = \emptyset$, furthermore $H_1 = \{x_i\}_{i\in I_1}$ and $H_2 = \{x_j\}_{j\in I_2}$ form hyper-nodes, that is there is no intra-edge between the nodes in a set: $H_1 \times H_1 \cap E = \emptyset$ and $H_2 \times H_2 \cap E = \emptyset$. H_1 and H_2 is connected by a hyper-edge (H_1, H_2) iff there is inter-edges between the nodes in the different sets, i.e. $H_1 \times H_2 \cap E > 1$.

When using graphs in this paper, we shall represent hyper-edges with two parallel lines to make them distinct from regular edges for which a single line is used. Similarly, for hyper-nodes we will add another slightly larger circle around the node, and annotate the node with a set of variables. Although, the colouring of nodes is implicit in our model, as it starts with assigning a unique colour to every node, then removing colours as nodes are merged into hyper-nodes, we will colour nodes to make the process easier to follow. Coloured nodes are shown by colouring half the node. If nodes are coloured with the same colour, they have the same half coloured.

The 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 $G(V, E)$. 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.

The *Integer Merge Operation* merges an initial row r_i into an arbitrary (initial or merged) row r_j if and only if $(j, i) = 0$ (i.e., none of the nodes in the hyper-node $\{x_{j_1}, \ldots, x_{j_m}\}$ are 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 these rows, which in the context of the graph $G(V, E)$ amounts to either creating a hyper-node $\{x_i, x_{j_1}, \ldots, x_{j_m}\}$ or merging two hyper-nodes $\{x_{i_1}, \ldots, x_{i_t}, x_{j_1}, \ldots, x_{j_m}\}$ with $2 \leq m, t \leq n$ and $m + t \leq n$.

Definition 3: The Integer Merge Operation Let S 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,

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

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 are left in the IMT, i.e., the number of colours used, then the number of Integer Merge Operations is $|V| - k$, where $k \in \{\chi, \ldots, |V|\}$.

B. Computational cost of the merge operation in practice

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 $|V|$. In fact, we need only to increment the value in the row r_j, where the corresponding element in the row r_i is non-zero, which is $d(x_i)$ number of operations. The number of all operations are less than $\sum_i d(x_i) = 2|E|$ for a valid colouring. This occurs when a list based representation of the rows is available in an implementation. Note that, incrementing an integer value may require less CPU time than an integer addition.

Since, nowadays computer processors (CPU) support parallel operations, e.g., vector addition operations (VADD), a merge operation can be only one instruction instead of $|V|$ or $d(x_i)$ instructions. In this case at most $|V| - k$ number of VADD operations are needed for a valid colouring (see above). The order of real life graphs can vary from a hundred nodes to thousands of nodes, but a common CPU can be able to perform less additions simultaneously. In other words, the space for performing VADD operations is smaller than the size of the graph. More explicitly, due to a cell value of an IMT is always less than n, thus we need at most $\lceil \log_2 n \rceil$ bits allocated for each one. Since, an IMT row has n cells, we need an array of bits of size $n \lceil \log_2 n \rceil$ for a VADD operation, which can be distributed over several registers, in a vector machine to get *a merge in one operation*.

For instance, the IBM PowerPC CPU used in an Xbox [10] has 49 152 ($3 \cdot 128 \cdot 128$) bits for this operation. Thus, we can use one merge operation for graphs having at most 4 000 number of nodes. Hence, one can require special hardware, a vector processor to achieve the appropriate VADD size required to keep the $|V| - k$ complexity in colouring. Nevertheless, having smaller VADD size, say l, the necessary VADD operations are $\lceil |V|/l \rceil \, (|V| - k)$, which can still significantly reduce the computational efforts for a merge. Especially, if $l \geq |V|$, then we get back the $|V| - k$ as mentioned previously. Examples that show how such hardware can speed up computation are found in the survey by [11], and more specifically for constraint satisfaction in [12].

C. Computational complexity of the constraint checks

When solving a graph colouring problem while using the original graph representation to check for violations, approximately $|V|^2$ constraint checks are required to get to a valid colouring. In contrast, the IMM supported scheme uses at most $|V| \cdot k$ number of checks ($|V| \geq k \geq^2 \chi$). This is possible because each node will be compared at most to the existing hyper-nodes/colours, of which is there are not more than k or χ if a solution exists. Hence, their quotient determines the improvement of an IMM supported colouring, which is proportional to the $|V|/k$ ratio. We verify this claim theoretically in this section and empirically in the experiments section.

In traditional schemes, adjacency matrix representation plays the key role in the GCP [1]. We have two choices when colouring a node for constraint checking; either along the already coloured nodes, or along all the neighbours of the node considered. In the following, we show how to considerably reduce the number of constraint checks by applying our proposed representation.

Let Π is the sequence of the nodes occur in the colouring process. Define $\overset{\curvearrowleft}{d}\,(x)$ as the coloured-degree of the node x being currently coloured, which refers backwards to the already coloured nodes and $\overset{\curvearrowright}{d}\,(x)$ of the uncoloured-degree refers forwards to the uncoloured node. Furthermore, denote $k_{\Pi(x)}$ the number of colours used before x would have been coloured according to Π.

Corollary 1: Given a random graph $G(n,p)$ with fixed p and given a colouring algorithm $\mathcal{A}$, then the following performance is expected on average based on counting constraint checks $\#(.)$:

1) checking the coloured nodes: $\#(\mathcal{A}_{col}) = O(n^2)$

2) checking the neighbours: $\#(\mathcal{A}_{neigh}) = O(n^2)$

3) checking the hyper-nodes/colour classes: $\#(\mathcal{A}_{imm}) \leq O(n^2/\log n)$

Proof:

1) Checking the already coloured nodes requires as many neighbour checks as the number of the edges, because we have to check the t number of coloured nodes if the $t+1$. node comes to colour, that is,

$$\#(\mathcal{A}_{col}) = \sum i = \frac{1}{2} n(n-1) = O(n^2) \qquad (1)$$

2) When the neighbours of the node currently being coloured are checked for constraint violation, the number of performed constraint checks are equal to the sum of the degrees, i.e., twice the number of edges

$$\#(\mathcal{A}_{neigh}) = \sum d_i = 2|E| \propto pn(n-1) = O(n^2) \qquad (2)$$

3) Using the IMM representation, merge operations provide hyper-nodes, which represent colour classes, thus checking along the hyper-nodes, requires at most as many checks as the number of colours used at that moment. The worst case is when the colouring is tight, meaning node x is in position $\Pi(x)$ coloured by at least the colour $k_\Pi(x)$.

$$\begin{aligned} \#(\mathcal{A}_{imm}) \leq \sum k_i = & \; n\tfrac{\sum k_i}{n} \propto nr\chi \propto n\tfrac{rn}{2\log n} && (3) \\ = & \; O(n^2/\log n) \\ n\frac{rn}{2\log n} \propto & \; \tfrac{rp}{2p}\tfrac{n(n-1)}{\log(n-1)} \propto \tfrac{pn(n-1)}{\log(n-1)^{2p/r}} && (4) \end{aligned}$$

[1]List based or incidence matrix representations requires more operations for graph colouring.

where r is constant[2] and $\chi \propto \frac{n}{2 \log n}$ according[3] to [16].

■

While, the theorem above tells us an asymptotic behaviour of the algorithms, we can check the worst case behaviour of the $\mathcal{A}$ using these different approaches. It is clear that, the application of the $\mathcal{A}_{col}$ for dense graphs are better against the $\#(\mathcal{A}_{neigh})$ and conversely $\mathcal{A}_{col}$ has worse properties in spare graphs against $\#(\mathcal{A}_{neigh})$. The following theorem state that using our IMM approach, the $\#(\mathcal{A}_{imm})$ always outperforms the other techniques mentioned.

Corollary 2: Let G an arbitrary graph, then the following relations are hold

1) $\#(\mathcal{A}_{imm}) \le \#(\mathcal{A}_{col})$
2) $\#(\mathcal{A}_{imm}) \le \#(\mathcal{A}_{neigh})$

Proof:

1) The number of colours are less than the number of coloured nodes:
$\#(\mathcal{A}_{imm}(x)) \le k_{\Pi(x)} \le \Pi(x)$, then
$\#(\mathcal{A}_{imm}) \le \sum k_i \le \sum i$.
2) When $\overset{\frown}{d}(x)$ refers to distinctly coloured nodes then $\#(\mathcal{A}_{imm}(x)) = \overset{\frown}{d}(x)$. Otherwise, if $\overset{\frown}{d}(x)$ refers to same coloured nodes as well as distinct ones the $\#(\mathcal{A}_{imm}(x)) = \overset{\frown}{d}(x)$, since hyper-nodes encompasses the same coloured nodes. Consequently

$\#(\mathcal{A}_{imm}(x)) \le d(x)$ due to $d(x) = \overset{\frown}{d}(x) + \overset{\frown}{d}(x)$, and $\#(\mathcal{A}_{imm}) \le \sum \overset{\frown}{d}(x) \le \sum d(x)$.

■

Conclusion of the Corollary 2 tells us more. Namely, an IMM based algorithm could perform better than that which could check the coloured neighbours only. However, to implement such an algorithm, which checks the coloured neighbours only, we have to use additional computation efforts, thus an IMM algorithm performs even better.

D. 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 $P_1 = r_1, r_4, r_2, r_5, r_3, r_6$ 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 $P_2 = r_1, r_4, r_2, r_6, r_3, r_5$ then the result will consist of a 3-colouring, as shown in Figure 1(e) with the merges *merge*(r_1, r_4), *merge*(r_2, r_6) and *merge*(r_3, r_4). The 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.

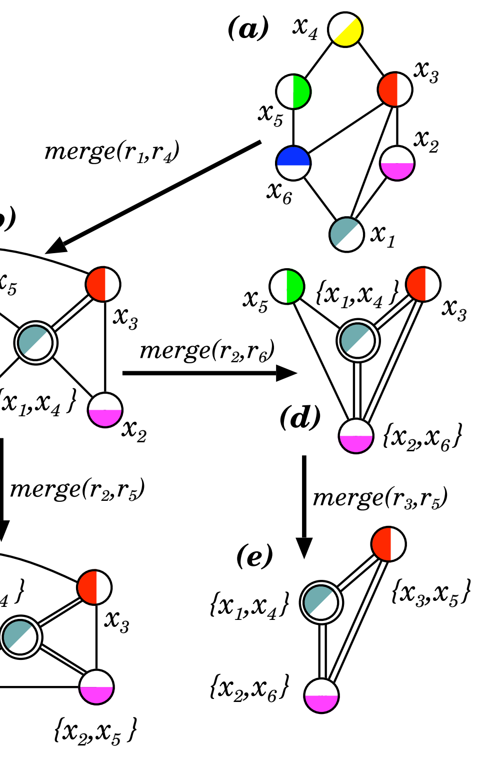


Figure 1. Examples of the result of two different merge orders $P_1 = r_1, r_4, r_2, r_5, r_3, r_6$ and $P_2 = r_1, r_4, r_2, r_6, r_3, r_5$. The double-lined edges are hyper-edges and double-lined nodes are hyper-nodes. The P_1 order yields a 4-colouring (c), however with the P_2 order we get a 3-colouring (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.

E. 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. Next to this favourable property, the model gives incremental insight into the graph structure with the progress of the merging steps. This information can be used in a beneficial way, for instance, 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 in the algorithm is explained

[2] This is true in the context of the naturally defined greedy algorithms $r \approx 2$ [13]–[15], other algorithms are designed to perform better.
[3] Note, the base of the logarithms is $1/(1-p)$.

(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

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

in Sections III-A and III-B, where we describe the two algorithms in which we have embedded the Integer Merge Model.

In practice, in the initial graphs none of the nodes are coloured. The colouring is then 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 3 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 $P_1 = r_1, r_4, r_2, r_6, r_3, r_5$ 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 3. The 1st and the 4th rows are merged together, but the 2nd cannot be merged with the result of *merge*(r_1, r_4), thus the 2nd row remains unaltered in the related IMT.

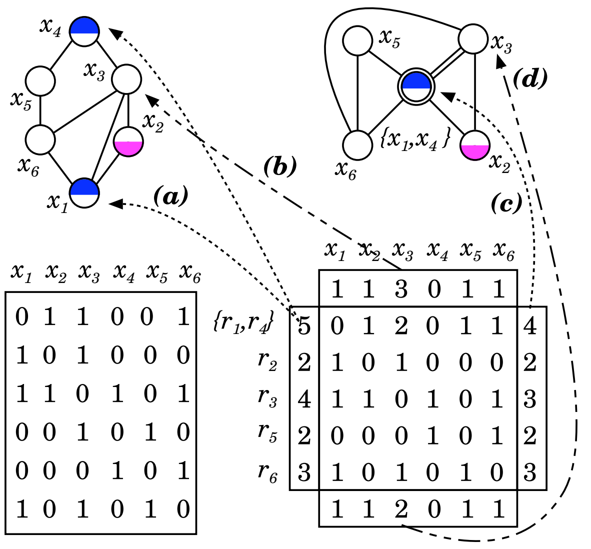
The left, top, right, and bottom bars are defined around an IMT to store the following four co-structures.

a) The left co-structure: is associated with the original graph and each row contains the sum of the values of that row. 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).

b) The top co-structure: contains for each row the sum of that row, which gives us the number of coloured neighbours in the original graph, i.e., the graph without taking merging into account.

c) The right co-structure: supplies information about the hyper-graph represented by the sub-IMT. Its values are calculated by counting the number of non-zero values in the rows and columns in the order described. It provides the hyper-degree value of the nodes, which is especially interesting in case of hyper-nodes. The hyper-degree tells us how many unique normal nodes are connected to the hyper-node being examined. This counts a node once, even if it is connected to the hyper-node in question by more than one normal edge folded into a hyper-edge.

d) The bottom co-structure: also supplies information about the hyper-graph represented by the sub-IMT. It counts the number of non-zero values in each column in the order described. For every node, we then derive 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	
$\{r_1,r_4\}$	5	0	1	2	0	1	1	4
r_2	2	1	0	1	0	0	0	2
r_3	4	1	1	0	1	0	1	3
r_5	2	0	0	0	1	0	1	2
r_6	3	1	0	1	0	1	0	3
		1	1	2	0	1	1	

Figure 3. The left side shows the partial colouring of the G graph according to the x_1, x_4, x_2 greedy order and a common adjacency matrix of the graph. The right side shows the IMT related to this partial colouring with its co-structures and IMT induced hyper-graph.

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.

Procedure DSATUR*imm*

1) *Find those uncoloured nodes which have the highest saturated value:*
   $S = \{x | \tau_b(x) = \max_y(\tau_b(y)), x, y \in V\}$
2) *Choose those nodes from* S *that have the highest uncoloured-degree:*
   $N = \arg\max_x(d(x) - \tau_t(x))$
3) *Choose the first row/node from the set* N
4) *Merge it with the first non-neighbour hyper-node*
5) *If there exists an uncoloured node then continue with Step 2*

Figure 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.

III. Embedding the integer merge model

We shall now describe how our proposed representation can be embedded into two different algorithms. The first, DSATUR, uses a backtracking algorithm that provides a complete algorithm, where the efficiency is mainly dependent on the ordering of colouring the variables, which is determined using a heuristic. The second, a simplistic evolutionary algorithm that uses a permutation as its genotype, and consequently applies suitable genetic operators on this genotype to maintain valid permutations.

A. DSATUR heuristic

This algorithm of Brélaz's [9] 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., the number of adjacent colours, is selected from the uncoloured sub-graph and is assigned the lowest indexed colour that still yields a valid colouring, which is the first-order heuristic. If there exist several such nodes, the algorithm chooses a node with the highest degree, which forms the 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, which is the 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.

A backtracking algorithm is used to discover a valid colouring [17]. 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.

B. Evolutionary algorithm

We have two goals with this meta-heuristic. The first is to find a successful order of the nodes (see Section II-D) and the second is to find a successful order for assigning colours. This approach differs from DSATUR, where a greedy colour assignment is used. For the first goal, we must search the permutation search space of the model described in Section II-D, which is of size $n!$. Here, we use an evolutionary algorithm [18] 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 assignment strategy is a generalisation 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 generalise 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 fitness 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

Procedure EA*IMM*

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

Figure 5. The EA$_{imm}$ uses directly the IMM structure.

known, and used as one of the stopping criteria ($f(p) = 0$) 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 search in a negative way.

We use a generational model with 2-tournament selection and replacement, where we employ elitism of size one [19]. The initial population is created with 100 random individuals. Two variation operators are used to provide offspring. First, the 2-point order-based crossover (OX2) [19, in Section C3.3.3.1] is applied. Second, simple swap mutation operator is applied, 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. The stop condition is either a colouring with the chromatic number is found, or the maximum number of constraint checks, set in the experiments section, is reached. All these parameter settings are taken from the experiments in [7].

IV. 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.

A. 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 in the worst case.

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 evolutionary algorithms, we use ten independent runs.

TABLE I.
PROPERTIES OF THE GRAPHS IN THE DIMACS SUITE, SHOWING THE NUMBER OF VERTICES $|V|$, THE NUMBER OF EDGES $|E|$, AND THE CHROMATIC NUMBER OF THE GRAPH χ

GRAPH	$\|V\|$	$\|E\|$	χ
fpsol2.i.2	451	8691	30
fpsol2.i.3	425	8688	30
homer	561	1629	13
inithx.i.1	864	18707	54
inithx.i.2	645	13979	31
inithx.i.3	621	13969	31
miles500	128	1170	20
miles750	128	2113	31
miles1000	128	3216	42
miles1500	128	5198	73
mulsol.i.5	186	3973	31
myciel6	95	755	7
myciel7	191	2360	8
queen5_5	25	160	5
queen7_7	49	476	7
queen8_8	64	728	9
r75_5g_8	75	1407	13

B. DIMACS Challenge problem instances

The first test suite consists of problem instances taken from “The Second DIMACS Challenge” [20] and Michael Trick’s graph colouring repository [20]. Most of these graphs originate from real world problems, with some additional artificial ones. We can find problems from scheduling, register allocation, football games, city distances, placement of queens on a chessboard, connections of letters in a book. We present the basic properties of the graphs in Table I. Especially, the artificial problems are hard to solve due to the method of generating by Mycielski transformation [21]. They are triangle free, and the chromatic number increases with the problem size.

In Table II, we compare the results of two variants of DSATUR and two variants of the evolutionary algorithm without using heuristics. First, we observe that

TABLE II.
NUMBER OF CONSTRAINT CHECKS REQUIRED TO SOLVE TO OPTIMALITY FOR THE DIMACS TEST SUITE FOR DSATUR AND THE EA *without heuristics*. RESULTS ARE SHOWN WITH AND WITHOUT THE USE OF IMM (LATTER IS DENOTED BY PURE). PREFIX-INDICES SHOW THE SUCCESS RATIO IF IT IS NOT ONE.

GRAPH	DSATUR$_{imm}$	DSATUR$_{pure}$	EA$^{noheur}_{imm}$	EA$^{noheur}_{pure}$
fpsol2.i.2	3059091	40527833	**4541**	56027
fpsol2.i.3	2660498	32683629	**4988**	61015
homer	2085103	75198957	**3672**	171641
inithx.i.1	22305812	345876238	**5456**	142315
inithx.i.2	6030391	95778467	**3680**	112000
inithx.i.3	5762200	86482594	**3804**	124508
miles500	147922	1046162	**46276**	75445
miles750	**204871**	1121864	693403	5103811
miles1000	**244886**	1249001	559636	1120068
miles1500	329361	1500956	**14584**	19550
mulsol.i.5	472872	2750261	**1370**	8905
myciel6	27807	624340	**331**	2146
myciel7	134956	4810974	**1350**	11163
queen5_5	**1665**	12408	1777	2488
queen7_7	**1176441**	9106599	6675813	25332278
queen8_8	–	–	$_{0.4}$**102517235**	–
r75_5g_8	35693383	–	$_{0.2}$**122257875**	$_{0.2}$129031499

TABLE III.
NUMBER OF CONSTRAINT CHECKS REQUIRED TO SOLVE TO OPTIMALITY FOR THE DIMACS TEST SUITE FOR DSATUR AND THE EA *with heuristics*. RESULTS ARE SHOWN WITH AND WITHOUT THE USE OF IMM (LATTER IS DENOTED BY PURE). PREFIX-INDICES SHOW THE SUCCESS RATIO IF IT IS NOT ONE.

GRAPH	DSATUR$_{imm}$	DSATUR$_{pure}$	EA$^{heur}_{imm}$	EA$^{heur}_{pure}$
fpsol2.i.2	3059091	40527833	**3414**	42022
fpsol2.i.3	2660498	32683629	**3174**	39151
homer	2085103	75198957	**2455**	57586
inithx.i.1	22305812	345876238	**4328**	120348
inithx.i.2	6030391	95778467	**2606**	84603
inithx.i.3	5762200	86482594	**2480**	79458
miles500	147922	1046162	**9066**	10366
miles750	204871	1121864	**120051**	145459
miles1000	244886	1249001	**57934**	116054
miles1500	329361	1500956	**5436**	7032
mulsol.i.5	472872	2750261	**1221**	7916
myciel6	27807	624340	**283**	1499
myciel7	134956	4810974	**901**	5602
queen5_5	1665	12408	**678**	906
queen7_7	1176441	9106599	**1092455**	2793682
queen8_8	–	–	$_{0.6}$**87482316**	$_{0.2}$125298157
r75_5g_8	35693383	–	**18668080**	$_{0.9}$29609833

in all cases, using the IMM representation improves the efficiency algorithm with varying degrees of speed-ups, on which we elaborate more later. The evolutionary algorithm without heuristics performs much better than DSATUR for most problem instances solving some problem instances in a fraction of the time, such as inithx.1. Exceptions are three graphs that are hard to solve for DSATUR (miles500, miles1000, and queen7_7), and one that is easy to solve (queen5_5).

When the evolutionary algorithm is making use of the co-structures through its heuristics, we get the results shown in Table II. First, when using heuristics, the evolutionary algorithm always performs better than when not using heuristics. Also, the success ratio for the two hardest problem instances (queen8_8 and r75_5g.8) has improved and r75_5g.8 is now always solved to optimality. Second, using heuristics, the evolutionary algorithm always performs better than DSATUR.

We can summarise the results on test suite one found in Table II and Table III 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.

In Figure 6, we show how much the speed of DSATUR and the evolutionary algorithm increases measured as the ratio of constraint checks used to solve the problem when not using IMM and when using IMM. For DSATUR, the lowest speed increase is 4.56, while the largest speed-up is 36.1. For the evolutionary algorithm, the lowest speed increase is 1.81, and the largest speed-up is 41.4. Depicted in Figure 7, is the correlation of the speed-up ratios of the two algorithms with the ratio $|V|/\chi$, i.e., the number of nodes divided by the chromatic number. DSATUR fits with a coefficient of 0.948 and an asymptotic error of 10.0%, while the evolutionary algorithm fits with a coefficient of 0.695, and an asymptotic error of 9.8%. We had predicted this speed-up in Section II-C, and the correlation fits rather well, especially for DSATUR.

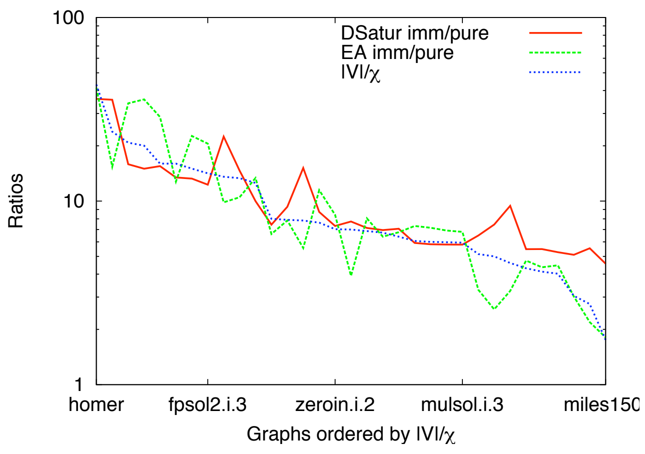


Figure 6. Speed increase ratios for DSATUR and the evolutionary algorithm for the DIMACS problems, which are ordered by $|V|/\chi$

C. Equi-partite graphs in the phase transition

The second test suite is generated using the well known graph k-colouring generator of Culberson [15]. 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 equi-partite,

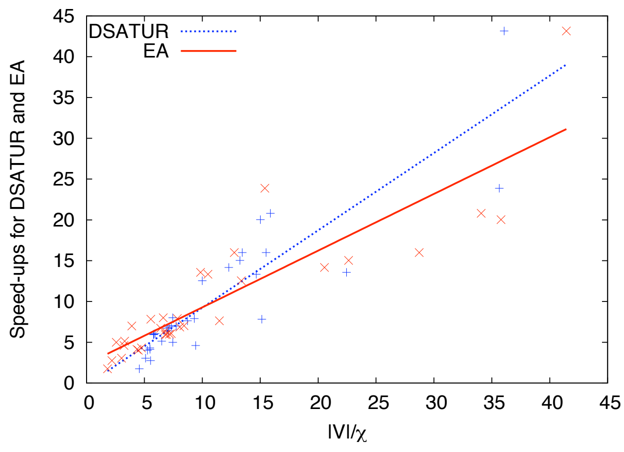
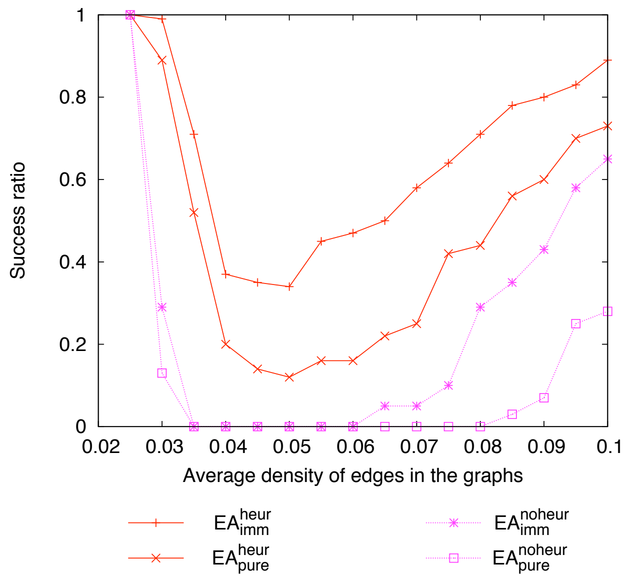


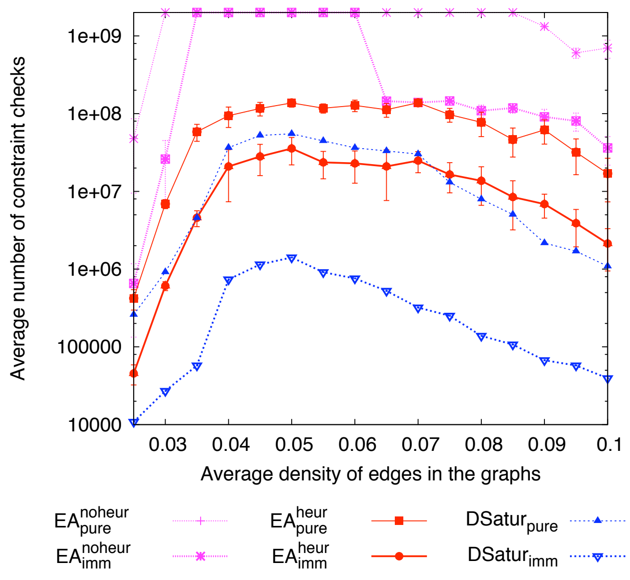
Figure 7. Linear correlation between the $|V|/\chi$ ratio and the speed-up ratios of DSATUR and the evolutionary algorithm with heuristics.

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 [22]. 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.

Figure 8 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 8), 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 8). DSATUR provides good results on the whole suite. Both the low target colour and the sparseness 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 perform 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.



(a) Success ratio (DSATUR is always one)



(b) Average constraint checks to a solution (95% confidence intervals included)

Figure 8. Results for the equi-partite graphs in the phase transition area for the DSATUR variants, the EAs with and without heuristics

V. 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, availability of useful information for designing heuristics that guide colouring, and it allows for a compact description of 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 efficient by adding heuristics that rely on the Integer Merge Model. Here we have shown a significant increase in both effectiveness, i.e., a solution is found more often, and efficiency, i.e., a solution is found faster. We show a speed-up of about $|V|/\chi$, i.e., the number of nodes in the graph divided by the chromatic number of the graph, is what can be expected based on our theoretical and empirical results.

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

Acknowledgment

During the cause of this work, the second author was funded through the ADONET Marie-Curie Training Network action, while working at the Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria.

References

[1] D. de Werra, "An introduction to timetabling," *European Journal of Operations Research*, vol. 19, pp. 151–162, 1985.

[2] P. Briggs, "Register allocation via graph coloring," Rice University, Houston, TX, USA, Tech. Rep. TR92-183, 24, 1998. [Online]. Available: citeseer.ist.psu.edu/briggs92register.html

[3] M. Garey, D. Johnson, and H. So, "An application of graph colouring to printed circuit testing," *IEEE Transaction on Circuits and Systems*, vol. CAS-23, pp. 591–599, 1976.

[4] M. Garey and D. Johnson, *Computers and Instractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: W.H. Freeman, 1979.

[5] B. Craenen, A. Eiben, and J. van Hemert, "Comparing evolutionary algorithms on binary constraint satisfaction problems," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 424–444, 2003.

[6] I. Juhos, A. Tóth, and J. van Hemert, "Binary merge model representation of the graph colouring problem," in *Evolutionary Computation in Combinatorial Optimization*, ser. LNCS, G. Raidl and J. Gottlieb, Eds., vol. 3004. Springer, 2004, pp. 124–134.

[7] ——, "Heuristic colour assignment strategies for merge models in graph colouring," in *Evolutionary Computation in Combinatorial Optimization*, ser. LNCS, vol. 3448. Springer, 2005, pp. 132–143.

[8] P. Briggs and L. Torczon, "An efficient representation for sparse sets," *ACM Lett. Program. Lang. Syst.*, vol. 2, no. 1-4, pp. 59–69, 1993.

[9] D. Brélaz, "New methods to color the vertices of a graph," *Communications of the* ACM, vol. 22, no. 4, pp. 251–256, Apr. 1979.

[10] CNET, "Xbox specs revealed (http://cnet.com/Xbox+specs+revealed/2100-1043_3-5705372.html)," 05 2005, accessed: Nov. 10, 2005. [Online]. Available: http://cnet.com/Xbox+specs+revealed/2100-1043_3-5705372.html

[11] J. Comba, C. Dietrich, C. Pagot, and C. Scheidegger, "Computation on gpus: from a programmable pipeline to an efficient stream processor," *Revista de Informatica Teorica e Aplicada*, vol. 10, no. 1, pp. 41–70, 2003. [Online]. Available: http://www.sci.utah.edu/~cscheid/pubs/rita_survey.pdf

[12] C. J. Thompson, S. Hahn, and M. Oskin, "Using modern graphics architectures for general-purpose computing: a framework and analysis," in *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 306–317.

[13] G. R. Grimmet and C. J. H. McDiarmid, "On colouring random graphs," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 77, pp. 313–324, 1975.

[14] C. McDiarmid, "Determining the chromatic number of a graph," *SIAM Journal on Computing*, vol. 8, no. 1, pp. 1–14, 1979.

[15] J. Culberson, "Iterated greedy graph coloring and the difficulty landscape," University of Alberta, Dept. of Computing Science, Tech. Rep. TR 92-07, 1992.

[16] B. Bollobás, "The chromatic number of random graphs," *Combinatorica*, vol. 8, no. 1, pp. 49–55, 1988.

[17] S. Golomb and L. Baumert, "Backtrack programming," *ACM*, vol. 12, no. 4, pp. 516–524, Oct. 1965.

[18] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[19] T. Bäck, D. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*. Institute of Physics Publishing Ltd, Oxford University Press, 1997.

[20] D. Johnson and M. Trick, *Cliques, Coloring, and Satisfiability*. American Mathematical Society, DIMACS, 1996.

[21] J. Mycielski, "Sur le coloriage des graphes (for colouring graphs)," *Colloquim Mathematiques*, vol. 3, no. 161, 1955.

[22] P. Cheeseman, B. Kenefsky, and W. Taylor, "Where the really hard problems are," in *Proceedings of the* IJCAI*'91*, 1991, pp. 331–337.

István Juhos is currently a Ph.D. candidate at the University of Szeged, Hungary. He received his MSc and BSc degrees in mathematics and computer science from University of Szeged, respectively.

He was a leader and co-leader of several industrial research and development project in the field of applied mathematics and artificial intelligence. His research interests include applied mathematics, graph theory and artificial intelligence. Within the latter he specialises in machine learning, data mining, and robotics.

Jano I. van Hemert received his MSc degree in computer science from Leiden University, The Netherlands, in 1998, and his PhD degree in mathematics and physical sciences from Leiden University, The Netherlands, in 2002.

He is currently a Researcher at the National e-Science Institute, University of Edinburgh and a Visiting Researcher at the Medical Research Council's Human Genetics Unit both in Edinburgh, United Kingdom. He has held research positions at the Leiden University in The Netherlands, the Vienna University of Technology in Austria, the University of Edinburgh in the United Kingdom, Napier University in Edinburgh, United Kingdom, and the National Research Institute for Mathematics and Computer Science (CWI) in Amsterdam, The Netherlands. Some of his relevant publications are "Evolving combinatorial problem instances that are difficult to solve" by J.I. van Hemert in Evolutionary Computation, in press, 2006; "Robust parameter settings for variation operators by measuring the resampling ratio: A study on binary constraint satisfaction problems" by J.I. van Hemert and Th.Bäck in Journal of Heuristics, 10(6):629–640, 2004. His research interests include evolutionary computation in conjunction with constraint optimisation, and e-Science applied in the field of biomedical applications, specifically in developmental biology.

Dr. van Hemert was awarded the prestigious talented young researcher fellowship by the Netherlands Organization for Scientific Research in 2004.