11

Contraction-Based Heuristics to Improve the Efficiency of Algorithms Solving the Graph Colouring Problem

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

[1] Department of Computer Algorithms and Artificial Intelligence, University of Szeged, Hungary
`paper@juhos.info`

[2] National e-Science Institute, School of Informatics, University of Edinburgh, United Kingdom
`j.vanhemert@ed.ac.uk`

Summary. Graph vertex colouring can be defined in such a way where colour assignments are substituted by vertex contractions. We present various hypergraph representations for the graph colouring problem all based on the approach where vertices are merged into groups. In this paper, we explain this approach and identify three reasons that make it useful. First, generally, this approach provides a potential decrease in computational complexity. Second, it provides a uniform and compact way in which algorithms, be it of a complete or a heuristic nature, can be defined and progress toward a colouring. Last, it opens the way to novel applications that extract information useful to guide algorithms during their search. These approaches can be useful in the design of an algorithm.

Keywords: Graph Contraction, Graph Colouring, Graph Representation, Heuristics, Graph Homomorphisms, Evolutionary Computation.

11.1 Introduction

The *graph colouring problem* (GCP) is an important problem from the class of non-deterministic polynomial problems. It has many applications in the real world such as scheduling, register allocation in compilers, frequency assignment and pattern matching. To allow these applications to handle larger problems, it is important fast algorithms are developed. Especially as high-performance computing is becoming more readily available, it is worthwhile to develop algorithms that can make use of parallelism.

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges $E \subseteq V \times V$ defines a relation over the set of vertices. We let $n = |V|$, $m = |E|$, $d(v)$ is the degree of a vertex $v \in V$ and A is the adjacency matrix of G. A colouring c of a graph is a map of colours to vertices ($c : V \to C$) where C is the set of colours used. There can be several such mappings, which can be denoted if necessary (e.g. $c_1, c_2, \dots$). A colouring c is a k-colouring iff $|C| = k$. It is a proper or valid colouring if for every $(v, w) \in E : c(v) \neq c(w)$. In general, when we refer to a colouring we mean a valid colouring, unless the context indicates otherwise. The chromatic number $\chi(G)$ of a graph G is the smallest k for which there exists a k-colouring of G. A colouring algorithm makes colouring steps, i.e., it progressively chooses an uncoloured vertex and then assigns it a colour. Let $t \in \{1, \ldots, n\}$ be the number of steps made.

The graph colouring approaches discussed here will construct a colouring for a graph by progressively contracting the graph. This condensed graph is then coloured. The colouring of intermediate graphs in examples serves only to help the reader follow the process. In reality, contractions themselves define the whole colouring process.

The operations that allow these contractions are naturally parallel. Moreover, they allow us to extract heuristic information to better guide the search. Two types of graph contractions exist in the literature. The first is the vertex-contraction; when unconnected or connected vertices are identified as one vertex, some original edges can either be kept or removed. The name 'contraction' can be seen as a merge or a form of coalescing ([225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235]) as well. The latter is commonly used in the domain of register allocation problems, which can be modeled as a graph colouring problem. In the colouring model of the register allocation, vertices are coalesced where this is safe, in order to eliminate move operations between distinct variables (registers). The second is the edge-contraction. This approach is similar to the idea described in case of vertices whenever two edges are merged together, vertices can be merged.

The purpose of the merging can either be simplification or the combination of several simple graphs into one larger graph. Both edge and vertex contraction techniques are valuable in proof by induction on the number of vertices of edges in a graph, where we can assume that a property holds for all contractions of a graph and use this to show it for the larger graph.

Usually, colouring algorithms use vertex merging for graph simplification and graph combination. Simplification is done by merging two or more unconnected vertices to get fewer vertices before or during colouring. In [225], [226] and [227] preprocessing of graphs is performed before colouring, where two vertices in a graph are merged to one if they are of the same colour in all colourings. This is analogous to studies of the development of a backbone or spine in the satisfiability problem [236, 237]. Here, the application of merging refers to removing one of two unconnected vertices. In fact, we could remove also edges that belong to the removed vertex. The only reason to perform these merges is to remove unnecessary or unimportant vertices from the graph in order to make it simpler. Those vertices that fulfil some specific condition, will be removed from the data structure, which describes the graph. This process will result in the loss of information.

The second approach is to consider two graphs, which have some colouring properties. For example the property could be that they are not k-colourable. Then, the two graphs are joined by merging vertices from both graphs to create a more complex graph, where the aim is that the original properties are inherited. In both cases the identified vertices get the same colour.

Register allocation can be modelled as a graph colouring problem, but it is modelled in other ways too. Coalescing is a terminology frequently used instead of merging, where two register are coalesced. If the problem is represented by graph colouring, coalescing is the same as merging unconnected vertices. Register coalescing techniques can be identified as vertex merging in a graph colouring problem [228, 229, 230, 231, 232].

Using the idea of graph contraction we will describe an efficient graph colouring model, where contractions are performed on unconnected vertices. We will define two well known and two novel heuristics based on this model. The model itself serves as a framework to describe graph colouring algorithms in a concise manner. Moreover, we show how the model allows an increase in the performance of certain algorithms by exploiting the data structures of the model through several heuristics.

11.2 Representing Solutions to Graph Colouring as Homomorphisms

We define merge operations to perform contraction of the original graph and subsequent contractions. A merge operation takes two unconnected vertices from a graph $G = (V, E)$ and produces a new graph $G' = (V', E')$ where these vertices become one hyper-vertex. If edges exist between another vertex and both the original vertices, then these become one hyper-edge. If $v_1, v_2 \in V$ are merged to $\{v_1, v_2\} \in V'$ and both $(v_1, u), (v_2, u) \in E$ then $(\{v_1, v_2\}, u) \in E'$ is called a hyper-edge. Examples of merge operations are shown in Figure 11.1. The merge operation is applied similarly to hyper-vertices.

By repeating merge operations we will end up with a complete graph. If during each merge operation we ensure only hyper-vertices and vertices are merged that have no edges between them, we then can assign all the vertices from the original graph that are merged into the same hyper-vertex one unique colour. The number of hyper-vertices in the final contracted graph corresponds to the number of colours in a valid colouring of the original graph. As Figure 11.1 shows, the order in which one attempts to merge vertices will determine the final colouring. Different colourings may use different number of colours. We will investigate a number of different strategies for making choices about which vertices to merge.

Graph colouring solvers make use of constraint checks during colouring. The adjacency checks to verify if assigned colourings are valid play a key role in the overall performance (see [238, 233, 234, 235]). The number of checks depends on the representation of the solution, the algorithm using the representation and the intrinsic difficulty of the problem. Graph colouring problem are known to exhibit an increase in difficulty in the so-called phase transition area [227]. In our experiments we will test several algorithms on problems in this area.

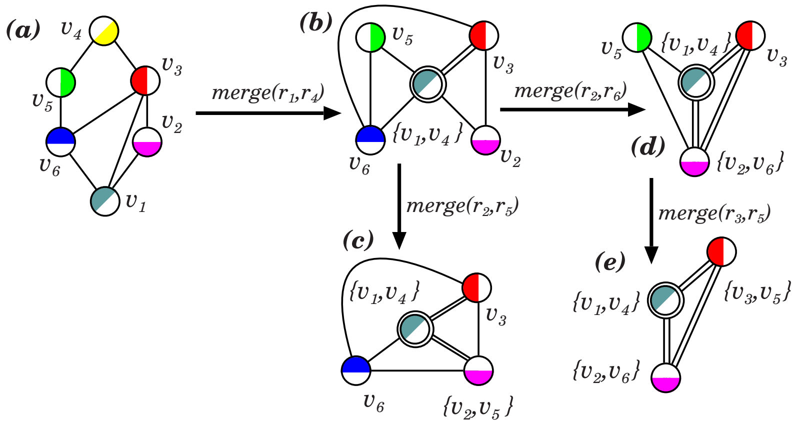


Fig. 11.1. Examples of the result of two different merge orders $P_1 = v_1, v_4, v_2, v_5, v_3, v_6$ and $P_2 = v_1, v_4, v_2, v_6, v_3, v_5$. The double-lined edges are hyper-edges and double-lined nodes are hyper-nodes. The P_1 order yields a 4-colouring (c), however with the P_2 order we get a 3-colouring (e).

Merge operations and the contracted graphs can reduce the number of constraint checks considerably [239]. There are two main possibilities to check if a colour can be assigned to a vertex. Either one examines the already coloured vertices for adjacency or one checks the neighbours of the vertex to validate the same colouring is not assigned to any of these. Using merge operations, we will have colour groups instead of sets of coloured vertices, thereby reducing the amount of checks required. In [235] a speed-up of about $\log n$ is derived both empirically and theoretically.

11.3 Vertex Colouring by Graph Homomorphisms

In this section we describe a valid vertex colouring of a graph G via several homomorphisms. Let ϕ be a homomorphism, we define ϕ in the following ways,

1. $\phi : V \to C$ defines an assignment from vertices to colours. It provides a valid colouring if $\phi(u) = \phi(v) \Rightarrow (u, v) \notin E$. Let the number of assigned colours be k, it defines the quality of the assignment when minimising the number of colours used (See Figure 11.2). We can represent the coloured graph by a composite structure $\langle G, C\rangle$. A homomorphism can be defined by atomic steps, where at each step one vertex is coloured. These steps will be defined in Section 11.3.1.
2. The second homomorphism groups vertices of the same colour together to form hyper-vertices, i.e., colour sets. The edges defined between normal and hyper-vertices (colour sets) provide an implicit representation of the colouring by the special graph formed this way. An example is shown in

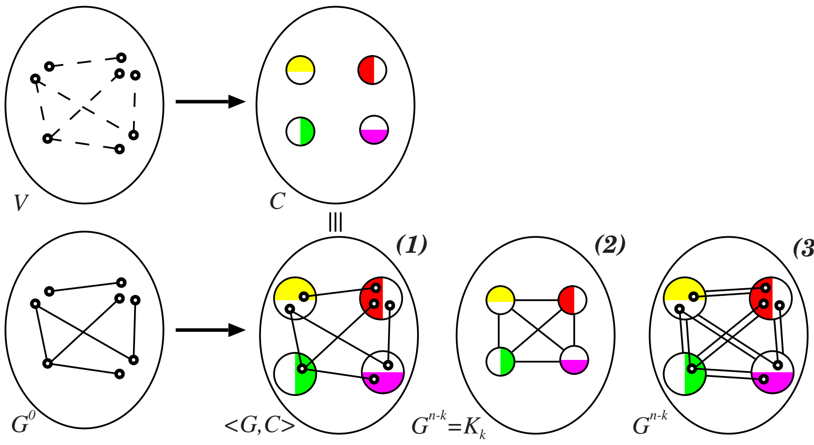


Fig. 11.2. Three homomorphisms from graphs to colourings

Figure 11.2. In this homomorphism, vertices grouped together form hyper-vertices as colour sets. The original vertices are associated with the colour set(s) where its adjacent vertices reside. When a normal vertex has more than one adjacent vertex in a colour set, then all the edges are folded into one hyper-edge. The cardinality of all the edges that were merged are assigned as a weight to the hyper-edge. This way we can keep track of the vertices in relation to the colours, and the cardinality allows us to measure the strength of such a relationship. We present a basic and a weighted representation of the contracted graph above, where vertices and edges are merged into one hyper-vertex or hyper-edge. The matrices that result from these merge operations are non-square, which is why the representation is called a table. Due to the basic and weighed edges, there are two types of representations, a *Binary* (BMT) and an *Integer* (IMT) type. The assigned A^{n-k} matrix dimension is $k \times n$. Columns represents normal vertices and rows refer to hyper-vertices, i.e., colour-sets. Zeros in a row define the members of the hyper-vertex. Non-zeros define edges between hyper-vertices and vertices.

3. The third homomorphism comes from the second by contracting normal nodes in color sets to one node. Let us define $\phi : G \to K$. It assigns a complete graph K to G. The number of vertices of K defines the quality of the colouring. The colouring is valid iff $\phi_V(u) = \phi_V(v) \Rightarrow (u, v) \notin E$. If we consider the original colour assignment then we merge all the vertices that have the same colour assigned. In other words, we can create the homomorphism by atomic sequential steps such as, colouring or merging one vertex at a time. Hence $n - k$ steps are needed to get K_k. An example is shown in Figure 11.2. In the previous homomorphism, the merge operations performed on G result in a complete graph on k vertices: K_k. Vertices assigned with the same colour are collapsed into a hyper-vertex, and all edges that connect to these vertices are collapsed into a hyper-edge. We provide a

representation of the resulting graph by two different matrices. The first matrix does not rely on the number of collapsed edges it is the basic adjacency matrix that represents the graph K_k, while the second does and it is the weighed adjacency matrix of the same graph. Here, the homomorphism is performed on the relevant matrices $\phi : A^0 \rightarrow A^{n-k}$. As the matrices that arise during the steps are square matrices, this model is called the *Merge Square* (MS) model. If we do not keep track of the cardinality of the edges we call this the *Binary Merge Square* model. Otherwise, we call this the *Integer Merge Square* model.

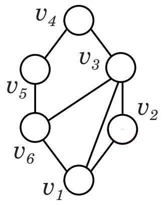
11.3.1 Merge Operations as Atomic Transformation Steps of the Homomorphisms

The different matrix representations require different kinds of merge operations. By combining two of these representations with two operations we will provide four colouring models. The representations are the Binary Merge Square or Integer Merge Square and the Binary Merge Table or Integer Merge Table, which are denoted by $A, \mathbb{A}, T, \mathbb{T}$ respectively. Merge Squares refers to the adjacency matrix of the merged graph. The Integer types assign weights to the (hyper-)edges according to the number of edges merged into hyper-edges. The Binary types collapse these edges simply into one common edge. The tables keep track of the vertices that were merged into a hyper-vertex and their cardinality. The graph depicted in Figure 11.3 and its adjacency matrix will be used in examples to explain the different models.

The zeroth element of the steps is the adjacency matrix of G in all cases:

$$A^0 = \mathbb{A}^0 = T^0 = \mathbb{T}^0 := A$$

We only deal with valid colourings, hence only non-adjacent vertices can be merged together. In case of Merge Squares representations columns and rows refer to the same (hyper-)vertex/colour-set. Thus, the condition of the merge depends on the relation between hyper-vertices; the coincidence of the appropriate row and column of the Merge Square must be zero. We can easily see that this condition is the same for Merge Tables (MT). Hence, we have to check



	v_1	v_2	v_3	v_4	v_5	v_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

Fig. 11.3. A graph G and its adjacency matrix: v-s refer to vertices and r-s refer to rows, i.e., colours

the adjacency between a normal vertex (which refers to an MT column) and a hyper-vertex/colour-set (which refers to an MT row). We can summarise the merge conditions by $a^t_{ij} = \mathbb{a}^t_{ij} = t^t_{ij} = \mathbb{t}^t_{ij} = 0$. Consequently $a^t_{ji} = \mathbb{a}^t_{ji} = 0$ and thanks to the inherited graph property $a^t_{ii} = a^t_{jj} = \mathbb{a}^t_{ii} = \mathbb{a}^t_{jj} = 0$.

We define the following matrices:

$$P : \mathbf{e}_i\mathbf{e}'_j \qquad M : \mathbf{e}_j\mathbf{e}'_j \qquad W = P - M,$$

where P (Plus) will be used for addition (or bitwise OR operation) of the j-th row of a matrix to its i-th row. M (Minus) will support the subtraction of the j-th row from itself, thereby setting its components to zero. This could be done also by a bitwise exclusive or (XOR). W combines these operations together. Let $\mathbf{a}$ and $\mathbf{b}$ define the i-th and j-th row vector of a merge matrix in step t. We need only $n - k$ contraction steps to get a solution instead of n needed by the traditional colouring methods. As much hardware nowadays provide CPUs with vector operations, this opens the possibility to perform the atomic merge operations in one CPU operation, thereby increasing the efficiency.

We now define the four models formulated both as row/column operations and matrix manipulations. First the integer-based models and then the binary-based model, which do not track the number of edges folded into a hyper-edge.

Integer Merge Table (IMT) model

Row based formulation of the i-th and j-th row of $\mathbb{T}$ after merging j-th vertex into i-th: let $\mathbb{t}_i$ is the i-th row and $\mathbb{t}_{\_i}$ is the column vector.

$$\mathbb{t}^{t+1}_i = \mathbf{a} + \mathbf{b} \quad , \quad \mathbb{t}^{t+1}_j = \mathbf{0} \tag{11.1}$$

Matrix based formulation:

$$\mathbb{T}^{t+1} = \mathbb{T}^t + W\mathbb{T}^t = (I + W)\mathbb{T}^t \tag{11.2}$$

In the example below, rows r_1 and r_4 are merged, after which the row corresponding to colour c_4 is removed.

	v_1	v_2	v_3	v_4	v_5	v_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

	v_1	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	0	1	2	0	1	1
r_2	1	0	1	0	0	0
r_3	1	1	0	1	0	1
r_4	0	0	0	0	0	0
r_5	0	0	0	1	0	1
r_6	1	0	1	0	1	0

	v_1	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	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

Integer Merge Square (IMS) model

Row/column based formulation: let $\mathbb{a}_i$ be the i-th row and $\mathbb{a}_{\_i}$ be the column vector and define $\mathbb{a}_j$ and $\mathbb{a}_{\_j}$ in the same way for the j-th row and column.

$$\mathbb{a}_i^{t+1} = \mathbf{a} + \mathbf{b}\ , \quad \mathbb{a}_j^{t+1} = \mathbf{0}' \tag{11.3}$$

$$\mathbb{a}_{\text{-}i}^{t+1} = \mathbf{a}' + \mathbf{b}'\ , \quad \mathbb{a}_{\text{-}j}^{t+1} = \mathbf{0} \tag{11.4}$$

Matrix based formulation:

$$\mathbb{A}^{t+1} = \mathbb{A}^t + W\mathbb{A}^t + \mathbb{A}^t W' \tag{11.5}$$

Since, $\mathbb{a}_{ij}^t = 0$ and $\mathbb{a}_{ji}^t = 0$, hence $W\mathbb{A}^t W'$=0. Due to this fact we can rewrite (11.5) as

$$\mathbb{A}^{t+1} = (I + W)\mathbb{A}^t (I + W)' \tag{11.6}$$

In the example below, a merge square has caused both columns and rows to be merged. The result is an adjacency matrix of the merged graph with weights on the edges, which describe the number of edges that were merged.

	v_1	v_2	v_3	v_4	v_5	v_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

	$\{v_1, v_4\}$	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	0	1	2	0	1	1
r_2	1	0	1	0	0	0
r_3	2	1	0	0	0	1
r_4	0	0	0	0	0	0
r_5	1	0	0	0	0	1
r_6	1	0	1	0	1	0

	$\{v_1, v_4\}$	v_2	v_3	v_5	v_6
$\{r_1, r_4\}$	0	1	2	1	1
r_2	1	0	1	0	0
r_3	2	1	0	0	1
r_5	1	0	0	0	1
r_6	1	0	1	1	0

Binary Merge Table Model (BMT) model

Row based formulations:

$$\begin{aligned} t_i^{t+1} &= \quad \mathbf{a} \vee \mathbf{b} \quad , && t_j^{t+1} = \mathbf{0} && (11.7)\\ t_i^{t+1} &= \mathbb{t}_i^{t+1} - \mathbf{a} \bullet \mathbf{b} \quad , && t_j^{t+1} = \mathbf{0} && (11.8)\\ \mathbf{a} \bullet \mathbf{b} &= \quad diag(\mathbf{a}'\mathbf{b}) && = \sum_i (\mathbf{a}'\mathbf{b}) e_i \end{aligned}$$

Matrix based formulations:

$$T^{t+1} = T^t \vee PT^t - MT^t \tag{11.9}$$

$$T^{t+1} = \mathbb{T}^{t+1} - \sum_j (\mathbf{a}'\mathbf{b}) E_{ji} \tag{11.10}$$

In the example below, row r_4 is merged with row r_1 to form $\{r_1, r_4\}$, after which r_4 is deleted.

	v_1	v_2	v_3	v_4	v_5	v_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

	v_1	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	0	1	1	0	1	1
r_2	1	0	1	0	0	0
r_3	1	1	0	1	0	1
r_4	0	0	0	0	0	0
r_5	0	0	0	1	0	1
r_6	1	0	1	0	1	0

	v_1	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	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

Binary Merge Square (BMS) model

Row/column based formulations: let a_j be the j-th row and $a_{\text{-}j}$ be the column vector.

$$a_i^{t+1} = \quad \mathbf{a} \vee \mathbf{b} \quad , \qquad a_j^{t+1} = \mathbf{0}' \tag{11.11}$$

$$a_{\text{-}i}^{t+1} = (A_i^{t+1})' \quad , \qquad a_{\text{-}j}^{t+1} = \mathbf{0} \tag{11.12}$$

Matrix based formulations:

$$A^{t+1} = A^t \vee (PA^t + A^tP') - (MA^t + A^tM') \tag{11.13}$$

$$A^{t+1} = A^t \vee (PA^tP') - (MA^tM') \tag{11.14}$$

The example below shows a binary merge collapse, which does not perform any accounting of merged structures.

	v_1	v_2	v_3	v_4	v_5	v_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

	$\{v_1, v_4\}$	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	0	1	1	0	1	1
r_2	1	0	1	0	0	0
r_3	1	1	0	0	0	1
r_4	0	0	0	0	0	0
r_5	1	0	0	0	0	1
r_6	1	0	1	0	1	0

	$\{v_1, v_4\}$	v_2	v_3	v_5	v_6
$\{r_1, r_4\}$	0	1	1	1	1
r_2	1	0	1	0	0
r_3	1	1	0	0	1
r_5	1	0	0	0	1
r_6	1	0	1	1	0

11.3.2 Information Derived from the Data Structures

First order structures are the cells of the representation matrices. They define the neighbourhood relation of the hyper-vertices for the binary and weighted relation for the integer models.

Secondary order structures or *co-structures* are the summary of the first order structures; i.e., the rows and columns in the representation matrices respectively. They form four vectors. Since, sequential colouring algorithms take steps, and the coloured and uncoloured part of the graphs are changing step-by-step

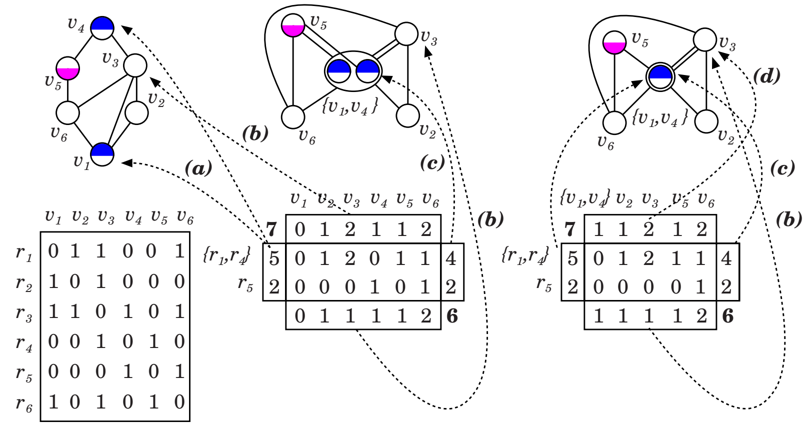


Fig. 11.4. The original graph, its induced sub-IMT and then its induced sub-IMS when colouring is in progress. (a) gives the sum of the degree of the nodes, (b) gives the number adjacent vertices already assigned a colour, (c) gives the degree of the hyper-vertex, and (d) gives the number of coloured hyper-edges.

it is worth to define these structures separately to the coloured/merged and uncoloured/non-merged sub-graphs. To identify these partial sums we use ^{col} and ^{unc} superscripts. We can obtain the sum of the rows and columns of binary merge matrices from their integer pairs by counting their non-zero elements. Hence, in Figure 11.4 only the left side of the sub-matrices sum the columns and the right side sum the non-zero elements to get the relevant summary in the binary matrix. This is the same for the columns, where the top vector is the sum of the rows and the bottom is the number of non-zeros. The second order structures are denoted by τ, using $_{t,b,l,r}$ indices as subscript to refer to the top, bottom, left and right vector.

Third order structures are formed by summarising the secondary order structures. These can be divided into two parts similar to the second order structures according to the coloured and uncoloured sub-graphs. These structures are denoted by ζ. In this study, they will be used in the fitness function of the evolutionary algorithm. The top-left sums the top vector (or the left vector) and the bottom-right sums the bottom vector (or the right vector). These are shown in bold font in Figure 11.4.

11.4 Heuristics Based on Merge Model Structures

before we define the heuristics, we first introduce two abstract sequential algorithms. These are appropriate for defining existing or novel GCP solvers in a concise manner.

Definition 1 (Sequential contraction algorithm 1 (SCA1))

1. *Choose a non-merged/uncoloured vertex v*
2. *Choose a non-neighbour hyper-vertex/colour-set r to v*
3. *Merge v to r*
4. *If there exists a non-merged vertex then continue with Step 1*

Definition 2 (Sequential contraction algorithm 2 (SCA2))

1. *Choose a hyper-vertex/colour-set r*
2. *Choose a non-neighbor non-merged/uncoloured vertex v to r*
3. *Merge v to r*
4. *If there exists a non-merged vertex then continue with Step 1*

The proceedings of the DIMACS Challenge II [240] states a core problem of the algorithm design for graph colouring: it is crucial to find an initially good solution, because each algorithm needs exhaustive searching to find an optimal solution, hence the problem becomes $\mathcal{NP}$-hard to solve. There are several polynomial algorithms (see [241, 242, 243, 244]), which provide a guarantee for the approximation of the colouring for a given number of colours. For example, ERDŐS HEURISTICS [242] guarantees at most $\mathcal{O}(n/log_{\chi}(n))$ number of colours in the worst case. Several algorithms have improved upon this bound, but many of them use the main technique introduced by Erdős. We will define this heuristics in our model. Another reference method is the well known heuristic of Brèlaz; the DSATUR algorithm, which works very well on graphs that have a small chromatic number. Two additional novel heuristics are introduced: the DOTPROD and COS heuristics.

DSatur

DSATUR is a SCA1 type algorithm. Where the choice of colouring the next uncoloured vertex v is determined by colour saturation degree. As colour saturation is calculated by observing the colours of neighbouring vertices, it requires $\mathcal{O}(n^2)$ constraint checks. However, if we merge the already coloured vertices, e.g., using MTs, we have the possibility to obtain this information by observing at most the number of hyper-vertices for v. Hence, $\mathcal{O}(nk_t)$ constraint checks are required, where k_t is the number of colours used in step t. The bottom co-structure of the relevant sub-IMT provide the saturation degree exactly which gives $\mathcal{O}(n)$ computational effort to find the largest one. Hence, IMT is an appropriate structure for the definition. Here, the choice for hyper-vertex/colour-set is done in a greedy manner.

Definition 3 (Non-merged/Uncoloured vertex choice of the DSatur$_{\mathbf{imt}}$)

1. *Find those non-merged/uncoloured vertices which have the highest saturated degree: $S = \arg\max_u \tau_b^{col}(u) : u \in V^{unc}$*
2. *Choose those vertices from S that have the highest non-merged/uncoloured-degree: $N = \arg\max_v \tau_b^{unc}(v)$*
3. *Choose the first vertex from the set N*

DotProd

This heuristic is based on a novel technique, which is shown to be useful in [239]. Two vertices of a contracted graph are compared by consulting the corresponding BMS rows. The dot product of these rows gives a valuable measurement for the common edges in the graph. These values are the same in binary MT and MS but can differ between the binary and integer variations. Application of the DotProd heuristics to the BMS representation provides the *Recursive Largest First* (RLF) algorithm. Unfortunately the name RLF is somewhat misleading, since the largest first itself does not define exactly where largest first is relating to. The meaning of it differs throughout the literature. Here, we introduce a DotProd heuristic that is combined with the BMS representation and the SCA2 type algorithm, which results in the RLF of Dutton and Birgham [245]. We explore only thess combinations, but other combinations are possible, making the DotProd a more general technique.

Definition 4 (Non-merged/Uncoloured vertex choice of the DotProd$_{\mathbf{bms}}$)

1. *Find those non-merged/uncoloured vertices which have the largest number of common neighbours with the first hyper-vertex (colour set):* $S = \arg\max_u \langle u, r\rangle : u \in V^{unc}$
2. *Choose the first vertex from the set* S

Cos

The Cos heuristics is the second novel heuristics introduced here. It is derived from DotProd by normalisation of the dot product. As opposed to DotProd, Cos takes in consideration the number of non-common neighbours as well. In the following definition we provide an algorithm of type SCA2 that uses the Cos heuristics to choose the next vector:

Definition 5 (Non-merged/Uncoloured vertex choice of the Cos$_{bms}$)

1. *Find the non-merged (i.e., uncoloured) vertices that have the largest number of common neighbours with the first hyper-vertex (colour set), and that have the least number of constraints:* $S = \arg\max_u \frac{\langle u,r\rangle}{\|u\|\|r\|} \equiv \arg\max_u \frac{\langle u,r\rangle}{\|u\|} \equiv \arg\max_u \frac{\langle u,r\rangle}{\tau_r(u)} : u \in V^{unc}$
2. *Choose the first vertex from the set* S

Pál Erdős $O(n/\log n)$ number of colours guaranteed heuristic

The approach [242, page 245] is as follows. Take the first colour and assign it to the vertex v that has the minimum degree. Vertex v and its neighbours are removed from the graph. Continue the algorithm in the remaining sub-graph in the same fashion until the sub-graph becomes empty, then take the next colour and use the algorithm for the non-coloured vertices and so on until each vertex is assigned a colour.

All of representations are suitable as a basis for this heuristic. It uses SCA2, where the choice for the next target r-th (hyper-)vertex/colour-set for merging and colouring is greedy.

Definition 6 (Non-merged/Uncoloured vertex choice of the Erdős$_{\mathbf{bmt}}$)

1. *Choose an uncoloured vertex with minimum degree.* $S = \arg\min_u \tau_b^{unc}(u)$
2. *Choose the first vertex from the set* S

The Erdős and DSatur heuristics make use of the secondary order structures, whereas the other two, DotProd and Cos, make use of the first order information. The Erdős heuristic uses similar assumptions as DSatur but in the opposite direction, which becomes clear in the results from the experiments.

11.5 An Evolutionary Algorithm Based on Merge Models

We apply an evolutionary algorithm (EA) to guide the heuristics. It uses the BMT representation for colouring and the SCA1 contraction scheme. The genotype is a permutation of the vertices, i.e., the rows of the BMT. The phenotype is a final BMT, where no rows can be merged further. Vertex selection is defined by an initial random ordering of all vertices. The order is then evolved by the EA toward an optimum directed by both the fitness function and the genetic operators of the EA. The strategy for merging of selected vertices is guided by one of the DotProd and Cos heuristics.

An intuitive way of measuring the quality of an individual (permutation) 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[1] is χ then we may normalise the 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 [234] to allow the use of first and second order structures introduced in Section 11.3.2.

The fitness relies on the heuristic that one generally wants to avoid highly constraint vertices 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 last merge the final BMT defines the groups of vertices with the same colour. There are $k(p)-\chi$ over-coloured vertices, i.e., merged rows. Generally, we use the indices of the over-coloured vertices to calculate the number of vertices that need to be minimized (see $g(p)$ above). But these vertices are not necessarily responsible for the over-coloured graph. Therefore, we choose to count the hyper-vertices 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 vertices discussed previously.

[1] In our experiments χ is defined in advance.

The fitness function used in the EA is then defined as follows. Let $\zeta^{unc}(p)$ denote the number of constraints, i.e., non-zero elements, in the rows of the final BMT that belong to the over-coloured vertices, i.e., the sum of the smallest $k(p) - \chi$ values of the right co-structure of the uncoloured vertices. This is the uncoloured portion of the (right-bottom) third order structure. The fitness function becomes $f(p) = g(p)\zeta^{unc}(p)$. Here, the cardinality of the problem is known, and used as a termination 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.

Procedure EA$_{\mathbf{dot}}$ and EA$_{\mathbf{cos}}$

```
1. population = generate initial permutations randomly
2. repeat
       evaluate each permutation p:
         merge $p_j$ − th unmerged vertex v into hyper-vertex r by DotProd or Cos
         calculate $f(p) = (k(p) - \chi)\zeta^{unc}(p)$
       $population_{xover}$ = xover(population, $prob_{xover}$)
       $population_{mut}$ = mutate($population_{xover}$, $prob_{mut}$))
       population = $select_{2\text{-}tour}$(population ∪ $population_{xover}$ ∪ $population_{mut}$)
3. until termination condition
```

Fig. 11.5. The EA meta-heuristic uses directly the BMT structure and either the DotProd or Cos merge strategy

Figure 11.5 shows the outline of the evolutionary algorithm. It uses a generational model with 2-tournament selection and replacement, where it employs 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 offspring. First, the 2-point order-based crossover (OX2) [246, in Section C3.3.3.1] is applied. Second, 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 0.4 and the probability for using the simple swap mutation is set to 0.6. These parameter settings are taken from previous experiments [234].

The Erdős heuristic guarantees its performance. We omit this heuristics from the EA, as we would not be able to guarantee this property once embedded in the EA. A baseline version of the EA called EA_{noheur} serves a basis of the comparison. and the DSatur with backtracking is used as well, as it is commonly used as a reference method. Moreover, as this algorithm performs an exhaustive search it is useful to find the optimal solutions to some of the problem instances; some of the instances are too difficult for it to solve and we have to terminate it prematurely.

11.6 Experiments and Results

The test suites are generated using the well known graph k-colouring generator of Culberson [247]. It consists of k-colourable graphs with 200 vertices, where

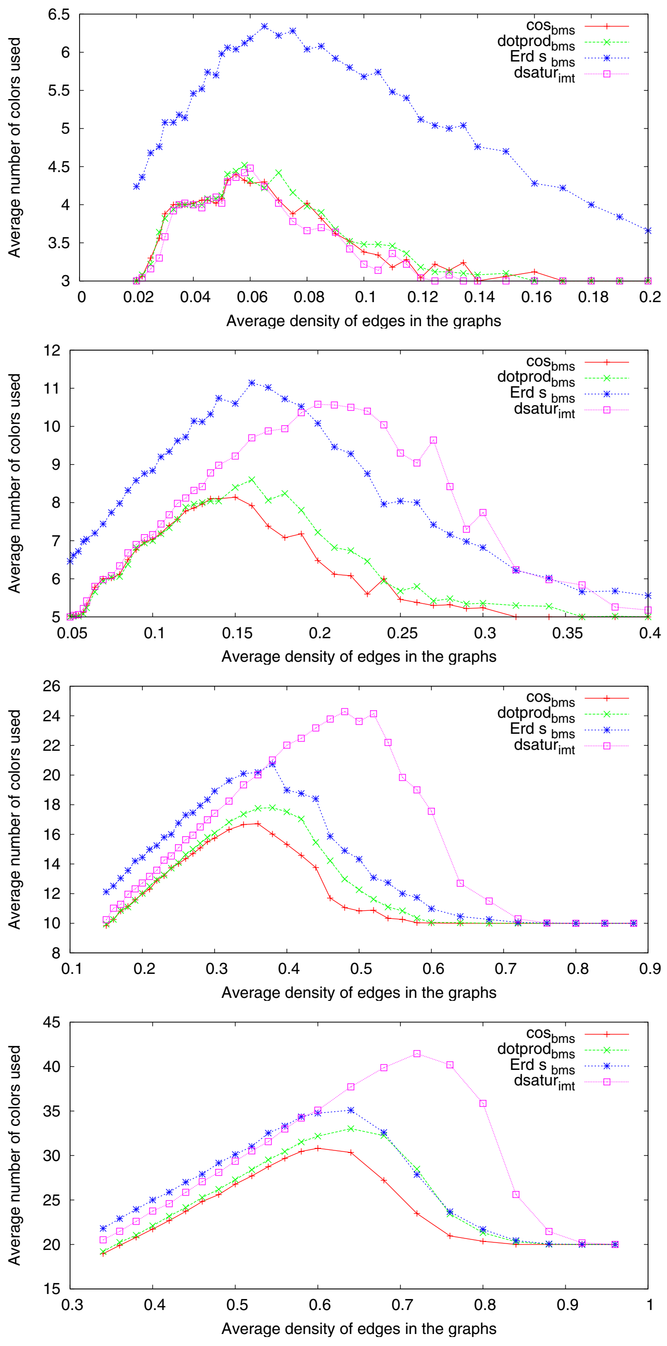


Fig. 11.6. Average number of colours used in sequential colouring heuristics. χ is 3, 5, 10, and 20, respectively in the sequence of figures.

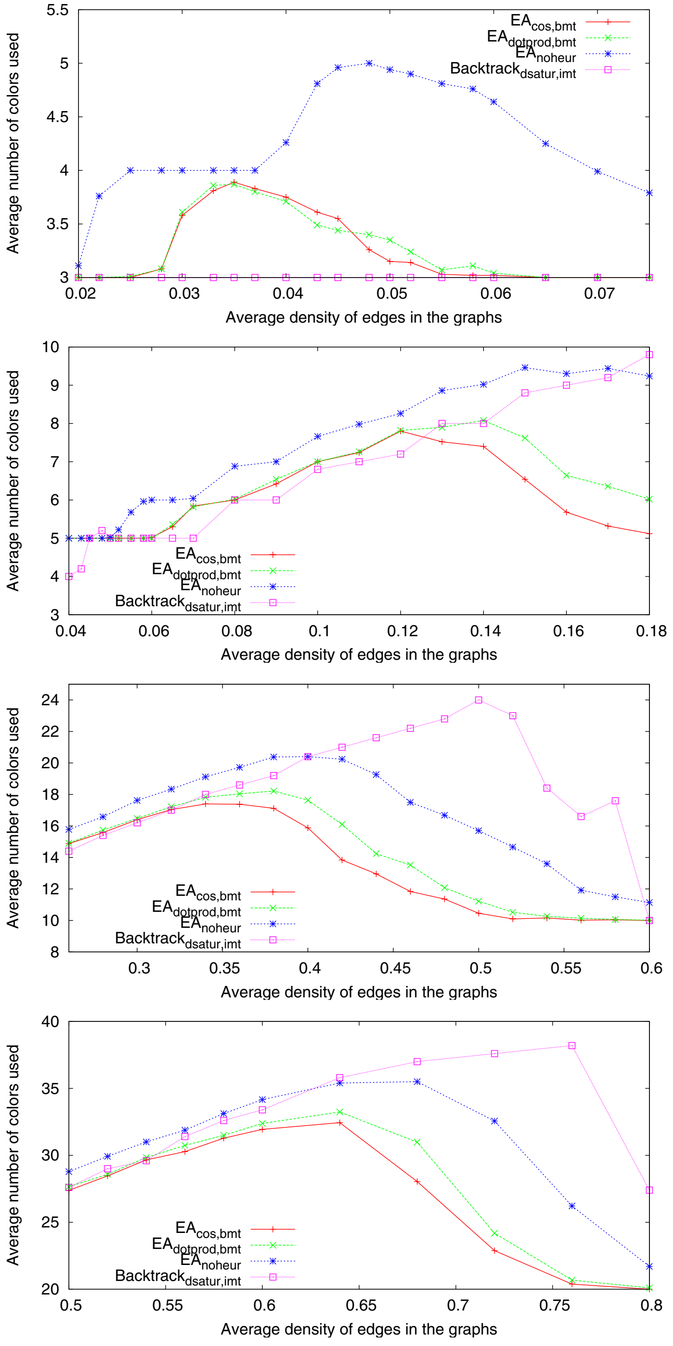


Fig. 11.7. Averaged number of colours used in the EA experiments. χ is 3, 5, 10, and 20, respectively in the sequence of the figures.

k is set to 3, 5, 10 and 20. For $k = 20$, ten vertices will form a colour set, therefore we do not use any larger chromatic number. 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 observed in the results as a typical easy-hard-easy pattern. The graphs are all generated to be equi-partite, which means that a solution should use each colour approximately as much as any other. The suite consists of groups where each group is a k-colouring with 20 unique instances.

In the first experiment, we compare the heuristics from Section 11.4 when embedded in DSatur. Figure 11.6 shows the results for each of the heuristics. Cos heuristics performed clearly better than the others except for the 3-colouring where DSatur performs equally well. The DotProd is ranked second. While DSatur performs well on sparse graphs having small chromatic number, Erdős heuristics performs well on graphs that require more colours, especially on dense graphs, i.e., that have a high average density of edges. Interesting is the location of the phase transitions. Figure 11.6 shows that it depends not only on the edge density of the graphs but also on the applied algorithm, especially the graph density where DSatur exhibits its worst performance moves away from the others with increasing k. DSatur and Erdős heuristics apply only second order information as opposed to the other two algorithms where first order information is used. The Erdős heuristic uses the secondary order structures in the opposite direction to DSatur and our results show how that affects the performance as the effectiveness flips for different ends of the chromatic number of graphs.

While sequential algorithms make one run to get their result, EA experiments are performed several times due to the random nature of the algorithm. Therefore a reduced number of instances is selected from the previous experiments. One set consists of five instances, except for 3 and 5 colourable instances where the set contains ten instances because the diversity of the results of the solvers is low for small chromatic numbers. On each instance we perform ten independent runs and calculate averages over the total number of runs. 3-colouring needs more instances to get more confident results.

Figure 11.7 shows the results for the EA with two different heuristics. Also shown are results for the reference EA without heuristics and the DSatur exhaustive algorithm. Similar to the previous experiments, the Cos heuristics performs well, especially for larger k, and the DotProd is a close second. DSatur is the strongest algorithm on 3-colourable graphs, always finding the optimum number of colours.

11.7 Conclusions

In this paper, we introduced four kinds of Merge Models 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, access to heuristics that help guide the search, and a compact description of algorithms. We showed how existing algorithms can be described in terms of the Merge Model concisely.

By incorporating the models in an exhaustive algorithm, DSATUR, and in a meta-heuristic, an evolutionary algorithm based on a permutation, we showed how the effectiveness of these algorithms can be improved.

Acknowledgements

This work is supported by the High Performance Computing Group of the University of Szeged. The second author is supported through the e-Science Data Information & Knowledge Transformation project funded by the Scottish Funding Council.