Software for proper vertex-colouring exploiting graph symmetry *

Leonard H. Soicher[0000−0001−5836−8914]

School of Mathematical Sciences
Queen Mary University of London
Mile End Road, London E1 4NS, UK
L.H.Soicher@qmul.ac.uk

Abstract. We describe the methods used in the GAP package GRAPE for proper vertex-colouring a graph, including the determination of a minimum vertex-colouring and hence the chromatic number. These methods are designed to exploit the automorphism group of the graph.

Keywords: proper vertex-colouring · minimum vertex-colouring · chromatic number · graph symmetry · GRAPE · GAP.

* This version of the contribution has been accepted for publication, after peer review but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://doi.org/10.1007/978-3-031-64529-7_12. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms.

1 Introduction

The GAP system [2] is a freely available open-source computer system for algebra and discrete mathematics, with an emphasis on computational group theory. See [3] for a tutorial introduction to GAP.

The GRAPE package [12] for GAP provides extensive functionality for graphs, and is designed primarily for applications in algebraic graph theory, permutation group theory, design theory, and finite geometry. See [13] for a tutorial introduction to GRAPE, including the use of much of the functionality described in this article.

In GRAPE, a graph *gamma* always comes together with an associated group *gamma*`.group` of automorphisms. This group is set (automatically or by the user) when the graph is constructed, and is used by GRAPE to store the graph compactly and to speed up computations with the graph. Often, but not always, this group is the full automorphism group of the graph.

GRAPE includes functionality for constructing graphs, determining their regularity properties, and classifying their cliques. GRAPE also provides seamless interfaces to both the nauty [11] and bliss [7] computer packages for computing the automorphism group of a graph and testing graph isomorphism.

GRAPE now also has machinery for properly vertex-colouring a graph, which exploits the automorphism group of that graph. This functionality includes the calculation of a minimum vertex-colouring and hence the determination of the chromatic number of the graph. We shall describe the main ideas and methods used for this functionality. We expect these to be of broader interest and application.

Our proper vertex-colouring software in GRAPE is meant to be of practical use, especially for graphs with large automorphism groups. This software was used to compute many of the chromatic numbers given in [1, Chapter 10] for specific interesting strongly regular graphs. We present two further examples in the last section.

Throughout this article, all graphs are *simple*, meaning they are finite, undirected, and have no loops and no multiple edges.

2 Proper vertex-colouring and cliques

Let Γ be a graph. A *proper vertex-colouring* of Γ is a labelling of its vertices by elements from a set of *colours*, such that adjacent vertices are labelled with different colours. Where k is a non-negative integer, a *vertex k-colouring* of Γ is a proper vertex-colouring using at most k colours. A *minimum vertex-colouring* of Γ is a vertex k-colouring with k as small as possible, and the *chromatic number* $\chi(\Gamma)$ of Γ is the number of colours used in a minimum vertex-colouring of Γ.

The problem of whether a given graph has a vertex k-colouring for a given k is a well-known NP-complete problem. Indeed, the problem is still NP-complete even for fixed $k = 3$ [4, Chapter 8]. Moreover, the problem of determining whether a graph has a vertex k-colouring for a given k appears to be very

difficult in practice. As users of GRAPE are usually interested in graphs with non-trivial, and often large, automorphism groups, it is important to be able to exploit any symmetry a graph may have when determining a vertex k-colouring for a given k or showing that no such vertex-colouring exists.

A *clique* of Γ is a set of pairwise adjacent vertices. Where t is a non-negative integer, a *t-clique* is a clique of size t. A *maximal clique* of Γ is a clique which is contained in no larger clique, while a *maximum clique* of Γ is a t-clique with t as large as possible. The *clique number* $\omega(\Gamma)$ of Γ is the number of vertices in a maximum clique of Γ. Clearly, $\omega(\Gamma) \leq \chi(\Gamma)$.

The problem of whether a given graph has a t-clique for a given t is a well-known NP-complete problem [4, Chapter 8]. However, this problem appears to be less difficult in practice than determining whether the graph has a vertex k-colouring for a given k. GRAPE contains powerful machinery for clique classification and for the determination of a maximum clique.

Now note that, up to the naming of the colours, the vertex k-colourings of Γ are in one-to-one correspondence with the partitions of the vertex set of the complement $\bar{\Gamma}$ of Γ into at most k cliques of $\bar{\Gamma}$ (the parts in such a partition are the colour classes of a vertex k-colouring of Γ). Also note that if g is an automorphism of Γ (and hence of $\bar{\Gamma}$) and $\mathcal{C}$ is an ordered partition of $V(\bar{\Gamma})$ into m cliques, then the g-image of $\mathcal{C}$ is also an ordered partition of $V(\bar{\Gamma})$ into m cliques.

The first step in GRAPE to try to find a vertex k-colouring of Γ is to perform a relatively inexpensive heuristic proper vertex-colouring (see [9]), but if this does not result in a vertex k-colouring, then a backtrack search is performed to find the "least" (defined later) ordered partition $(C_1, C_2, \ldots, C_m)$ of the vertices of $\bar{\Gamma}$ into cliques, such that $m \leq k$. This search will either find a vertex k-colouring of Γ or prove that such a colouring does not exist.

The method used at present in GRAPE for the determination of a minimum vertex-colouring of Γ (and hence $\chi(\Gamma)$) is a binary search for the least k for which a vertex k-colouring of Γ exists, together with the determination of a vertex k-colouring for this least k. To start with, a lower bound for k can be taken to be $\omega(\Gamma)$ or $\lceil |V(\Gamma)|/\omega(\bar{\Gamma}) \rceil$ and an upper bound for k can be obtained by a proper vertex-colouring heuristic, many of which are described in [9], the simplest of which is "greedy colouring".

In the GRAPE package for GAP, the user functions for proper vertex-colouring are `VertexColouring` (for vertex k-colouring), `MinimumVertexColouring`, and `ChromaticNumber`. In GRAPE, a proper vertex-colouring of a graph is output as a sequence of positive integers indexed by the vertices of the graph, with the i-th element of the sequence being the colour of vertex i.

3 The main tools

We now describe the three main tools from GRAPE that we use in our vertex k-colouring algorithm.

The first tool is used to classify the maximal cliques of given size in a graph, up to the action of a given group of automorphisms of that graph. In GRAPE, where *gamma* is a (simple) graph and t is a non-negative integer, the function call

`CompleteSubgraphsOfGivenSize(`*gamma*`,` t`, 2, true)`

returns a set of *gamma*`.group` orbit-representatives of all the maximal cliques of size t in *gamma*.

The second tool is Steve Linton's function `SmallestImageSet`, which is included in GRAPE. Where G is a permutation group on $X := \{1, \ldots, n\}$ and S is a subset of X, the function call

`SmallestImageSet(` G`,` S `)`

returns the lexicographically least set in the G-orbit of S with respect to the natural action of G on subsets of X, without explicitly computing this (possibly huge) orbit. We use the `SmallestImageSet` function to determine the lexicographically least clique in a group orbit, given an arbitrary clique in that orbit.

The algorithm for `SmallestImageSet` is given in [10]. Further developments in the computation of minimal and canonical images with respect to a group action are given in [5, 6].

Our third tool again employs `CompleteSubgraphsOfGivenSize`, this time for the calculation of exact set covers, exploiting symmetry, as detailed in Figure 1. See the GRAPE manual [12] for full documentation of the very flexible function `CompleteSubgraphsOfGivenSize`, as well as other GRAPE functions used.

4 A total ordering of finite sequences of subsets of $\{1, \ldots, n\}$

Let n be a non-negative integer, and let $A := \{a_1, \ldots, a_r\}$ and $B := \{b_1, \ldots, b_s\}$ be subsets of $\{1, \ldots, n\}$, with $a_1 < \cdots < a_r$ and $b_1 < \cdots < b_s$. We define

$$A \preceq B$$

to mean either $r > s$, or $r = s$ and $(a_1, \ldots, a_r) \leq (b_1, \ldots, b_r)$ in lexicographic order (w.r.t. the usual $\leq$ on the integers). For example, $\{3, 5, 6\} \preceq \{2, 3\}$, but $\{3, 4, 7\} \preceq \{3, 5, 6\}$.

Let n be a non-negative integer and let $\mathcal{A} := (A_1, \ldots, A_t)$ and $\mathcal{B} := (B_1, \ldots, B_u)$ be finite sequences of subsets of $\{1, \ldots, n\}$. We define

$$\mathcal{A} \preceq \mathcal{B}$$

to mean that $(A_1, \ldots, A_t)$ is less than or equal to $(B_1, \ldots, B_u)$ in lexicographic order, with respect to the order $\preceq$ on subsets of $\{1, \ldots, n\}$. For example,

$$(\{3, 4, 7\}, \{3, 5, 6\}, \{7, 8\}) \preceq (\{3, 4, 7\}, \{2, 3\}) \preceq (\{3, 4, 7\}, \{2, 3\}, \{1, 2, 3, 4\}).$$

```
ExactSetCover := function(G,blocks,n)
#
# Let  n  be a positive integer, let  G  be a permutation group on
# [1..n],  let  blocks  be a set of non-empty subsets of  [1..n],
# and suppose  S  is the union of the G-orbits of the sets in  blocks.
# Then this function returns an exact set cover of  [1..n]  by elements
# from  S,  if such a cover exists, and returns  'fail'  otherwise.
#
local gamma,i,j,wts,K;
gamma:=Graph(G,blocks,OnSets,
   function(x,y) return Intersection(x,y)=[]; end);
wts:=[];
for i in [1..OrderGraph(gamma)] do
   wts[i]:=ListWithIdenticalEntries(n,0);
   for j in VertexName(gamma,i) do
      wts[i][j]:=1;
   od;
od;
K:=CompleteSubgraphsOfGivenSize(gamma,
   ListWithIdenticalEntries(n,1),0,true,true,wts);
if K=[] then
   return fail;
else
   return Set(VertexNames(gamma){K[1]});
fi;
end;
```

Fig. 1. Exact set cover using GRAPE

5 The backtrack search

Now let Γ be a graph with non-empty vertex set $V(\Gamma) := \{1, \ldots, n\}$, and let Δ be the complement graph $\bar{\Gamma}$ of Γ, such that the vertices of Δ can be partitioned into at most k cliques of Δ, for a given positive integer k (that is, Γ has a vertex k-colouring). Then there is a unique least ordered such partition $(C_1, \ldots, C_m)$ with respect to $\preceq$, and we shall consider some properties of this least ordered partition. These properties give us very useful constraints on partial solutions for our backtrack search, which either proves that this least ordered partition does not exist or finds a vertex k-colouring of Γ (which need not correspond to $(C_1, \ldots, C_m)$).

We follow the general structure of backtrack search as described in [8, Section 4.1.2]. Full details of our backtrack search can be found in the open-source code of GRAPE [12]. The degree of difficulty for this backtrack search depends heavily on k and the clique structure of Δ.

5.1 Constraints on partial solutions

Let $(C_1, \ldots, C_m)$ be the least ordered partition of $V(\Delta)$ into cliques of Δ, with respect to $\preceq$, such that $m \leq k$. Let $\Delta_1 := \Delta$, let $G_1 := \mathrm{Aut}(\Delta)$, and for $i = 2, \ldots, m$, let Δ_i be the subgraph of Δ_{i-1} induced on $V(\Delta_{i-1}) \setminus C_{i-1}$ and let G_i be the image of the action on $V(\Delta_i)$ of the (setwise) stabilizer in G_{i-1} of C_{i-1} (G_i is the group we associate to Δ_i in GRAPE). Then it is easy to see that the following must hold:

- C_i is a maximal clique of Δ_i.
- C_i is the lexicographically least set in its G_i-orbit.
- If $i > 1$ then $C_{i-1} \prec C_i$.
- $(k - i + 1)|C_i| \geq |V(\Delta_i)|$.

Given $(C_1, \ldots, C_{i-1})$, with $1 \leq i \leq m$, the possible C_i satisfying these properties can be generated (in increasing order w.r.t. $\preceq$) making use of the functions `CompleteSubgraphsOfGivenSize` and `SmallestImageSet`. Of course, if we are given $(C_1, \ldots, C_m)$ then we return this solution and stop.

We now note the following useful result.

Lemma 1. *Let $1 \leq j < i \leq m$, let D be a clique of Δ_j containing C_i and let $g \in G_j$. Then $C_j \preceq D^g$, the image of D under g.*

Proof. Suppose $D^g \prec C_j$. Then we could make an ordered partition $(D_1, \ldots, D_m)$ of $V(\Delta)$ into cliques, such that $(D_1, \ldots, D_m) \prec (C_1, \ldots, C_m)$, as follows. For $1 \leq \ell \leq j-1$, $D_\ell := C_\ell$; $D_j := D^g$; $D_i := (C_j \setminus D)^g$; for $j+1 \leq \ell \leq m$, $\ell \neq i$, $D_\ell := (C_\ell \setminus D)^g$. □

Now, for $1 \leq j < i \leq m$, let $C_{i,j}$ be the largest clique that can be obtained by adding (zero or more) elements of C_j to C_i. Then, applying Lemma 1, we obtain the following further constraints that we apply on partial solutions in our backtrack search.

- If $i > 1$ then the least G_{i-1}-image of $C_{i,i-1}$ is $\succeq C_{i-1}$.
- Suppose $(k - i + 1)|C_i| = |V(\Delta_i)|$. Then $m = k$ and $\{C_i, C_{i+1}, \ldots, C_k\}$ is a partition of $V(\Delta_i)$ into maximal cliques of Δ_i, each of size $|C_i|$.
  Now additionally suppose that G_i is small, say G_i has order at most 24 (so that G_i-orbits of cliques are also small). Then, after we classify the maximal cliques of size $|C_i|$ in Δ_i, up to the action of G_i, we determine which G_i-orbits of these cliques could possibly contain elements of $\{C_i, \ldots, C_k\}$. We use the requirement that if $i > 1$, then for $\ell = i, \ldots, k$, the least G_{i-1} image of $C_{\ell,i-1}$ is $\succeq C_{i-1}$ (note that this requirement need only be checked for one representative from each G_i-orbit). We then make use of (an inline version of) the function `ExactSetCover` either to complete the partial solution $(C_1, \ldots, C_{i-1})$ to an ordered partition of $V(\Delta)$ into k cliques or to show that no least (w.r.t. $\preceq$) such completion exists.

6 Examples

In the example given in Figure 2, we load the GRAPE package (suppressing the banner), construct the M_{23}-graph on 253 vertices (see [1, Section 10.56]), and determine some of its properties. In particular, we show that this graph has chromatic number 15, a fact which appears to have been previously unknown. The whole calculation took about 15 minutes of CPU-time on an i5 laptop, with the calculation of the chromatic number taking almost all of the time.

```
gap> LoadPackage("grape",false);
true
gap> G:=PrimitiveGroup(253,5);
M(23)
gap> M23graph:=First(GeneralizedOrbitalGraphs(G,1),
>    x->VertexDegrees(x)=[112]);;
gap> GlobalParameters(M23graph);
[ [ 0, 0, 112 ], [ 1, 36, 75 ], [ 60, 52, 0 ] ]
gap> A:=AutomorphismGroup(M23graph);;
gap> Size(A);
10200960
gap> RankAction(A,[1..253]);
3
gap> CliqueNumber(M23graph);
4
gap> CliqueNumber(ComplementGraph(M23graph));
21
gap> ChromaticNumber(M23graph);
15
```

Fig. 2. Example calculation

Similarly, we have constructed the M_{22}-graph on 176 vertices (see [1, Section 10.51]), and have determined that this graph has chromatic number 12, which also appears to have been previously unknown. This calculation took about one-half minute of CPU-time on an i5 laptop.

References

1. Brouwer, A. E., Van Maldeghem, H.: Strongly Regular Graphs. Cambridge University Press, Cambridge (2022)
2. The GAP Group: GAP — Groups, Algorithms, and Programming. Version 4.13.0 (2024). https://www.gap-system.org
3. The GAP Group: GAP — A Tutorial. Release 4.13.0 (2024). https://www.gap-system.org/Manuals/doc/tut/manual.pdf
4. Gibbons, A.: Algorithmic Graph Theory. Cambridge University Press, Cambridge (1985)

5. Jefferson, C., Jonauskyte, E., Pfeiffer, M., Waldecker, R.: Minimal and canonical images. Journal of Algebra **521**, 481–506 (2019)
6. Jefferson, C., Pfeiffer, M., Waldecker, R., Jonauskyte, E.: The images package for GAP, minimal and canonical images. Version 1.3.2 (2024). https://gap-packages.github.io/images/
7. Junttila, T., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In: Applegate, D. et al. (eds.) Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithmics and Combinatorics, pp. 135–149. SIAM, Philadelphia (2007). bliss homepage: http://www.tcs.hut.fi/Software/bliss/
8. Kaski, P., Östergård, P. R. J.: Classification Algorithms for Codes and Designs. Springer, Berlin (2006)
9. Lewis, R. M. R.: A Guide to Graph Colouring: Algorithms and Applications. 2nd edn. Springer International Publishing, Switzerland (2021)
10. Linton, S.: Finding the smallest image of a set. In: J. Gutierrez (ed.) ISSAC '04: Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, pp. 229–234. ACM Press, New York (2004)
11. McKay, B. D., Piperno, A.: Practical graph isomorphism, II. Journal of Symbolic Computation **60**, 94–112 (2014). nauty and Traces homepage: https://pallini.di.uniroma1.it
12. Soicher, L. H.: The GRAPE package for GAP. Version 4.9.0 (2022). https://gap-packages.github.io/grape/
13. Soicher, L. H.: Using GAP packages for research in graph theory, design theory, and finite geometry. In: Ivanov, A. A. (ed.) Algebraic Combinatorics and the Monster Group, London Mathematical Society Lecture Note Series **487**, pp. 527–566. Cambridge University Press, Cambridge (2024)