Cliques and colourings in GRAPE

Leonard Soicher

Queen Mary University of London

G2R2, Novosibirsk, August 2018

Vladimir, USSR, August 1991

Akademgorodok, Novosibirsk, Russia, August 2018

Thank you to the organizers!

This talk

- In this talk I want to introduce you to the extensive facilities in my GAP package GRAPE to compute and classify cliques in a given graph.
- I also want to talk about the very new features in GRAPE to compute a vertex k-colouring of a given graph (or to prove that none exist), and to compute a minimum vertex-colouring. For this functionality, the clique machinery is heavily used, and the automorphism group of the given graph is exploited.
- Throughout this talk, GAP will refer to GAP 4.9.2 and GRAPE will refer to GRAPE 4.8.

GAP and GRAPE

- GAP is an internationally developed, freely available, open source computer system for discrete mathematics, groups, and algebra.
- GRAPE is a GAP package for computing with graphs together with associated groups of automorphisms.
- It is designed for applications in algebraic graph theory, permutation group theory, design theory, and finite geometry.
- In GRAPE, a graph `gamma` comes together with a subgroup `gamma.group` of the automorphism group of `gamma`, and `gamma.group` is used to store the graph compactly and to speed up computations. This group is usually set by GRAPE when the graph is constructed, but may also be specifiied by the user.
- GRAPE also provides seamless interfaces to both *nauty* and *bliss* for computing automorphism groups of graphs and testing graph isomorphism.

Cliques and co-cliques

- All graphs in this talk are finite, undirected, with no loops and no multiple edges.
- A *clique* in a graph is a set of pairwise adjacent vertices. In other words, a clique in a graph Γ is a subset of the vertex set of Γ on which the induced subgraph is complete.
- A *maximal* clique of Γ is a clique which is not properly contained in any clique of Γ, while a *maximum* clique of Γ is a clique of largest size in Γ. The *clique number* $\omega(\Gamma)$ is the size of a maximum clique of Γ.
- A *co-clique* (or *independent set*) in a graph is a set of pairwise non-adjacent vertices. In other words, a co-clique in a graph Γ is a subset of the vertex set of Γ on which the induced subgraph is empty.
- Clearly, the co-cliques in a graph are precisely the cliques in the complement of that graph.

Computing cliques in GRAPE

- Let `gamma` be a (simple) graph in GRAPE, with associated group $G :=$ `gamma.group`.
- GRAPE functions can compute the cliques of `gamma` or the maximal cliques of `gamma` or the cliques of given size or vertex-weight sum in `gamma`, or the maximal cliques of given size or vertex-weight sum in `gamma`, such that at most one such clique is determined or it is determined that no such cliques exist, or G-orbit generators of all such cliques are determined, or G-orbit representatives of all such cliques are determined.
- Also, there is now (in GRAPE 4.8) functionality to determine a maximum clique, and hence to determine the clique number of `gamma`.

CompleteSubgraphsOfGivenSize

For example, in GRAPE, where `gamma` is a (simple) GRAPE graph, `k` is a non-negative integer and `maxi` is a boolean, the function call

```
CompleteSubgraphsOfGivenSize(gamma, k, 2, maxi)
```

returns a set of `gamma.group` orbit-representatives of

- all the cliques of size `k` of `gamma` if `maxi=false`;
- all the maximal cliques of size `k` of `gamma` if `maxi=true`.

Cliques in vertex-weighted graphs

As another example, suppose now `wts` is a list, indexed by the vertices of `gamma`, of positive integer vertex weights, such that this list of weights is `gamma.group`-invariant (for the action that permutes the list positions in the natural way). Then the function call

```
CompleteSubgraphsOfGivenSize(gamma, k, 0, true, true, wts)
```

returns (a set consisting of) just one maximal clique of vertex-weight sum `k` of the graph `gamma` if such a clique exists, and otherwise, the empty set is returned.

Cliques in graphs with vectors as vertex weights

More generally, the elements of the list `wts` of vertex weights can be non-zero d-vectors of non-negative integers, where it is required that `gamma.group` acts on the set of all integer d-vectors by permuting vector positions, such that, for all vertices v and all g in `gamma.group`, we have

$$\texttt{wts}[v^g] = \texttt{wts}[v]^g.$$

Cliques in graphs with vectors as vertex weights

More generally, the elements of the list `wts` of vertex weights can be non-zero d-vectors of non-negative integers, where it is required that `gamma.group` acts on the set of all integer d-vectors by permuting vector positions, such that, for all vertices v and all g in `gamma.group`, we have

$$\texttt{wts}[v^g] = \texttt{wts}[v]^g.$$

Then, where `k` is a `gamma.group`-invariant d-vector of non-negative integers, the function call

```
CompleteSubgraphsOfGivenSize(gamma, k, 2, false, true, wts)
```

returns a set of `gamma.group`-orbit representatives of the cliques of `gamma` whose vertex weights sum to `k`.

Cliques in graphs with vectors as vertex weights cont'd

- This type of clique functionality in GRAPE is used in my DESIGN package for GAP, for functionality to construct and classify block designs of many types (including those invariant under a specified group), as well as to construct and classify parallel classes and resolutions of block designs.

- I refer you to the DESIGN manual (and source code) for details.

Proper vertex-colourings

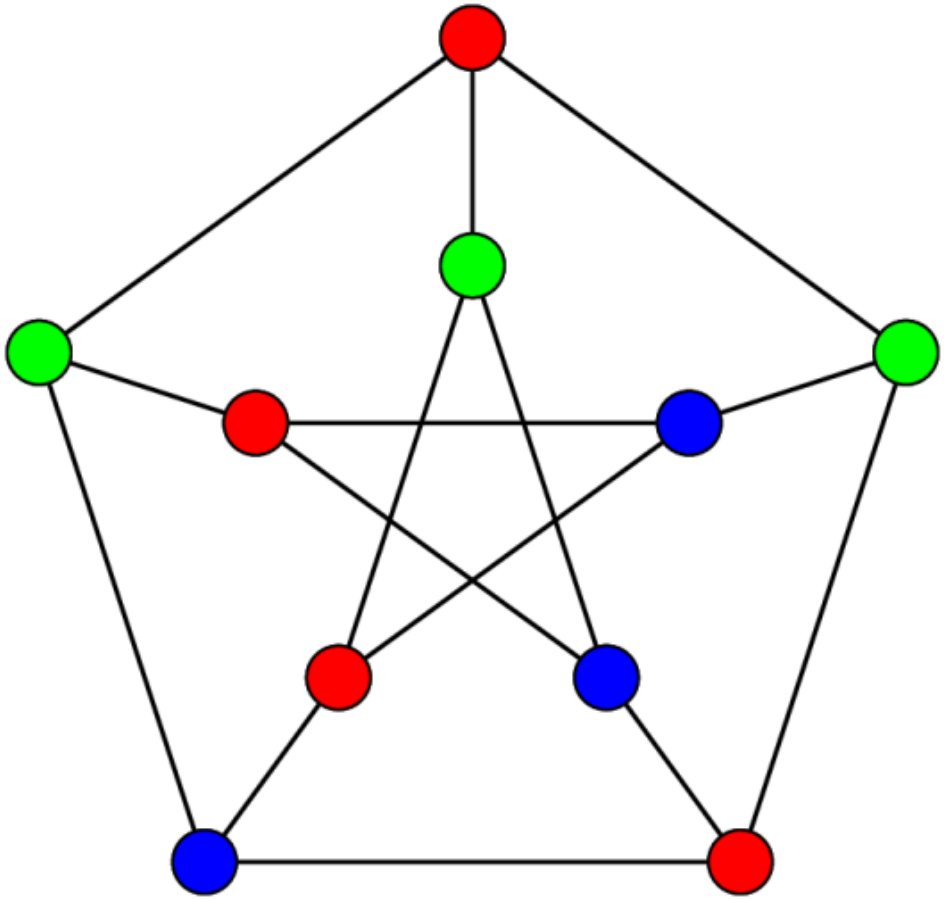
- A *proper vertex-colouring* of a graph is a labelling of the vertices of the graph by elements from a set of *colours*, such that adjacent vertices are labelled with different colours.

- Where k is a non-negative integer, a *vertex k-colouring* of a graph is a proper vertex-colouring using at most k colours.

- 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 Γ into at most k co-cliques, or equivalently, the partitions of the vertex set of the complement $\bar{\Gamma}$ of Γ into at most k cliques (the parts in such a partition are the colour classes of a vertex k-colouring of Γ).

- A *minimum vertex-colouring* of a graph Γ is a vertex k-colouring with k as small as possible, and the *chromatic number* $\chi(\Gamma)$ is the number of colours used in a minimum vertex-colouring of Γ.

A minimum vertex-colouring of the Petersen graph



https://graphtheoryinlatex.wordpress.com/2010/02/18/a-coloring-of-the-petersen-graph-2/

VertexColouring

In GRAPE, where `gamma` is a (simple) GRAPE graph and `k` is a non-negative integer, the function call

```
VertexColouring(gamma, k)
```

returns a vertex `k`-colouring of `gamma` if such a colouring exists; otherwise, the special value `fail` is returned.

In GRAPE, a proper vertex-colouring of a graph is given as a list of positive integers (the *colours*), indexed by the vertices of the graph, such that the i-th list element is the colour of vertex i.

Simple GRAPE example

```
gap> LoadPackage("grape",false);
true
gap> P:=Graph(SymmetricGroup([1..5]),
>   Combinations([1..5],2), OnSets,
>   function(x,y) return Intersection(x,y)=[]; end,
>   true);
rec( adjacencies := [ [ 8, 9, 10 ] ], group := Group([ (1,
   5,8,10,4)(2,6,9,3,7), (2,5)(3,6)(4,7) ]),
  isGraph := true,
  names := [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 1, 5 ],
      [ 2, 3 ], [ 2, 4 ], [ 2, 5 ], [ 3, 4 ], [ 3, 5 ],
      [ 4, 5 ] ], order := 10, representatives := [ 1 ],
  schreierVector := [ -1, 2, 2, 1, 1, 1, 2, 1, 1, 1 ] )
gap> VertexColouring(P,3);
[ 1, 1, 2, 3, 1, 2, 3, 2, 3, 2 ]
gap> VertexColouring(P,2);
fail
```

A major application of the GRAPE colouring functions

I have used GRAPE and its new proper vertex-colouring functionality (together with some theory) for the determination of the primitive permutation groups G of degree at most 255, such that G is a group of automorphisms of a non-null non-complete graph having chromatic number equal to its clique number.

These groups are precisely the non-synchronizing primitive permutation groups (of degree at most 255), of interest in both permutation group theory and automata theory.

Minimum vertex-colourings in GRAPE

- The method used in GRAPE for the determination of a minimum vertex-colouring of a graph Γ 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.

- Our approach to determining a vertex k-colouring is to perform a backtrack search for the "least" (defined later) ordered partition $(C_1, C_2, \ldots, C_m)$ of the vertices of $\bar{\Gamma}$ into cliques, such that $m \leq k$.

- The search will either find such a partition or prove no such partition exists. [For the puposes of this talk, we will ignore some heuristics that are used which mean that, in practice, a vertex k-colouring may be found that does not correspond to the least partition of the vertices of $\bar{\Gamma}$ into at most k cliques].

An ordering of the 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

$$A \trianglelefteq B$$

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).

Example

$\{3, 5, 6\} \trianglelefteq \{2, 3\}$, but $\{3, 4, 7\} \trianglelefteq \{3, 5, 6\}$.

An ordering of finite sequences of subsets of $\{1, \ldots, n\}$

Where $\mathcal{A} = (A_1, \ldots, A_t)$ and $\mathcal{B} = (B_1, \ldots, B_u)$ are finite sequences of subsets of $\{1, \ldots, n\}$, we define

$$\mathcal{A} \trianglelefteq \mathcal{B}$$

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 $\trianglelefteq$ on subsets of $\{1, \ldots, n\}$.

Example

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

The least vertex k-colouring

- Now let Γ be a graph with (non-empty) vertex set $V(\Gamma) = \{1, \ldots, n\}$, such that $V(\Gamma)$ can be partitioned into at most k cliques of Γ, for a given positive integer k.
- Then there is a unique least ordered such partition $(C_1, \ldots, C_m)$ with respect to $\trianglelefteq$.
- We now consider some properties of this least ordered partition, which give us very useful constraints on partial solutions in a backtrack search.

Constraints on partial solutions

Let $(C_1, \ldots, C_m)$ be the least ordered partition of $V(\Gamma)$ into cliques, with respect to $\trianglelefteq$, such that $m \leq k$. Let $\Gamma_1 := \Gamma$, let $G_1 := \mathrm{Aut}(\Gamma)$, and for $i = 2, \ldots, m$, let Γ_i be the subgraph of Γ_{i-1} induced on $V(\Gamma_{i-1}) \setminus C_{i-1}$ and let G_i be the (setwise) stabilizer in G_{i-1} of C_{i-1}. Then:

Constraints on partial solutions

Let $(C_1, \ldots, C_m)$ be the least ordered partition of $V(\Gamma)$ into cliques, with respect to $\trianglelefteq$, such that $m \leq k$. Let $\Gamma_1 := \Gamma$, let $G_1 := \mathrm{Aut}(\Gamma)$, and for $i = 2, \ldots, m$, let Γ_i be the subgraph of Γ_{i-1} induced on $V(\Gamma_{i-1}) \setminus C_{i-1}$ and let G_i be the (setwise) stabilizer in G_{i-1} of C_{i-1}. Then:

- $C_1 \triangleleft \cdots \triangleleft C_m = V(\Gamma_m)$;

Constraints on partial solutions

Let $(C_1, \ldots, C_m)$ be the least ordered partition of $V(\Gamma)$ into cliques, with respect to $\trianglelefteq$, such that $m \leq k$. Let $\Gamma_1 := \Gamma$, let $G_1 := \mathrm{Aut}(\Gamma)$, and for $i = 2, \ldots, m$, let Γ_i be the subgraph of Γ_{i-1} induced on $V(\Gamma_{i-1}) \setminus C_{i-1}$ and let G_i be the (setwise) stabilizer in G_{i-1} of C_{i-1}. Then:

- $C_1 \trianglelefteq \cdots \trianglelefteq C_m = V(\Gamma_m)$;
- for $i = 1, \ldots, m$, C_i is a non-empty maximal clique of Γ_i, with $(k - i + 1)|C_i| \geq |V(\Gamma_i)|$, and if $(k - i + 1)|C_i| = |V(\Gamma_i)|$ then $m = k$ and $\{C_i, C_{i+1}, \ldots, C_m\}$ is a partition of $V(\Gamma_i)$ into maximal cliques of Γ_i, each of size $|C_i|$;

Constraints on partial solutions

Let $(C_1, \ldots, C_m)$ be the least ordered partition of $V(\Gamma)$ into cliques, with respect to $\trianglelefteq$, such that $m \leq k$. Let $\Gamma_1 := \Gamma$, let $G_1 := \mathrm{Aut}(\Gamma)$, and for $i = 2, \ldots, m$, let Γ_i be the subgraph of Γ_{i-1} induced on $V(\Gamma_{i-1}) \setminus C_{i-1}$ and let G_i be the (setwise) stabilizer in G_{i-1} of C_{i-1}. Then:

- $C_1 \trianglelefteq \cdots \trianglelefteq C_m = V(\Gamma_m)$;
- for $i = 1, \ldots, m$, C_i is a non-empty maximal clique of Γ_i, with $(k - i + 1)|C_i| \geq |V(\Gamma_i)|$, and if $(k - i + 1)|C_i| = |V(\Gamma_i)|$ then $m = k$ and $\{C_i, C_{i+1}, \ldots, C_m\}$ is a partition of $V(\Gamma_i)$ into maximal cliques of Γ_i, each of size $|C_i|$;
- for $i = 1, \ldots, m$, C_i is the lexicographically least set in its G_i-orbit (as determined by Steve Linton's `SmallestImageSet` function included in GRAPE);

Constraints on partial solutions

Let $(C_1, \ldots, C_m)$ be the least ordered partition of $V(\Gamma)$ into cliques, with respect to $\trianglelefteq$, such that $m \leq k$. Let $\Gamma_1 := \Gamma$, let $G_1 := \mathrm{Aut}(\Gamma)$, and for $i = 2, \ldots, m$, let Γ_i be the subgraph of Γ_{i-1} induced on $V(\Gamma_{i-1}) \setminus C_{i-1}$ and let G_i be the (setwise) stabilizer in G_{i-1} of C_{i-1}. Then:

- $C_1 \triangleleft \cdots \triangleleft C_m = V(\Gamma_m)$;
- for $i = 1, \ldots, m$, C_i is a non-empty maximal clique of Γ_i, with $(k - i + 1)|C_i| \geq |V(\Gamma_i)|$, and if $(k - i + 1)|C_i| = |V(\Gamma_i)|$ then $m = k$ and $\{C_i, C_{i+1}, \ldots, C_m\}$ is a partition of $V(\Gamma_i)$ into maximal cliques of Γ_i, each of size $|C_i|$;
- for $i = 1, \ldots, m$, C_i is the lexicographically least set in its G_i-orbit (as determined by Steve Linton's `SmallestImageSet` function included in GRAPE);
- for $i = 2, \ldots, m$, where $C_i^* := C_i \cup \{v \in C_{i-1} : C_i \subseteq \mathrm{Adjacency}_{\Gamma_{i-1}}(v)\}$, we have $|C_i^*| \leq |C_{i-1}|$, and if $|C_i^*| = |C_{i-1}|$, then the lexicographically least set in the G_{i-1}-orbit of C_i^* is not lexicographically less than C_{i-1}.

McLaughlin graph example

```
gap> G:=AllPrimitiveGroups(DegreeOperation,275);
[ McL, McL:2, A(275), S(275) ]
gap> M:=EdgeOrbitsGraph(G[2],[1,2]);;
gap> GlobalParameters(M);
[ [ 0, 0, 112 ], [ 1, 30, 81 ], [ 56, 56, 0 ] ]
gap> CliqueNumber(M);
5
gap> CliqueNumber(ComplementGraph(M));
22
```

McLaughlin graph example cont'd

```
gap> M1:=DistanceSetInduced(M,1,1);;
gap> GlobalParameters(M1);
[ [ 0, 0, 30 ], [ 1, 2, 27 ], [ 10, 20, 0 ] ]
gap> ChromaticNumber(M1);
8
gap> M2:=DistanceSetInduced(M,2,1);;
gap> GlobalParameters(M2);
[ [ 0, 0, 56 ], [ 1, 10, 45 ], [ 24, 32, 0 ] ]
gap> ChromaticNumber(M2);
10
gap> Collected(VertexColouring(M2,10));
[ [ 1, 21 ], [ 2, 21 ], [ 3, 19 ], [ 4, 19 ], [ 5, 15 ],
  [ 6, 12 ], [ 7, 16 ], [ 8, 15 ], [ 9, 12 ], [ 10, 12 ] ]
```

A challenge problem

- The previous computation (which took under 30 seconds on a desktop PC) shows that that the first subconstituent of the McLaughlin graph M has chromatic number 8 (previously known), and the second subconstituent has chromatic number 10 (previously unknown?).

- Hence $\chi(M) \leq 18$.

A challenge problem

- The previous computation (which took under 30 seconds on a desktop PC) shows that that the first subconstituent of the McLaughlin graph M has chromatic number 8 (previously known), and the second subconstituent has chromatic number 10 (previously unknown?).
- Hence $\chi(M) \leq 18$.
- In fact, using an eigenvalue-based lower bound and "ant colony optimization" for establishing an upper bound, I can show that $15 \leq \chi(M) \leq 16$.
- The problem is to either find a vertex 15-colouring of M or to show there is no such colouring.

Possible future work

- Use of advanced heuristics (such as ant colony optimization) to find vertex k-colourings when they exist.

  See: R.M.R. Lewis, *A Guide to Graph Colouring. Algorithms and Applications*, Springer, 2016. Downloadable programs available from `http://www.rhydlewis.eu/`

- Parallelization of my clique and colouring algorithms using HPC-GAP.

- Implementing (certain parts) of the clique and colouring algorithms in python or C for efficiency and better integration with *nauty/traces* and the available heuristic colouring programs.

Getting GAP and GRAPE

- The latest version of the GRAPE package (4.8) is included with the latest version of GAP, which is freely available at:

  `http://www.gap-system.org`

  and runs under UNIX/Linux, Windows and macOS.

- Please refer to their extensive online documentation for precise details and many examples of their use.

Thanks for listening.

I hope you find GRAPE useful for cliques and colourings and more!