Computing with graphs and groups

Leonard H. Soicher

Abstract

In this chapter we discuss the computational study of graphs with groups acting on them, and demonstrate various ways in which computational group theoretical methods are used in the study of graphs and groups. We place particular emphasis on the ideas and methods behind the GRAPE and nauty computer packages.

1 Introduction

The study of graphs with groups acting on them is the study of $\mathcal{G}$-graphs. A *$\mathcal{G}$-graph* $G = (G, \mathcal{G}, \phi)$ consists of a (di)graph G, a group $\mathcal{G}$, and a homomorphism $\phi : \mathcal{G} \to \mathrm{Aut}(G)$. (The study of graphs without groups acting on them is just the special case of $\mathcal{G}$-graphs for which $\mathcal{G}$ is the trivial group.) $\mathcal{G}$-graphs arise naturally in many areas, most obviously in the study of graphs related to permutation groups, but also in the study of finite geometries and designs.

An example of a $\mathcal{G}$-graph (where $\mathcal{G}$ is the symmetric group $\mathcal{S}_n$) is the *Johnson graph* $J(n,k)$, defined to be the graph whose vertex set consists of all k-subsets of $\{1,\ldots,n\}$, with vertex v adjacent to vertex w exactly when $|v \cap w| = k-1$. Now $J(n,k)$ is an $\mathcal{S}_n$-graph $(J(n,k), \mathcal{S}_n, \phi)$, where, if $v = \{i_1,\ldots,i_k\}$ is a vertex of $J(n,k)$ and $x \in \mathcal{S}_n$, then $v\phi(x) = \{i_1x,\ldots,i_kx\}$.

The GAP [13] package GRAPE [38] is designed for computing with $\mathcal{G}$-graphs, and in particular, makes use of the group $\mathcal{G}$ acting on a $\mathcal{G}$-graph G in order to construct, store, and compute with G efficiently. The nauty [26] package contains the most powerful programs available for computing automorphism groups of graphs and testing graph isomorphism; nauty is available as a standalone package or may be used from within GRAPE or Magma [27].

There are many computer systems that are useful for studying graphs and groups which we do not discuss here. These include the algebra system Magma, which supports computation with graphs and includes the nauty package, the CoCo package [11] for computing with coherent configurations (see [5, 12]), and W. Kocay's package Groups & Graphs [21] (for Macintosh and Windows), which includes a graphical user interface to compute with graphs, geometric configurations, combinatorial designs, and their automorphism groups. Also worth mentioning are the algorithms of S. Rees and the author for computing fundamental groups and covers of combinatorial cell complexes, and which have been implemented for simplicial complexes [32].

This chapter is organized as follows. We first describe basic algorithms for permutation groups that are used in the study of $\mathcal{G}$-graphs. We then discuss the efficient storage and construction of $\mathcal{G}$-graphs, and how to use a modified form of breadth-first search to determine efficiently many properties of a $\mathcal{G}$-graph. Following that, we concentrate on the methods used by nauty. Next, we discuss the application of computational methods to the study of vertex-transitive graphs. Then, after a brief discussion of the coset enumeration procedure, we describe some applications of coset enumeration to the study of symmetric graphs.

Throughout this chapter, a digraph is allowed to have loops (but no multiple arcs), and by the *adjacency set* $G(v)$ of a vertex v in a (di)graph G we mean the set of all vertices w such that (v,w) is an arc of G.

2 Permutation group algorithms

In this section, we describe some basic permutation group algorithms that are important in the study of $\mathcal{G}$-graphs; more detailed elementary expositions can be found in [4, 9], and a comprehensive treatment of the state of the art in permutation group algorithms is the book by Seress [33]. Throughout this section, V is a finite set of size n, and we are computing with the permutation group $\mathcal{G}$ given by a generating set X of permutations of V. The image of $v \in V$ under the permutation x of V is denoted by vx.

2.1 Orbits and Schreier vectors

Let $v \in V$. The *orbit* of v under $\mathcal{G}$ is the set $v\mathcal{G} = \{vg : g \in \mathcal{G}\}$. The calculation of all of the orbits $\{v\mathcal{G} : v \in V\}$ of $\mathcal{G} = \langle X \rangle$ is equivalent to determining the connected components of the 'Schreier graph' with respect to X for $\mathcal{G}$ acting on V. This *Schreier graph* $S(V, X)$ is a digraph with vertex set V, with (v, w) an arc exactly when $vx = w$, for some $x \in X$. A set of rooted directed spanning trees of the connected components of $S(V, X)$ describes a set R of orbit representatives (the roots) and their $\mathcal{G}$-orbits (the vertices in each tree). These trees are called *Schreier trees*, and we always direct a Schreier tree so that the direction of each arc is away from the root. A set of Schreier trees can be compactly encoded using a 'Schreier vector', usually implemented as an array. A *Schreier vector* $s = s(V, X)$ for a set R of orbit representatives and a set of Schreier trees in $S(V, X)$ that they root, is a map

$$s : V \to \{0\} \cup X,$$

such that $s(v) = 0$ if $v \in R$, and $s(v) = x \in X$ means that (vx^{-1}, v) is an arc of one of the Schreier trees (so vx^{-1} is the 'parent' of v in that tree).

The computation of a single orbit $r\mathcal{G}$ and the definition of the Schreier vector entries $s(v)$ for the v in this orbit is usually done using a breadth-first search of the connected component containing r. We first define $s(r) = 0$, and perform the breadth-first search outwards from r, finding the adjacency set of a vertex u by applying each element of X to u. Whenever we encounter a new vertex v in our breadth-first search, say by applying x to u, we define $s(v) = x$.

If we are given a Schreier vector $s = s(V, X)$ and a point $v \in V$, we can then determine a pair $(r, \mathbf{w})$ for which r is the root of the Schreier tree containing v, and $\mathbf{w}$ is a word in X such that $r\mathbf{w} = v$. This calculation proceeds as follows:

```
1. $\mathbf{w}$ := emptyword; $r := v$; $x := s(r)$;
2. while $x \neq 0$ do $\mathbf{w} := x\mathbf{w}$; $r := rx^{-1}$; $x := s(r)$; end_do;
```

On completion, $(r, \mathbf{w})$ is the required pair.

Some brief remarks are in order. We do not keep evaluating the inverses of elements of X, since we either make X inverse-closed, and so can access directly the inverse of a generator in X, or we compute rx^{-1} by tracing r through its cycle in x until we find the point mapped to r by x. Finally, note that our definition of a Schreier vector (for **all** the orbits of a group at once) is not the usual one.

2.2 Bases and strong generating sets

A sequence $B = (b_1, \ldots, b_m)$ of elements of V is a *base* for $\mathcal{G}$ if the (pointwise) stabilizer $\mathcal{G}_B$ of B is the trivial group. A base B defines a chain

$$\mathcal{G} = \mathcal{G}^{(1)} \geq \mathcal{G}^{(2)} \geq \cdots \geq \mathcal{G}^{(m)} \geq \mathcal{G}^{(m+1)} = \{1\}$$

of subgroups of $\mathcal{G}$, such that $\mathcal{G}^{(i)}$ is the (pointwise) stabilizer of $(b_1, \ldots, b_{i-1})$. We observe that

$$|\mathcal{G}| = \prod_{i=1}^{m} |\mathcal{G}^{(i)} : \mathcal{G}^{(i+1)}| \quad \text{and} \quad |\mathcal{G}^{(i)} : \mathcal{G}^{(i+1)}| = |b_i \mathcal{G}^{(i)}|.$$

A *strong generating set* for $\mathcal{G}$, relative to B, is a generating set Y for $\mathcal{G}$ with the property that

$$\langle Y \cap \mathcal{G}^{(i)} \rangle = \mathcal{G}^{(i)}, \quad \text{for } i = 1, \ldots, m+1.$$

Given a strong generating set relative to B, the orbits $b_i \mathcal{G}^{(i)}$ can easily be computed, and we can thus obtain the order of $\mathcal{G}$. A base and associated strong generating set are required by most advanced permutation group algorithms, and are very useful in $\mathcal{G}$-graph computations.

As well as introducing the fundamental concepts of base and strong generating set, Sims [34] devised an algorithm, now called the *Schreier-Sims algorithm*, to construct a base and associated strong generating set for the permutation group $\mathcal{G} = \langle X \rangle$. Modern variants of this algorithm and others, implemented in GAP and MAGMA, can be used to compute bases and strong generating sets for many permutation groups of degree 10^5 or more (see [33, Ch. 5, 8]). It is shown in [33, Ch. 5] that, given a base B for $\mathcal{G}$, a strong generating set for $\mathcal{G}$ can be computed in $O(n|B|^2|X|(\log|G|)^3)$ time. Of course, one could take B to be a sequence of length n of all of the elements of V, but in practice one often knows or can compute a much shorter base.

Some permutation group algorithms, such as those for determining the $\mathcal{G}$-stabilizer of a set of points of V or determining the centralizer of a subgroup of $\mathcal{G}$, currently use backtrack search, and are not polynomial-time algorithms. Modern implementations of certain permutation group backtrack algorithms can, however, be quite efficient in practice, even when V has size 10^5 or more (see [22]). It would be extremely interesting if polynomial-time algorithms could be found for set-stabilizer or subgroup-centralizer, especially since the problem of determining graph isomorphism is polynomial-time reducible to each of these tasks (see [33, C. 3]).

3 Storing and accessing a $\mathcal{G}$-graph

We now consider how GRAPE stores a $\mathcal{G}$-graph $G = (G, \mathcal{G}, \phi)$ in a compact way that also enables the efficient recovery of basic information about the (di)graph G. In this implementation, we store $\phi(\mathcal{G})$ by a generating set, so we now assume that $\mathcal{G} = \phi(\mathcal{G}) \leq \mathrm{Aut}(G)$ and let $\mathcal{G}$ be given by a generating set X.

Let $V_1, \ldots, V_k$ be the orbits of $\mathcal{G}$ on the set V of vertices of G, with respective representatives $v_1, \ldots, v_k$. We store G using a record data structure, containing:

1. the permutation generators X for $\mathcal{G}$;
2. the list $v_1, \ldots, v_k$ of orbit representatives;
3. the list $G(v_1), \ldots, G(v_k)$ of the adjacency sets of these $\mathcal{G}$-orbit representatives;
4. a Schreier vector $s(V, X)$.

Note that even if $\mathcal{G}$ is trivial, then the above method of storing G is not significantly worse than a represention by a list of adjacency sets of the vertices.

We now describe how to use the Schreier vector $s = s(V, X)$ to determine whether, for vertices v and w, (v, w) is an arc of G, and to calculate the adjacency set $G(v)$ of v. We first use s to determine a pair $(v_i, \mathbf{w})$ for which v_i is the orbit representative of $v\mathcal{G}$ and $\mathbf{w}$ is a word in X mapping v_i to v. Then (v, w) is an arc of G if and only if $w\mathbf{w}^{-1} \in G(v_i)$, and we have $G(v) = G(v_i)\mathbf{w}$.

Our method of storing a $\mathcal{G}$-graph is space-efficient at the cost of time for the recovery of the adjacency sets of vertices. It is a good idea for the Schreier trees encoded by s to be as shallow as is reasonably possible – hence the use of breadth-first search. A thorough discussion on computing shallow Schreier trees is given in [33, Ch. 4]. We do remark, however, that our method of storing $\mathcal{G}$-graphs can often save time when constructing a graph, since we need only calculate the adjacency sets of orbit representatives for $\mathcal{G}$ on V.

We discuss one final point concerning the GRAPE data structure for a $\mathcal{G}$-graph G, which has turned out to be extremely useful when doing 'real-life' calculations. Internally, the vertices of G are represented by the integers $1, 2, \ldots, |V(G)|$, but also each vertex has a 'name', which can be an object of any GAP type. When constructing a new graph (possibly from another graph), the vertices of this new graph are numbered $1, 2, \ldots$, but their names are chosen to reflect the mathematical nature of the vertices. Details of this naming for specific functions can be found in the GRAPE manual. (The use of this naming, which is implemented by a list, was suggested by P. J. Cameron.)

4 Constructing $\mathcal{G}$-graphs

Although not always necessary, we assume from here on that if we are computing with a $\mathcal{G}$-graph $G = (G, \mathcal{G}, \phi)$, then we have a base and associated strong generating set for $\phi(\mathcal{G})$ acting on $V = V(G)$. This allows, for example, the efficient calculation of the $\phi(\mathcal{G})$-stabilizer of an arbitrary point in V (see [33, Ch. 5]). We remark that in many constructions of $\mathcal{G}$-graphs, our knowledge about $\mathcal{G}$ often allows us to compute this base and strong generating set much more efficiently than otherwise. For example, we often know the order of $\phi(\mathcal{G})$, and this allows the use of a very fast randomized algorithm to compute a base and strong generating set (see [33, Ch. 8]), often applicable to groups of degree 10^6 or more. We also remark that the nauty package [26] (see Section 6) outputs the automorphism group of a graph as a base and associated strong generating set for that group.

GRAPE has many functions to construct $\mathcal{G}$-graphs, including Cayley graphs, orbital (di)graphs, induced subgraphs, and quotient graphs. However, the most useful and general way of constructing a $\mathcal{G}$-graph in GRAPE is to use the function `Graph`, which behaves as follows. The input is a group $\mathcal{G}$ (which may or may not be a permutation group), a finite set V on which $\mathcal{G}$ acts (the action can be one of the standard actions in GAP or may be one supplied by the user), and a $\mathcal{G}$-invariant relation *rel* on V (given as a function of two vertices v and w and which returns TRUE or FALSE according to whether or not (v,w) is in the relation). The output is the $\mathcal{G}$-graph G with vertex set V, where (v,w) an arc if and only if $rel(v,w)$.

The first step is to compute the orbits of $\mathcal{G}$ acting on V, and the associated Schreier vector. Then, for each orbit-representative r, we need to determine the $v \in V$ such that $rel(r,v)$. In fact, we need check $rel(r,s)$ only for those s in a set of orbit-representatives of the orbits on V of the $\mathcal{G}$-stabilizer $\mathcal{H}$ of r, since $rel(r,t)$ for each t in $s\mathcal{H}$ if and only if $rel(r,s)$.

5 $\mathcal{G}$-breadth-first search in a $\mathcal{G}$-graph

In this section, for ease of exposition, we assume that our $\mathcal{G}$-graph $G=(V,E)$ is a simple graph, and that $\mathcal{G} \leq \mathrm{Aut}(G)$. We define $G_i(v)$ to be the set of all vertices at distance i from a vertex v of G.

Many properties of G can be determined by (possibly repeated) application of breadth-first search. These include finding the connected components, diameter, and girth, as well as determining various regularity properties, such as whether G is distance-regular. We describe here a version of breadth-first search, which we call $\mathcal{G}$-breadth-first search, which takes into account $\mathcal{G} \leq \mathrm{Aut}(G)$.

Let $v \in V$, and let $\mathcal{H} = \mathcal{G}_v$ be the stabilizer in $\mathcal{G}$ of v. The key observation is that if w is a vertex at distance i from v, then each vertex in the orbit $w\mathcal{H}$ is at distance i from v.

Suppose that $V_1 = \{v\}, V_2, \ldots, V_k$ are the orbits of $\mathcal{H}$ on V, with respective representatives $v_1 = v, v_2, \ldots, v_k$, and let $R = \{v_1, \ldots, v_k\}$. In a $\mathcal{G}$-breadth-first search from v, we determine the sets $R_0, R_1, \ldots$ of $\mathcal{H}$-orbit representatives, where

$$R_i = G_i(v) \cap R.$$

Given R_i, we then have $G_i(v)$ as the union of the (already computed) orbits represented by the elements of R_i.

Clearly $R_0 = \{v\}$. The basic step is to obtain R_{i+1} from R_i. We actually do more, in order to obtain more information from our $\mathcal{G}$-breadth-first search. We start by setting $R_{i+1} := \{\}$, and then for each $r \in R_i$, do the following:

1. determine $C := G(r) \cap G_{i-1}(v)$, $A := G(r) \cap G_i(v)$, and $B := G(r) - (C \cup A)$;
2. add to R_{i+1} the representatives of the $\mathcal{H}$-orbits that intersect B nontrivially;
3. for later use, store $c_i(v,r) := |C|$, $a_i(v,r) := |A|$, and $b_i(v,r) := |B|$.

The $\mathcal{G}$-breadth-first search stops with $R_0, \ldots, R_m$ when R_{m+1} is empty, but R_m is not. If required, it is trivial to recover $G_0(v), \ldots, G_m(v)$, the union of these sets being the vertices in the connected component containing v. Also note that m is the greatest distance $d(v)$ from v to any vertex in the connected component of v. Moreover, if G is connected, then its diameter $\mathrm{diam}(G)$ is the maximum value of $d(w)$, as w ranges over a set of representatives of the $\mathcal{G}$-orbits on V.

Let $g(v)$ denote the length of a shortest cycle containing v, if v is on some cycle of G, and let $g(v) = \infty$ otherwise. The numbers $c_i(v,r)$ and $a_i(v,r)$ computed above (for each $i = 1, \ldots, m$ and each $r \in R_i$) can be used to determine $g(v)$, as follows. Let t be the least value of $i \in \{1, \ldots, m\}$ such that $c_i(v,r) \geq 2$ (for some $r \in R_i$) or $a_i(v,r) \geq 1$ (for some $r \in R_i$), if such an i exists. If no such i exists then $g(v) = \infty$; otherwise, if $c_t(v,r) \geq 2$ for some $r \in R_i$, then $g(v) = 2t$; if not then $g(v) = 2t + 1$. Note that the girth of G is the minimum value of $g(w)$, as w ranges over a set of representatives of the $\mathcal{G}$-orbits on V (a girth of ∞ means that G has no cycles).

Now define $c_0(v,v) = 0$, $a_0(v,v) = 0$, and $b_0(v,v) = |G(v)|$. Let $0 \leq i \leq m$ and $r \in R_i$. If $c_i(v,r)$ depends only on i and v (and not the $\mathcal{H}$-orbit representative r), then we denote this quantity by $c_i(v)$ and call it a *local parameter* of G. Similarly, if $a_i(v,r)$ and $b_i(v,r)$ do not depend on r, then these too are called local parameters, and are respectively denoted $a_i(v)$ and $b_i(v)$. Such local parameters, if and when they exist, are used in the determination of various regularity properties of G, the strongest of which is distance-regularity. Indeed, the graph G is distance-regular if and only if G is connected, $d(v)$ is the same for all vertices v (so each $d(v) = \mathrm{diam}(G)$), and for each vertex v and $i = 0, \ldots, \mathrm{diam}(G)$, all local parameters $c_i(v)$, $a_i(v)$, and $b_i(v)$ exist and do not depend on v, thus giving the parameters c_i, a_i, and

b_i of a distance-regular graph; see Chapter **??**. Of course, we need to check these conditions only for those vertices v in a set of orbit-representatives of $\mathcal{G}$ on V.

Finally, we remark that $\mathcal{G}$ acts distance-transitively on G (see Chapter **??**) if and only if G is connected, $\mathcal{G}$ has just one orbit on V, and, for some (and hence all) vertices v, the number of $\mathcal{G}_v$-orbits on V is $\mathrm{diam}(G) + 1$.

$\mathcal{G}$-breadth-first search is efficiently implemented in GRAPE in order to determine connected components, diameter, girth, and regularity properties of $\mathcal{G}$-graphs. Although not discussed here, GRAPE includes many other functions for computing with $\mathcal{G}$-graphs, such as backtrack search functions for classifying complete subgraphs of given weight-sum in a vertex-weighted $\mathcal{G}$-graph and for classifying partial linear spaces with given point graph and parameters.

6 Automorphism groups and graph isomorphism

Automorphism groups of graphs are discussed in Chapter **??**. The most advanced algorithms and programs for computing automorphism groups of graphs and for testing graph isomorphism are those of McKay, freely available as part of his nauty package [26], and useful for graphs with up to about 10^4 vertices (nauty also works with digraphs, although we do not consider them in this section). We shall briefly describe McKay's method of 'partition backtrack', which has been very influential in computational group theory. Indeed, partition-based backtrack methods are now used in the most advanced available algorithms for calculating set-stabilizers, centralizers, and normalizers in permutation groups (see [33, Ch. 9]).

Let $G = (V, E)$ be a graph, with $V = \{1, \dots, n\}$, and let π be an ordered partition of V. Thus π is a sequence $(V_1, \dots, V_k)$ of distinct subsets of V, such that $\{V_1, \dots, V_k\}$ is a partition of V. The elements of V are called *cells*. (One can think of (G, π) as being a (not necessarily properly) vertex-coloured graph, with vertices v and w having the same colour if and only if v and w belong to to the same cell of π.) Define the ordered partition $c(\pi)$ of V to be

$$(\{1, \dots, |V_1|\}, \{|V_1| + 1, \dots, |V_1| + |V_2|\}, \dots, \{n - |V_k| + 1, \dots, n\}).$$

For x a permutation of V, define Gx to be the graph (V, Ex), where $Ex = \{\{vx, wx\} : \{v, w\} \in E\}$, and define $\pi x = (V_1x, \dots, V_kx)$, where $V_ix = \{vx :$

$v \in V_i\}$. The *automorphism group* $\mathrm{Aut}(G, \pi)$ of (G, π) is the group of all permutations x of V for which $(Gx, \pi x) = (G, \pi)$. Thus, when $\pi = (V)$, the automorphism group of (G, π) is just $\mathrm{Aut}(G)$.

The main functions of the nauty package are to determine $\mathrm{Aut}(G, \pi)$ (in the form of a base and associated strong generating set), and in the process, to compute the image of (G, π) under a canonical labelling map (described below), which is used for isomorphism testing.

A *canonical labelling map* is a function $\mathcal{C}$ such that, for each graph G with vertex set $V = \{1, \ldots, n\}$, each ordered partition π of V, and each permutation x of V, we have:

- $\mathcal{C}(G, \pi) = Gy$ for some permutation y of V such that $\pi y = c(\pi)$;
- $\mathcal{C}(Gx, \pi x) = \mathcal{C}(G, \pi)$.

The importance of a canonical labelling map $\mathcal{C}$ is as follows. Suppose that G_1 and G_2 are graphs on the same vertex-set $V = \{1, \ldots, n\}$, and that π_1 and π_2 are ordered partitions of V with $c(\pi_1) = c(\pi_2)$. Then there is a permutation y of V such that $(G_1 y, \pi_1 y) = (G_2, \pi_2)$ if and only if $\mathcal{C}(G_1, \pi_1) = \mathcal{C}(G_2, \pi_2)$. In particular, G_1 is isomorphic to G_2 if and only if $\mathcal{C}(G_1, (V)) = \mathcal{C}(G_2, (V))$.

6.1 Partition backtrack

As before, G is a graph with vertex set $V = \{1, \ldots, n\}$ and $\pi = (V_1, \ldots, V_k)$ is an ordered partition of V. The cells of π of size 1 are called *singletons*, and a *discrete* partition is one in which all cells are singletons.

For each $v \in V$, let $u(v, \pi)$ denote the index of the cell in π containing v; in other words, $u(v, \pi) = i$ means that $v \in V_i$. We say that an ordered partition ν of V is a *refinement* of π if:

- every cell of ν is contained in some cell of π;
- for each $v, w \in V$, if $u(\pi, v) < u(\pi, w)$, then $u(\nu, v) < u(\nu, w)$.

A *refinement process* is a function $\mathcal{R}$, such that for each graph G with vertex set V, each ordered partition π of V, and each permutation x of V,

$$\mathcal{R}(G, \pi) \text{ is a refinement of } \pi, \text{ and } \mathcal{R}(G, \pi)x = \mathcal{R}(Gx, \pi x).$$

In particular, if $x \in \mathrm{Aut}(G)$, then $\mathcal{R}(G, \pi)x = \mathcal{R}(G, \pi x)$, and so x maps the ordered partition π to ν only if x maps the refinement $\mathcal{R}(G, \pi)$ to $\mathcal{R}(G, \nu)$.

In particular, if $c(\mathcal{R}(G,\pi)) \neq c(\mathcal{R}(G,\nu))$, then there is no element of $\mathrm{Aut}(G)$ that maps π to ν. Note that, if $\mathcal{R}(G,\pi)$ is discrete, then $\mathrm{Aut}(G,\pi)$ is trivial.

An ordered partition $\pi = (V_1,\ldots,V_k)$ of V is *G-equitable* if there are constants d_{ij} $(1 \leq i,j \leq k)$ such that, for each vertex $v \in V_i$, $|G(v) \cap V_j| = d_{ij}$. The default refinement process $\mathcal{R}$ used by nauty (for simple graphs) maps (G,π) to an ordered G-equitable partition ν, such that ν is a refinement of π, and, up to the order of its cells, is the coarsest equitable partition that is a refinement of π. More precise details can be found in [25].

We now outline (roughly) how the nauty procedure finds $\mathrm{Aut}(G,\pi)$. For much more thorough details, together with how a canonical labelling map is found, see [25] (see also [20, 24, 28]).

We let $\mathcal{R}$ be the default refinement process used by nauty. The nauty procedure proceeds by a depth-first search in a search tree whose nodes are ordered G-equitable partitions of V; the root is $\mathcal{R}(G,\pi)$ and the leaves are discrete partitions. We call the first leaf found ζ. Any other leaf λ may give rise to a new automorphism of (G,π), and we check to see whether the unique permutation of V mapping ζ to λ is in $\mathrm{Aut}(G,\pi)$. A non-leaf ν in the search tree is not discrete, and we obtain its 'children' as follows. First, we choose a non-singleton cell C of ν according to some rule (we usually just take the first non-singleton cell). Then for each $v \in C$ (in ascending order of v), we *isolate* v – that is, we form the ordered partition $\nu \circ v$ obtained from ν by replacing C by $\{v\}, C - \{v\}$, and then we add $\mathcal{R}(G, \nu \circ v)$ as a 'child' of ν. The search tree is pruned in various ways, so as to avoid searching subtrees providing no new information (see [25]). The search is structured so as to provide a base and associated strong generating set for $\mathrm{Aut}(G,\pi)$. In determining the first leaf ζ, G-vertices $w_1,\ldots,w_m$, say, are isolated (w_i at depth i) for the formation of the ancestors of ζ (other than the root) and the formation of ζ itself. It is not difficult to see that $(w_1,\ldots,w_m)$ is a base for $\mathrm{Aut}(G,\pi)$; nauty computes a strong generating set relative to this base.

7 Computing with vertex-transitive graphs

Recall that a (di)graph G is *vertex-transitive* if $\mathrm{Aut}(G)$ acts transitively on $V(G)$. The class of vertex-transitive graphs includes Cayley graphs and symmetric graphs (studied in Chapter **??**), which further includes the class of distance-transitive graphs (studied in Chapter **??**). In this section we consider the computational study of a $\mathcal{G}$-graph G, where $\mathcal{G} \leq \mathrm{Aut}(G)$ acts transitively on $V(G)$. Note that the GRAPE data structure for storing a $\mathcal{G}$-graph is especially compact in this case.

7.1 Collapsed adjacency matrices

Let $\mathcal{G}$ be a transitive permutation group on a finite set V. Then $\mathcal{G}$ has a natural action on $V \times V$, defined by $(v, w)x = (vx, wx)$. The orbits of this action are called *orbitals*, and the orbits of the stabilizer $\mathcal{G}_v$ of a point $v \in V$ are called *suborbits*. It is well known that the orbitals for $\mathcal{G}$ are in one-to-one correspondence with these suborbits: this correspondence maps an orbital E to the suborbit $\{w : (v, w) \in E\}$. The *orbital digraph* for $\mathcal{G}$ associated with an orbital E is simply the digraph (V, E). If the orbital E is non-diagonal and self-paired (see Chapter **??**) then we associate the *orbital graph* $(V, \{\{v, w\} : (v, w) \in E\})$ with E.

Let $v_1 \in V$, and suppose that $V_1 = \{v_1\}, V_2, \ldots, V_k$ is an ordering of the orbits of $\mathcal{G}_{v_1}$, with respective representatives $v_1, v_2, \ldots, v_k$; k is the *rank* of $\mathcal{G}$. Let $G = (V, E)$ be a (di)graph on which $\mathcal{G}$ acts vertex-transitively, so that E is a union of orbitals and $G(v)$ is the union of the corresponding suborbits contained in $\{V_1, \ldots, V_k\}$. For $i, j = 1, \ldots, k$, define

$$a_{ij} = |G(v_i) \cap V_j\}|.$$

Note that a_{ij} does not depend on the choice v_i of suborbit representative, and it can easily be computed using the GRAPE data structure for the $\mathcal{G}$-graph G (in practice, for $|V|$ up to about 10^6). The $k \times k$ integer matrix $\mathbf{A} = (a_{ij})$ is the *collapsed adjacency matrix* for G, with respect to $\mathcal{G}$ and the ordering of the suborbits. This matrix (which is extremely compact when k is small) contains at least as much information as that computed in a $\mathcal{G}$-breadth first search from v.

Since $\mathcal{G}$ acts vertex-transitively, the single collapsed adjacency matrix $\mathbf{A}$ for G can be used to determine whether G is (strongly) connected, and if so, what its diameter is; whether G is a simple graph, and if so what its girth is, whether G is distance-regular, and whether $\mathcal{G}$ acts distance-transitively on G. See [31] for more detailed information and applications of collapsed adjacency matrices, and also Chapter **??** for examples of 'collapsed adjacency diagrams'.

We remark that the collapsed adjacency matrices for the orbital digraphs for the transitive group $\mathcal{G}$ are useful in studying the coherent configuration associated with $\mathcal{G}$ (see [5, 12]), since, with respect to a fixed ordering $V_1, \ldots, V_k$

of the suborbits as above, the collapsed adjacency matrix for an orbital digraph (V, E) for $\mathcal{G}$ is the transpose of the intersection matrix (as defined in [5, Ch. 3]) corresponding to the orbital paired with E.

7.2 Distance-transitive graphs

Chapter **??** provides an overview of the state of the classification of distance-transitive graphs. Here we discuss the application of computing, which has been used in the discovery, analysis, and classification of certain distance-regular and distance-transitive graphs (see, for example, [37, 31, 23, 18]). We also remark that computing is used in the determination of feasible intersection arrays for possible distance-regular graphs (see, for example, [3]).

Suppose that $\mathcal{G}$ acts distance-transitively on a graph G. Then G must be an orbital graph for $\mathcal{G}$ of the smallest or second-smallest vertex-degree (see [18]). Furthermore, each orbital for $\mathcal{G}$ must be *self-paired*, which is equivalent to the property that the permutation character of $\mathcal{G}$ on V is the sum of distinct complex irreducible characters, each with Frobenius-Schur indicator $+1$ (see [2, p. 64]). For these, and other reasons, it makes sense to analyse the lower degree orbital graphs of permutation representations whose character is *multiplicity-free* – that is, the sum of distinct complex irreducible characters.

Computation has been applied extensively in the classification of the graphs on which a sporadic simple group or its automorphism group acts primitively and distance-transitively (see [23, 18]; the results of this classification are given in Chapter **??**). In the process of this classification, the primitive multiplicity-free permutation characters for these sporadic groups were also determined, and for most of the primitive permutation representations of sporadic groups, a collapsed adjacency matrix was computed for the orbital graph of least degree. This built on the work of Praeger and Soicher [31], where collapsed adjacency matrices were computed for the orbital digraphs for all permutation representations of rank at most 5 of the sporadic simple groups and their automorphism groups. The practical computational determination of permutation characters is described in some detail in Linton, Lux and Soicher [23], which also details randomized techniques for computing collapsed adjacency matrices for certain permutation representations of degree about 10^{11}, where it would be impossible to store explicit permutation generators. These techniques make use of graph algorithms as well as permutation group algorithms.

An ambitious project at Lehrstuhl D für Mathematik, RWTH, Aachen, involving T. Breuer, I. Höhler, and J. Müller, has since determined collapsed adjacency matrices for all orbital digraphs for all multiplicity-free permutation representations of the sporadic simple groups and their automorphism groups, and the results are published on the world-wide web [1] (although what they call 'collapsed adjacency matrices' we would call intersection matrices).

We remark that many permutation and matrix representations of finite simple groups and related groups can be downloaded from the online ATLAS of group representations [41]. These group representations are very useful for contructing related $\mathcal{G}$-graphs and collapsed adjacency matrices.

8 Coset enumeration

Coset enumeration is one of the oldest and most useful methods of computational group theory (see [30, 35, 14] and their references). For this chapter, we concentrate on what coset enumeration does, and on the application of coset enumeration and related procedures to problems in graph theory.

Let $\mathcal{G} = \langle X : R \rangle$ be a finitely presented group – that is, $\mathcal{G}$ is generated by the finite set X, subject (only) to the finite set R of relators, which are words in $X \cup X^{-1}$ that evaluate to the identity in $\mathcal{G}$ (where $X^{-1} = \{x^{-1} : x \in X\}$). The input to coset enumeration is (X, R, Y), where Y is a set of words in $X \cup X^{-1}$ that generates a subgroup $\mathcal{H}$ of $\mathcal{G}$.

The coset enumeration process attempts to construct a set V, with $1 \in V$, and a transitive permutation representation $\rho : \mathcal{G} \to \mathrm{Sym}(V)$, such that in this representation, $\mathcal{H}$ is the stabilizer in $\mathcal{G}$ of the point 1. Coset enumeration does this by using a trial-and-error process for constructing the permutations $\rho(X \cup X^{-1})$. The name *coset enumeration* comes from the fact that, if a coset enumeration is successful, there is a one-to-one correspondence between the elements of V and the cosets of $\mathcal{H}$ in $\mathcal{G}$, with 1 corresponding to $\mathcal{H}$.

If the index of $\mathcal{H}$ in $\mathcal{G}$ is infinite, then the coset enumeration process does not terminate; if it is finite, then the process terminates, but there can be no computable general bound (in terms of the size of the input and the putative index) on the time or store required for termination (see, for example, [30]).

There is an enormous amount of flexibility in the coset enumeration process, and many different approaches have been suggested and experimented with (see [30, 35, 14]). Depending on the presentation and the approach used, there can be huge variations in the time and store taken. Currently, the most advanced methods are due to G. Havas and C. Ramsay, and these methods are available in their ACE package [15], also available as a GAP package [16] and within the MAGMA system [27].

There are many useful variations on coset enumeration. For example, a 'modified Todd-Coxeter' enumeration gives us a presentation for $\mathcal{H}$, and the 'low-index subgroups procedure' determines (up to permutation isomorphism) all transitive representations of $\mathcal{G}$ up to some given degree k; see [30] for an excellent introduction to coset enumeration and its variations.

9 Coset enumeration in the study of symmetric graphs

A graph G is *symmetric* if $\mathrm{Aut}(G)$ acts transitively on both its vertices and arcs (ordered pairs of adjacent vertices). A subgroup $\mathcal{G}$ of $\mathrm{Aut}(G)$ acts *symmetrically* on G if $\mathcal{G}$ acts transitively on both the vertices and arcs of G.

One common way to study connected symmetric graphs with given properties is by determining the groups that act on them symmetrically, as quotients of universal completions of appropriate amalgams (see Chapter **??**, [19] for a useful overview, and [17] for a more general geometrical context). In this approach, we first use the given graph-theoretical properties to determine the possible amalgams of the form $\mathcal{A} = \{\mathcal{G}_v, \mathcal{G}_{\{v,w\}}\}$, where (v, w) is an arc in the putative graph on which $\mathcal{G}$ acts symmetrically. Since the universal completion $U(\mathcal{A})$ of such an amalgam of two groups (with neither a subgroup of the other) is infinite, we need to add further relations to $U(\mathcal{A})$ to obtain the finite groups of automorphisms we seek. Such extra relations could come from cycles in our graph or from the local graph structure.

Coset enumeration can then be used in an effort to determine the (hopefully finite) index of $\mathcal{H} = \mathcal{G}_v$ in $\mathcal{G}$, and to construct the representation of $\mathcal{G}$ acting on the set V of right cosets of $\mathcal{H}$ in $\mathcal{G}$. Given such a representation of $\mathcal{G}$ on V, we can reconstruct and study the graph G which may have the properties we seek (or we may have been able to prove theoretically that G has the required properties). This graph G is simply the orbital graph for $\mathcal{G}$ for which the orbital contains $(\mathcal{H}, \mathcal{H}g)$, where g is an element of $\mathcal{G}_{\{v,w\}} - \mathcal{H}$. Applications of this kind include [29], where the 4- and 5-arc-transitive connected cubic graphs of girth up to 11 and girth 13 are classified; see Chapter **??** for a discussion of s-arc transitivity, and Conder and Dobcsányi [6], where the connected symmetric cubic graphs on up to 768 vertices are determined using a powerful new low-index subgroups procedure [7], parallel computation, and coset enumeration. Of course, computational studies of this kind often lead to conjectures and theoretical results. For more applications of coset enumeration to the study of symmetric graphs, see Weiss [39] for a beautiful and natural characterization and construction of the sporadic simple group J_3, and also [40, 36, 10].

9.1 Graphs locally a given graph

We now give an example where additional transitivity assumptions and local stucture specification lead to an amalgam of three groups. We use the ATLAS notation [8] for group structures.

Let G and H be graphs. Then G is said to be *locally-H* if, for each vertex v of G, the induced subgraph on $G(v)$ is isomorphic to H (this situation is discussed briefly in Chapter **??**). Given a graph H, coset enumeration can sometimes be used effectively to study presentations that arise in the classification of the connected graphs G that are locally-H, for which $\mathrm{Aut}(G)$ acts transitively on the ordered triangles of G. These presentations come from applying the amalgam method to putative stabilizers of a vertex, incident edge, and triangle, contained in a fixed triangle of such a G.

A simple, but good, example of this application of coset enumeration is given in [10], where H is the incidence graph of the unique 2–(11,5,2) design, and where the ordered-triangle-transitive graphs G which are locally-H are classified. We discuss here the case where a vertex-stabilizer in the automorphism group of G is isomorphic to $\mathrm{PGL}_2(11)$. For such a graph G, it is shown that there is (essentially) only one possible amalgam $\mathcal{A} = \{\mathcal{X}, \mathcal{Y}, \mathcal{Z}\}$ of the $\mathrm{Aut}(G)$-stabilizers $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ of x, $\{x, y\}$, and $\{x, y, z\}$, contained in a triangle $\{x, y, z\}$ of G. In this amalgam, $\mathcal{X} \cong \mathrm{PGL}_2(11)$, $\mathcal{Y} \cong \mathcal{S}_5$, $\mathcal{Z} \cong (\mathcal{A}_4 \times 3):2$, $\mathcal{X} \cap \mathcal{Y} \cong \mathcal{A}_5$, $\mathcal{X} \cap \mathcal{Z} \cong \mathcal{S}_4$, and $\mathcal{Y} \cap \mathcal{Z} \cong \mathcal{S}_4$. Then $\mathrm{Aut}(G)$ is a homomorphic image of $U(\mathcal{A})$, which has the following presentation, determined in [10]:

$$\begin{aligned}\langle a, b, c, d, e : a^3 = b^2 = c^2 = d^2 = (ab)^3 = (ac)^2 = (ad)^2 = a(cd)^4 \\ = (bc)^3 = (bd)^2 = e^2 = (ae)^2 = (be)^2 = (ce)^2 = (de)^3 = 1\rangle.\end{aligned}$$

For this presentation of $U(\mathcal{A})$, $\mathcal{X}$ maps onto $\langle a, b, c, d\rangle$, $\mathcal{Y}$ maps onto $\langle a, b, c, e\rangle$ and $\mathcal{Z}$ maps onto $\langle a, b, d, e\rangle$; the relations $(ac)^2 = (ad)^2 = 1$ are consequences of the others.

Applying coset enumeration, we find that $\langle a, b, c, d \rangle$ has index 432 in $U(\mathcal{A})$. It is then shown that $U(\mathcal{A}) \cong (3 \times M_{12}):2$, and that there are just two connected ordered-triangle-transitive locally-H graphs whose vertex-stabilizer is $\mathrm{PGL}_2(11)$, having (respectively) 432 and 144 vertices and automorphism groups isomorphic to $U(\mathcal{A})$ and $U(\mathcal{A})/\langle (bcde)^{11} \rangle \cong M_{12}:2$, the automorphism group of the Mathieu group M_{12}.

References

[1] T. Breuer and J. Müller, The character tables of endomorphism rings of multiplicity-free permutation modules of the sporadic simple groups, their automorphism groups, and their cyclic central extension groups; `http://www.math.rwth-aachen.de/~Juergen.Mueller/mferctbl/mferctbl.html`.

[2] A. E. Brouwer, A. M. Cohen and A. Neumaier, *Distance-Regular Graphs*, Springer, 1989.

[3] A. E. Brouwer and J. H. Koolen, The distance-regular graphs of valency four, *J. Algebr. Combin.* **10** (1999), 5–24.

[4] G. Butler, *Fundamental Algorithms for Permutation Groups*, Lecture Notes in Computer Science **559**, Springer, 1991.

[5] P. J. Cameron, *Permutation Groups*, London Math. Soc. Student Texts **45**, Cambridge University Press, 1999.

[6] M. Conder and P. Dobcsányi, Trivalent symmetric graphs on up to 768 vertices, *J. Combin. Math. Combin. Comput.* **40** (2002), 41–63.

[7] M. Conder and P. Dobcsányi, Applications and adaptations of the low index subgroups procedure, *Math. Comput.*, to appear.

[8] J. H. Conway, R. T. Curtis, S. P. Norton, R. A. Parker and R. A. Wilson, *ATLAS of Finite Groups*, Clarendon Press, Oxford, 1985.

[9] H. Cuypers, L. H. Soicher and H. Sterk, Working with finite groups, *Some Tapas of Computer Algebra* (ed. A. M. Cohen, H. Cuypers and H. Sterk), Springer (1999), 184–207.

[10] H. Cuypers, L.H. Soicher and H. Sterk, The small Mathieu groups (Project), *Some Tapas of Computer Algebra* (ed. A. M. Cohen, H. Cuypers and H. Sterk), Springer (1999), 323–337.

[11] I. A. Faradžev and M. H. Klin, Computer package for computations with coherent configurations, *Proc. ISSAC '91* (ed. S. M. Watt), Assoc. Comp. Mach. (1991), 219–223.

[12] I. A. Faradžev, M. H. Klin and M. E. Muzichuk, Cellular rings and groups of automorphisms of graphs, *Investigations in Algebraic Theory of Combinatorial Objects* (ed. I. A. Faradžev, A. A. Ivanov, M. H. Klin and A. J. Woldar), Kluwer (1994), 1–152.

[13] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.3*, Aachen, St Andrews, 2002; `http://www.gap-system.org`.

[14] G. Havas and C. Ramsay, Experiments in coset enumeration, *Groups and Computation III* (ed. W. M. Kantor and Á. Seress), de Gruyter (2001), 183–192.

[15] G. Havas and C. Ramsay, The "Advanced Coset Enumerator" ACE; `http://www.csee.uq.edu.au/~cram/ce.html`.

[16] G. Havas, C. Ramsay, G. Gamble and A. Hulpke, The ACE Package for GAP; `http://www.gap-system.org/Share/ace.html`.

[17] A. A. Ivanov, *Geometry of Sporadic Groups. I: Petersen and Tilde Geometries*, Cambridge University Press, 1999.

[18] A. A. Ivanov, S. A. Linton, K. Lux, J. Saxl and L. H. Soicher, Distance-transitive representations of the sporadic groups, *Comm. Algebra* **23** (1995), 3379–3427.

[19] A. A. Ivanov and S. V. Shpectorov, Applications of group amalgams to algebraic graph theory, *Investigations in Algebraic Theory of Combinatorial Objects* (ed. I. A. Faradžev, A. A. Ivanov, M. H. Klin and A. J. Woldar), Kluwer (1994), 417–441.

[20] W. Kocay, On writing isomorphism programs, *Computational and Constructive Design Theory* (ed. W. D. Wallis), Kluwer (1996), 135–175.

[21] W. Kocay, Groups & Graphs; `http://www.paddle.mb.ca/G&G/G&G.html`.

[22] J. S. Leon, Partitions, refinements, and permutation group computation, *Groups and Computation II* (ed. L. Finkelstein and W. M. Kantor), DIMACS Series in Discrete Mathematics and Theoretical Computer Science **28**, Amer. Math. Soc. (1997), 123–158.

[23] S. A. Linton, K. Lux and L. H. Soicher, The primitive distance-transitive representations of the Fischer groups, *Experimental Math.* **4** (1995), 235–253.

[24] B. D. McKay, Computing automorphisms and canonical labellings of graphs, *Comb. Math., Proc. Int. Conf., Canberra 1977*, Lecture Notes in Mathematics **686**, Springer (1978), 223–232.

[25] B. D. McKay, Practical graph isomorphism, *Congr. Numerantium* **30** (1981), 45–87.

[26] B. D. McKay, nauty; `http://cs.anu.edu.au/people/bdm/nauty`.

[27] The Magma Computational Algebra System, Computer Algebra Group, School of Mathematics and Statistics, University of Sydney; `http://magma.maths.usyd.edu.au`.

[28] T. Miyazaki, The complexity of McKay's canonical labelling algorithm, *Groups and Computation II* (ed. L. Finkelstein and W. M. Kantor), DIMACS Series in Discrete Mathematics and Theoretical Computer Science **28**, Amer. Math. Soc. (1997), 239–256.

[29] M. J. Morton, Classification of 4- and 5-arc-transitive cubic graphs of small girth, *J. Aust. Math. Soc. (A)* **50** (1991), 138–149, and Corrigendum, *J. Aust. Math. Soc. (A)* **52** (1992), 419–420.

[30] J. Neubüser, An elementary introduction to coset table methods in computational group theory, *Groups – St. Andrews 1981* (ed. C. M. Campbell and E. F. Robertson), London Math. Soc. Lecture Note Series **71**, Cambridge University Press (1982) 1–45.

[31] C. E. Praeger and L. H. Soicher, *Low Rank Representations and Graphs for Sporadic Groups*, Australian Math. Soc. Lecture Series **8**, Cambridge University Press, 1997.

[32] S. Rees and L.H. Soicher, An algorithmic approach to fundamental groups and covers of combinatorial cell complexes, *J. Symbolic Comp.* **29** (2000), 59–77; GAP program available at: `http://www.maths.qmul.ac.uk/~leonard/fundamental`.

[33] Á. Seress, *Permutation Group Algorithms*, Cambridge University Press, 2003.

[34] C. C. Sims, Computation with permutation groups, *Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation* (ed. S. R. Petrick), Assoc. Comp. Mach. (1971), 23–28.

[35] C. C. Sims, *Computation with Finitely Presented Groups*, Cambridge University Press, 1994.

[36] L. H. Soicher, On simplicial complexes related to the Suzuki sequence graphs, *Groups, Combinatorics and Geometry* (ed. M.W. Liebeck and J. Saxl), London Math. Soc. Lecture Note Series **165**, Cambridge University Press (1992), 240–248.

[37] L. H. Soicher, Three new distance-regular graphs, *Europ. J. Comb.* **14** (1993), 501–505.

[38] L.H. Soicher, The GRAPE Package for GAP; `http://www.gap-system.org/Share/grape.html`.

[39] R. Weiss, A characterization and another construction of Janko's group J_3, *Trans. Amer. Math. Soc.* **298** (1986), 621–633.

[40] R. Weiss, Presentations for (G, s)-transitive graphs of small valency, *Math. Proc. Camb. Philos. Soc.* **101** (1987), 7–20.

[41] R.A. Wilson *et al.*, ATLAS of finite group representations; `http://www.mat.bham.ac.uk/atlas`.