ISOMORPH-FREE EXHAUSTIVE GENERATION

Brendan D. McKay

Computer Science Department, Australian National University,

Canberra, ACT 0200, Australia

bdm@cs.anu.edu.au

AMS Subject Numbers: 68R05, 05C30

Suggested running head: ISOMORPH-FREE EXHAUSTIVE GENERATION

Abstract.

We describe a very general technique for generating families of combinatorial objects without isomorphs. It applies to almost any class of objects for which an inductive construction process exists. In one form of our technique, no explicit isomorphism testing is required. In the other form, isomorph testing is restricted to within small subsets of the entire set of objects. A variety of different examples are presented, including the generation of graphs with some hereditary property, the generation of Latin rectangles and the generation of balanced incomplete block designs. The technique can also be used to perform unbiased statistical analysis of sets of objects too large to generate exhaustively.

1. Introduction.

Problems of exhaustive generation fall into several genres, depending on the structure of the objects being generated. The objects most easily generated seem to be characterised by having easy isomorphism problems, frequently due to a fundamentally recursive structure. There is a vast literature concerning the generation of subsets, permutations, partitions, trees, and similar objects. At the other extreme, we have generation problems typified by graphs. In this case the isomorphism problem tends to be difficult, and the success of efficient methods is due in large part to devices for avoiding it. These latter are the sort of problems we will be concerned with in this paper.

Most methods proposed for such problems can be classified into three types. In the most common method, there is a canonical labelled object in each isomorphism class and that is the one generated. The labelling is chosen to impose restrictions on sub-objects; for example that there must be one maximal sub-object which is also canonical. The most general description of this method was given by I. A. Faradzev [11]. The most famous example of this method is Read's algorithm for graphs [30]. More recent examples are Brinkmann's generator of cubic graphs [5], Meringer's generator of regular graphs [29], and the generation of one-factorisations of the complete graph by Dinitz, Garnick and this author [10].

The second method, the subject of this paper, can loosely be described as generation by canonical construction path, as opposed to canonical representation. Objects are produced by somehow augmenting a smaller object, with only objects made via a canonical augmentation being accepted. Intermediate objects can even be relabelled at random with no effect. The first published use of this method was for cubic graphs in 1986 [26], but many applications have appeared since. We will give a list of known examples later. These are so diverse in nature that a very abstract setting is required to define the method in sufficient generality to cover them all. This paper is devoted to that task.

A third method, recently developed by Laue and others [14], is the "method of homomorphisms". In this approach objects are constructed along a path of combinatorially determined sub-objects, using elementary group computations to handle the relationship between the automorphism structure of consecutive sub-objects. It is necessary to use another method for the sub-objects which do not decompose further. For example, Meringer's generator for regular graphs [29] can be used as the basis for a very fast generator of graphs with a specified degree partition. There is some relationship between Laue's approach and ours, but it has not yet been explored.

Of recent generation algorithms that do not precisely fit any of our three categories, perhaps the method of Brinkmann and Dress for generating fullerenes [7] is the most interesting.

In several recent papers [1, 2], Avis and Fukuda describe their "reverse search" method for generating classes of objects. Reverse search shares with our method the idea of defining a tree structure on a set of objects by means of a "parent" function, then scanning it from the root outwards. The major difference in our approach is that we allow a rich group of symmetries to be acting, whereas the examples treated by Avis and Fukuda are essentially labelled.

The advantages of our approach are several. Most importantly, the method applies to a great number of diverse problems, as we shall demonstrate. Secondly, representatives of each isomorphism type are generated in a stream with no need to store more than a handful at a time, which allows us to generate extremely large classes. Thirdly, the method allows for very simple parallelization with near-linear speedup. Finally, a simple extension of the method (in many cases) allows unbiased sampling of the isomorphism types in a class too big to generate exhaustively.

2. The Algorithm.

In order to assist the reader to understand the formal treatment, we will take an actual example and carry it along in parallel. Our example will be the generation of triangle-free graphs (*TFGs*) starting with a single vertex and repeatedly adding vertices. To keep the general description distinguishable from the example, we will use the labels "*General*" and "*TFG*".

General: Let G be a permutation group acting on a (possibly infinite) set $\mathcal{L}$. The elements of $\mathcal{L}$ will be called *labelled objects*, and the orbits of G in $\mathcal{L}$ will be called *unlabelled objects*. The set of unlabelled objects will be denoted by $\mathcal{U}$. Each labelled object $X \in \mathcal{L}$ has an *order* $o(X) \in \mathbb{N}$, with the restriction that $o(X)$ is constant on unlabelled objects. This restriction permits us to define $o(S)$, for $S \in \mathcal{U}$, to be the common order of the labelled objects comprising S.

TFG: In our example, $\mathcal{L}$ is the set of all labelled TFGs, using the labels $\{1, 2, \ldots, n\}$ to label a TFG X with n vertices. Take $o(X) = n$. The group G is the group of all relabellings of labelled TFGs, so that one orbit (an "unlabelled TFG") consists of all the TFGs isomorphic to given TFG. Formally, we can take $G = S_1 \times S_2 \times S_3 \times \cdots$, where the action on $\mathcal{L}$ is such that the component S_n acts by permuting the labels on TFGs of order n.

General: The general problem we will consider is that of making a list of labelled objects, such that the list contains exactly one labelled object from each unlabelled object whose order is at most some specified value. We will see that a small amount of extra structure permits this to be done in a systematic manner.

With each labelled object $X \in \mathcal{L}$, associate a finite set $L(X)$ of *lower objects* and a finite set $U(X)$ of *upper objects*. Define $\check{\mathcal{L}} = \bigcup_{X \in \mathcal{L}} L(X)$ and $\hat{\mathcal{L}} = \bigcup_{X \in \mathcal{L}} U(X)$. For distinct $X_1, X_2 \in \mathcal{L}$ we require that the six sets $\{X_1\}$, $L(X_1)$, $U(X_1)$, $\{X_2\}$, $L(X_2)$ and $U(X_2)$ be mutually disjoint. The *order* of the elements of $L(X)$ and $U(X)$ is defined to be the same as the order of X, with the same notation. Furthermore, suppose there is a function $f : \check{\mathcal{L}} \to \hat{\mathcal{L}}$ and a group G acting on $\mathcal{L} \cup \check{\mathcal{L}} \cup \hat{\mathcal{L}}$ such that the following hold.

- C1. G fixes each of $\mathcal{L}$, $\check{\mathcal{L}}$ and $\hat{\mathcal{L}}$ setwise.
- C2. For each $X \in \mathcal{L}$ and $g \in G$ we have $L(X^g) = L(X)^g$ and $U(X^g) = U(X)^g$.
- C3. For each $\check{X} \in \check{\mathcal{L}}$ and $g \in G$ there exists $h \in G$ such that $f(\check{X}^g) = f(\check{X})^h$.
- C4. For any $\hat{X} \in \hat{\mathcal{L}}$ and $g \in G$, let $\check{Y}_1 \in f^{-1}(\hat{X})$ and $\check{Y}_2 \in f^{-1}(\hat{X}^g)$. Then there exists $h \in G$ such that $\check{Y}_1^h = \check{Y}_2$.
- C5. For each $X \in \mathcal{L}$ and $g \in G$ we have $o(X^g) = o(X)$.
- C6. For each $\check{X} \in \check{\mathcal{L}}$ we have $o(f(\check{X})) < o(\check{X})$.

TFG: We are making TFGs by adding a new vertex to a smaller TFG, and a lower object contains the information needed to go backwards one step. We can define it as a pair $\langle X, v\rangle$, where X is a TFG and $v \in V(X)$. This provides a route "remove v from X" from a larger TFG to a smaller one. The set $L(X)$ contains all such pairs, except for the special case $L(K_1) = \emptyset$. Conversely, an upper object contains the information needed to go from a smaller object to a larger one. So, we define $U(X')$ to be the set of all pairs $\langle X', W\rangle$, where $W \subseteq V(X')$ is an independent set of X'. It provides a route "add a new vertex to X', and join it to W" to increase the order. (W must be an independent set in order to prevent the creation of triangles.)

The action of G is easily extended to upper and lower objects: just take $\langle X, v\rangle^g = \langle X^g, v^g\rangle$ and $\langle X', W\rangle^g = \langle X'^g, W^g\rangle$. This reasonable consistency of the action of G on $\mathcal{L}$, $\check{\mathcal{L}}$ and $\hat{\mathcal{L}}$ is all that we need for C1 and C2 to be satisfied.

Clearly, the upper and lower objects are complementary: $\langle X, v\rangle \in L(X)$ is related to $\langle X - v, W\rangle$, where W is the neighbourhood of v in X. This relationship is formalized by the function $f : \langle X, v\rangle \mapsto \langle X - v, W\rangle$, where the labelling of $X - v$ is made by squashing out the label on v. (For example, if $V(X) = \{1, 2, 3, 4, 5\}$ and $v = 4$, then vertex 5 is relabelled as 4 when forming $X - v$.) This need for relabelling prevents f from having a unique inverse: forming $\langle X, v\rangle$ from $\langle X - v, W\rangle$ requires us to choose a label for v, which can be done in $o(X)$ ways. However, these alternatives are all equivalent under G, and more generally the way in which f preserves the labelling as best it can is all that is needed for compliance with C3 and C4.

General: We will say that two labelled objects $X_1, X_2 \in \mathcal{L}$ are *isomorphic* if $X_2 = X_1^g$ for some $g \in G$. Similarly, for any labelled object $X \in \mathcal{L}$, the *automorphism group* $\mathrm{Aut}(X)$ of X is the stabiliser $\{g \in G \,|\, X^g = X\}$. From condition C2 we see that $\mathrm{Aut}(X)$ fixes each of $L(X)$ and $U(X)$ setwise.

An unlabelled object $S \in \mathcal{U}$ will be called *elementary* if $L(X) = \emptyset$ for each $X \in S$. (Note that the condition $L(X) = \emptyset$ is invariant under G, by condition C2.) Define $\mathcal{U}_0$ to be the set of elementary unlabelled objects, and $\mathcal{U}_1$ to be the set of non-elementary unlabelled objects. Thus, $\mathcal{U} = \mathcal{U}_0 \cup \mathcal{U}_1$.

TFG: In our example, the concepts of isomorphism and automorphism are just the usual ones. We have only a single elementary object, namely the unlabelled graph K_1.

General: Our final requirement is a function $m : \mathcal{L} \to 2^{\check{\mathcal{L}}}$ with the following properties.

M1. If $L(X) = \emptyset$, then $m(X) = \emptyset$.

M2. If $L(X) \neq \emptyset$, then $m(X)$ is an orbit of the action of $\mathrm{Aut}(X)$ on $L(X)$.

M3. For each $X \in \mathcal{L}$ and $g \in G$ we have $m(X^g) = m(X)^g$.

The properties of the mappings f and m enable us to define a nilpotent mapping p

from $\mathcal{U}_1$ to $\mathcal{U}$ that imposes a structure on $\mathcal{U}$ in the form of a family of disjoint trees.

Lemma 1. *There is a unique mapping $p : \mathcal{U}_1 \to \mathcal{U}$ with the following property.*

P1. *For each $S \in \mathcal{U}_1$, $X \in S$ and $\check{X} \in m(X)$, we have $f(\check{X}) \in U(Y)$ for some $Y \in p(S)$.*

Proof. Choose $S \in \mathcal{U}_1$ and $X_1, X_2 \in S$. For $i = 1, 2$ choose $\check{X}_i \in m(X_i)$, set $\hat{Y}_i = f(\check{X}_i)$ and define Y_i by $\hat{Y}_i \in U(Y_i)$. It will suffice to show that Y_1 and Y_2 are isomorphic.

Since $X_1, X_2 \in S$, there is $g \in G$ such that $X_2 = X_1^g$. From properties M2 and M3, this implies the existence of $h \in G$ such that $\check{X}_2 = \check{X}_1^h$. Further, $\hat{Y}_2 = f(\check{X}_2) = f(\check{X}_1^h) = f(\check{X}_1)^k = \hat{Y}_1^k$ for some $k \in G$, where the second last step is justified by property C3. Property C2 now implies that $Y_2 = Y_1^k$. ∎

TFG: The definition of $m(X)$ is the most onerous requirement for any practical application. A possible definition of $m(X)$ would be this: consider all the labellings of X, and choose the one which is greatest under some ordering of labelled graphs (such as a lexicographic ordering). Let v^* be the vertex whose label is "1" in this maximal labelling. Then define $m(X)$ to be the set of lower objects $\langle X, v\rangle$ such that v is equivalent to v^* under $\mathrm{Aut}(X)$.

It is easy to see that conditions M2 and M3 are satisfied, but we are faced with the problem of computing $m(X)$. In practice, we can do better using sophisticated tools for canonical labelling, with judicious screening beforehand using combinatorial invariants. We will discuss this in more detail in Sections 3 and 4; meanwhile we can continue with the definition above for the sake of the example.

The nilpotent function p predicted by Lemma 1 is now easily identified. Starting with an non-elementary unlabelled object (isomorphism class of nontrivial TFGs) S, choose any labelling $X \in S$, then any lower object $\langle X, v\rangle \in m(X)$, then evaluate $f(\langle X, v\rangle)$ to get the upper object $\langle X', W\rangle$ and, finally, note the unlabelled object (isomorphism class) S' containing X'. The point of Lemma 1 is that the same S' is reached no matter how the two arbitrary choices are made.

General: The unlabelled object $p(S)$ will be called the *parent* of the unlabelled object S, and the set $\{S, p(S), p(p(S)), \ldots\}$ will be called the *ancestors* of S. The inverse concepts *child* and *descendant* are defined in the obvious way. It is clear from property C6 that p is nilpotent, in other words that no object has more than finitely many ancestors.

The relation $\{(S, p(S)) \mid S \in \mathcal{U}_1\}$ on $\mathcal{U}$ is a set of disjoint directed rooted trees with the edges directed towards the roots. The roots are precisely the elementary unlabelled objects. In order to achieve our aim of generating a transversal for $\mathcal{U}$, we can scan these trees in some systematic way. We will use a preorder traversal (also called depth-first search). This is achieved by the following algorithm.

> **procedure** $\mathtt{scan}(X$: labelled object, n : integer)
>   **output** X
>   **for** each orbit A of the action of $\mathrm{Aut}(X)$ on $U(X)$ **do**
>   select any $\hat{X} \in A$
>   **if** $f^{-1}(\hat{X}) \neq \emptyset$ **then**
>   select $\check{Y} \in f^{-1}(\hat{X})$, and suppose $\check{Y} \in L(Y)$
>   **if** $o(Y) \leq n$ **and** $\check{Y} \in m(Y)$ **then** $\mathtt{scan}(Y, n)$ **endif**
>   **endif**
>   **endfor**
> **endprocedure**

Theorem 1. *Suppose $X_0 \in S_0 \in \mathcal{U}$ with $o(X_0) \leq n$. Then the call $\mathtt{scan}(X_0, n)$ will output exactly one labelled object belonging to each unlabelled object of order at most n which is descended from S_0.*

Proof. Let us say that a descendant S of S_0 belongs to *generation* i if S has $i+1$ ancestors. Recall that an unlabelled object is an ancestor of itself, so generation 0 is the set $\{S_0\}$.

Suppose as an induction hypothesis that the theorem is true for all objects of order at most n and generation at most i. This statement is clearly true for $i = 0$.

Now suppose S is an arbitrary descendant of S_0 which has order at most n and belongs to generation $i+1$. Let $T = p(S)$.

Take an arbitrary $Y' \in S$, choose $\check{Y}' \in m(Y')$, and define $\hat{X}' = f(\check{Y}')$. By the induction hypothesis and the nature of the **for** condition, there is some $X \in T$ and some $\hat{X} \in U(X)$ such that there is a call $\mathtt{scan}(X, n)$, $\hat{X}$ is 'selected' inside the **for**, and $\hat{X}^g = \hat{X}'$ for some $g \in G$. Hence, by property C4, $\check{Y}^h = \check{Y}'$ for some $h \in G$ and so $Y^h = Y'$ by condition C2, implying that $Y \in S$. Furthermore, since $\check{Y}' \in m(Y')$, by assumption, we have $\check{Y} \in m(Y)$ by condition M3. Therefore, the call $\mathtt{scan}(Y, n)$ is made and Y is output.

We are left with the question of whether the unlabelled object S can be represented more than once in the output. Suppose that distinct $Y_1, Y_2 \in S$ are both output, and choose $g \in G$ such that $Y_2 = Y_1^g$. Let $\check{Y}_1 \in m(Y_1)$ and $\check{Y}_2 \in m(Y_2)$ be the lower objects presented to the **if** preceding the calls $\mathtt{scan}(Y_1, n)$ and $\mathtt{scan}(Y_2, n)$, respectively. Now, $\check{Y}_2 \in m(Y_2) = m(Y_1^g) = m(Y_1)^g$ by condition M3, and so $\check{Y}_2 = \check{Y}_1^h$ for some $h \in G$, by condition M2. Write $\hat{X}_1 = f(\check{Y}_1)$ and $\hat{X}_2 = f(\check{Y}_2)$. Then $\hat{X}_2 = \hat{X}_1^k$ for some $k \in G$, by condition C3, and if $\hat{X}_1 \in U(X_1)$ and $\hat{X}_2 \in U(X_2)$ we have $X_2 = X_1^k$ by condition C2. By the induction hypothesis, since both X_1 and X_2 are output, we must have $X_1 = X_2$ and so $k \in \mathrm{Aut}(X_1)$. But this shows $\hat{X}_1$ and $\hat{X}_2$ to be in the same orbit of $\mathrm{Aut}(X_1)$, implying that $\hat{X}_1 = \hat{X}_2$ by the **for** condition, and hence that $Y_1 = Y_2$ contrary to assumption. ∎

TFG: In informal terms, given an isomorphism class of TFGs (represented by any labelling), we make a set of potential children by applying the inverse of f. These are tested using m to see which are real children and which are not. The former are accepted, the latter rejected. Theorem 1 states that exactly one member of each isomorphism class is accepted as a child.

General: For some practical purposes, it is convenient to have use a modified version of procedure `scan` which replaces an orbit computation by explicit isomorph testing. It has the same external properties as procedure `scan`, as can proved by similar means.

> **procedure** `scan2`(X : labelled object, n : integer)
>   **output** X
> $C := \emptyset$
>   **for** each $\hat{X} \in U(X)$ such that $f^{-1}(\hat{X}) \neq \emptyset$ **do**
>   select $\check{Y} \in f^{-1}(\hat{X})$, and suppose $\check{Y} \in L(Y)$
>   **if** $o(Y) \leq n$ **and** $\check{Y} \in m(Y)$ **then** $C := C \cup \{Y\}$ **endif**
>   **endfor**
>   remove isomorphs from the set C
>   **for** each $Y \in C$ **do** `scan2`(Y, n) **endif**
> **endprocedure**

Theorem 2. *Suppose $X_0 \in S_0 \in \mathcal{U}$ with $o(X_0) \leq n$. Then the call* `scan2`(X_0, n) *will output exactly one labelled object belonging to each unlabelled object of order at most n which is descended from S_0.* ∎

It is clear from Theorems 1 and 2 that, if we make either the call `scan`(X_0, n) or the call `scan2`(X_0, n) for exactly one representative of each elementary unlabelled object of order at most n, we will obtain exactly one representative of *every* unlabelled object of order at most n.

One way to quantify the efficiency of algorithm `scan` is to compare the number of executions of the body of the **for** loop to the number of objects output. Since we cannot make general precise statements about how $o(Y)$ compares to $o(X)$ (in the notation of the procedure), we concentrate our attention on the test "$\check{Y} \in m(Y)$".

Theorem 3. *Consider the computations initiated by calling* $\texttt{scan}(X_0, n)$ *for exactly one representative of each elementary unlabelled object of order at most* n. *Let* N_1 *be the number of times the test "*$\check{Y} \in m(Y)$*" is made, and let* N_2 *be the number of times it is passed. Then* $N_1 \le cN_2$, *where* c *is the average number of orbits of* $\mathrm{Aut}(X)$ *on* $L(X)$, *with the average taken over one representative from each non-elementary unlabelled object of order at most* n.

Proof. Suppose $S \in \mathcal{U}_1$ and $o(S) \le n$. Let $Y_1, Y_2 \in S$, $\check{Y}_1 \in L(Y_1)$ and $\check{Y}_2 \in L(Y_2)$. Define $\hat{X}_1 = f(\check{Y}_1)$ and $\hat{X}_2 = f(\check{Y}_2)$. If $\check{Y}_1^g = \check{Y}_2$ for some $g \in G$, then $\hat{X}_1^h = \hat{X}_2$ for some $h \in G$, by property C3. As in the proof of Theorem 1, this implies that $\hat{X}_1 = \hat{X}_2$ if $\hat{X}_1$ and $\hat{X}_2$ both occur as the value of $\hat{X}$ during some invocation of `scan`. Therefore, at most one of the pairs $(\check{Y}_1, Y_1)$ and $(\check{Y}_2, Y_2)$ is passed to the test "$\check{Y} \in m(Y)$". Hence, the number of times the test "$\check{Y} \in m(Y)$" is performed for any $Y \in S$ is at most the number of orbits of the action of G on $\bigcup_{Y \in S} L(Y)$, which is the same as the number of orbits of the action of $\mathrm{Aut}(Y)$ on $L(Y)$ for any given $L \in S$. ∎

TFG: Theorem 3 is usually very easy to apply. In our TFG example, it is clear that each class $L(X)$ of lower objects contains exactly $n = |V(X)|$ members, and thus at most n orbits. Thus, generating all the TFGs up to order n (supposing there are N of them) requires less than N automorphism group computations and at most nN computations of the function m.

Generally speaking, objects for which the isomorphism and automorphism problems have polynomial-time algorithms (such as many types of planar graphs) will have generation schemes with polynomial amortised time per output. More precise statements are hard to make in this generality.

3. Graphs and `nauty`.

A large number of applications of our methods can be found in the fields of graphs and hypergraphs. Efficient implementation of the function m and the computation of orbits is a nontrivial task, but can be accomplished with existing tools. In our applications, we have used the program `nauty` described in [19]. Most of the mathematical foundations of `nauty` can be found in [18]. For our purposes here it will suffice to give a functional overview.

Let X be a graph with vertex-set $V = V(X) = \{1, 2, \ldots, n\}$. A *partition* of V is a sequence $\pi = (V_1, V_2, \ldots, V_k)$ of non-empty disjoint subsets (called *cells*) of V whose union is V. A permutation g of V acts on π cell-wise; i.e., $\pi^g = (V_1^g, V_2^g, \ldots, V_k^g)$. Similarly, g acts on X by permuting the names of the vertices, so that v and w are adjacent in X if and only if v^g and w^g are adjacent in X^g. The *automorphism group* of

(X, π) is $\mathrm{Aut}_\pi(X) = \{g \in S_n \mid X^g = X \text{ and } \pi^g = \pi\}$. For the special partition $\pi_0 = (V)$, we will abbreviate $\mathrm{Aut}_{\pi_0}(X)$ as $\mathrm{Aut}(X)$. If $\pi = (V_1, V_2, \ldots, V_k)$ is a partition of V, then $c(\pi)$ is the partition $(\{1, 2, \ldots, |V_1|\}, \{|V_1|+1, \ldots, |V_1|+|V_2|\}, \ldots, \{n-|V_k|+1, \ldots, n\})$. Thus, $c(\pi)$ depends only on the sizes of the cells of π and their order. A *canonical labelling map* is a function C such that for any graph X with vertex set V, and partition π of V, we have

N1. $C(X, \pi) = X^g$ for some $g \in S_n$ such that $\pi^g = c(\pi)$.

N2. $C(X^h, \pi^h) = C(X, \pi)^h$ for every $h \in S_n$.

The behaviour of `nauty` can now be described. For input (X, π), there are three outputs. One is the value of $C(X, \pi)$ for some fixed canonical labelling map C, the second is a permutation $g \in S_n$ such that $X^g = C(X, \pi)$ and $\pi^g = c(\pi)$, and the third is a set of generators for $\mathrm{Aut}_\pi(X)$. The map C computed by `nauty` is designed to be efficiently computed rather than easily described; fortunately, we will only need to know that it satisfies properties N1 and N2.

4. Graphs with a hereditary property.

In Section 2, we used the example of generating triangle-free graphs. It is clear that the same idea works for generation of graphs that possess some arbitrary hereditary property $\mathcal{P}$. Here, the adjective *hereditary* indicates that, for any graph X with property $\mathcal{P}$, every induced subgraph of X also has property $\mathcal{P}$. It will be easily seen that the requirement that $\mathcal{P}$ be hereditary can be relaxed to the requirements that $\mathcal{P}$ is held by at least one vertex-deleted subgraph of any non-trivial graph which holds it, and that $\mathcal{P}$ is invariant under isomorphisms.

In this section we will look a little harder at this problem.

Let X be any labelled graph with property $\mathcal{P}$. The *order* of X is $o(X) = |V(X)|$, as usual. We will assume for definiteness that $V(X) = \{1, 2, \ldots, o(X)\}$ as in the previous section. If $o(X) > 1$, then the lower objects $L(X)$ are the pairs $\langle X, v\rangle$ for $v \in V(X)$. The trivial graph K_1 has $L(K_1) = \emptyset$; its isomorphism class is the only elementary unlabelled object. The upper objects $U(X)$ are the pairs $\langle X, W\rangle$, where $W \subseteq V(X)$ is such that the graph X_W formed by appending a new vertex to X and joining it to the elements of W has property $\mathcal{P}$.

The function f is defined as follows. Suppose $\langle X, W\rangle \in U(X)$ and $\langle Y, v\rangle \in L(Y)$. Then $f(\langle Y, v\rangle) = \langle X, W\rangle$ if $Y = X_W$, with v being the new vertex and the other vertices being labelled in the same order in X and Y.

Since we are dealing with standard graph isomorphisms, we can take the group G to be any group which acts on the graphs of each order n as the symmetric group of degree n, represented as the set of all possible permutations of the vertex set. The manner in which the actions on graphs of different orders are related in G is immaterial; in practice there is little need to even define it.

We can now define a function m satisfying conditions M1–M3. For purposes of computational efficiency, of which more in a moment, we begin with a function λ defined on labelled graphs X with property $\mathcal{P}$, such that

L1. $\emptyset \neq \lambda(X) \subseteq V(X)$;

L2. for any $g \in S_n$, we have $\lambda(X^g) = \lambda(X)^g$.

These conditions imply that $\lambda(X)$ is a nonempty union of orbits of $\mathrm{Aut}(X)$. We could in fact take $\lambda(X) = V(X)$ for all X, but it will be more efficient to choose a smaller value—for example, the set of vertices of X of maximum degree, or defined by some more stringent invariant property. Now define the function m as follows.

Let X be a labelled graph with property $\mathcal{P}$. If X has only one vertex, define $m(X) = \emptyset$. Otherwise, let $\pi = (\lambda(X), V(X) - \lambda(X))$ where the second cell is omitted if it is empty. Let $g \in S_n$ be such that $X^g = C(X, \pi)$ and $\pi^g = c(\pi)$, and let W be the orbit of $\mathrm{Aut}(X)$ which contains the vertex $1^{g^{-1}}$. Then define $m(X) = \{(X, v) \mid v \in W\}$.

Lemma 2. *Under the conditions just described, $m(X)$ is well-defined and satisfies requirements M1–M3.*

Proof. It is obvious that M1 and M2 are satisfied. To see that $m(X)$ is well-defined, suppose that h is another permutation such that $X^h = C(X, \pi)$ and $\pi^h = c(\pi)$. Then $g^{-1}h$ is an automorphism of X and so $1^{g^{-1}}$ and $1^{h^{-1}}$ lie in the same orbit of X. Finally, property M3 follows from N2 and L2. ∎

With some tuning, this method can be made quite fast. In Tables 1 and 2, we give some examples of graph types generated by the author's program `makeg6`. In each case, the number of graphs produced per second on a Sun SPARCstation 20/71 computer at 75MHz is given. This number is approximately independent of the order of the graph for the practical range, but must fall for larger orders. `makeg6` is many times faster than previously published methods [9, 16], and slightly faster than the current implementations of the method of homomorphisms [14]. We believe it was the first program to construct all the 11-vertex graphs, and the only one to contruct all the 12-vertex graphs.

Other published examples of this approach to graphs have included several types of Ramsey graph [12, 21, 30].

5. Other examples for graphs and hypergraphs.

Instead of constructing graphs one vertex at a time, we could do it one edge at a time. The details are very similar: the upper objects consist of a graph and a pair of

n	general	C_3-free	C_4-free
1	1	1	1
2	2	2	2
3	4	3	4
4	11	7	8
5	34	14	18
6	156	38	44
7	1044	107	117
8	12346	410	351
9	274668	1897	1230
10	12005168	12172	5069
11	1018997864	105071	25181
12	165091172592	1262180	152045
13		20797002	1116403
14		467871369	9899865
15			104980369
speed	33000/sec	18000/sec	15000/sec

Table 1. Counts of general, C_3-free and C_4-free graphs

non-adjacent vertices, while the lower objects consist of a graph with a distinguished edge. An example from 1984 (but not published until 1995, see [8]) was a step in the generation of cubic cages.

More complicated operations than simple addition of elements can be used too. For example, the first construction of the cubic graphs on 20 vertices [26] used the operations of adding a path of arbitrary length between two vertices of degree 2, and of attaching a path of arbitrary length to one vertex of degree two with an arbitrary cycle fixed onto the other end of the path. This complicated process worked but was not very efficient. The main reason for the inefficiency was that the operation did not keep us within the class of cubic graphs. We needed to also consider some graphs with degrees 2 and 3, which greatly expanded the search space. A more promising approach [27] uses the operation of subdividing two edges and joining the two new edges. Probably it will be competitive with Brinkmann's totally different approach to cubic graphs [5].

In some cases, careful choice of the function m can have importance beyond mere

n	(C_3, C_4)-free	bipartite	C_4-free bipartite
1	1	1	1
2	2	2	2
3	3	3	3
4	6	7	6
5	11	13	10
6	23	35	21
7	48	88	39
8	114	303	86
9	293	1119	182
10	869	5479	440
11	2963	32303	1074
12	12066	251135	2941
13	58933	2527712	8424
14	347498	33985853	26720
15	2455693	611846940	90883
16	20592932		340253
17	202724920		1384567
18			6186907
19			30219769
20			161763233
speed	11000/sec	17000/sec	6000/sec

Table 2. Counts of (C_3, C_4)-free, bipartite, and C_4-free bipartite graphs

efficiency. In [20], we show that no two graphs on 11 vertices have the same set of 11 vertex-deleted subgraphs. Since there are more than 10^9 graphs to compare to each other, simply being able to generate them quickly is not enough. However, defining m according to the isomorphism types of the vertex-deleted subgraphs, we can arrange that any pair of graphs forming a counterexample must appear as children of the same 10-vertex graph. This enables the comparisons to be done in small groups.

An example of hypergraph generation appeared in [22], and an example of digraph generation in [15].

A number of applications to graph generation, some with chemical motivation, have been carried out by Brinkmann and others [4, 6, 13].

Avis and Fukuda [1] state as a problem the generation of unrooted triangulations of the plane. It is clear (see [3]) that one can use the operation of deleting a vertex and triangulating the resulting face to reduce any triangulation to the smallest one (K_4). With the help of linear-time isomorphism and automorphism algorithms, an amortised complexity of $O(n^2)$ per triangulation (or better) can be achieved. This process is ideal for our method, and should result in a very fast generator (Brinkmann, McKay et al., in progress).

6. Examples for non-graph objects.

A number of other applications of this method have been implemented. For example, [23] describes a method for generating block designs. The intermediate objects are the partial designs induced by a subset of the vertices. If $\langle W\rangle$ is that object defined by $W \subseteq V$, where V is the set of all the points, then the lower objects have the form $(\langle W \cup \{v\}\rangle, v, 0)$ for $v \in V - W$ and the upper objects have the form $(\langle W\rangle, v, 1)$ for $v \in W$. The mapping f merely takes $(\langle W \cup \{v\}\rangle, v, 0)$ to $(\langle W \cup \{v\}\rangle, v, 1)$. In plain terms, we proceed by adding one point at a time until we have them all.

The same method was used in [24] to eliminate a class of 2-designs.

A similar approach makes Latin rectangles by augmenting with one row at a time. The upper objects consist of pairs $\langle R, x\rangle$, where R is a Latin rectangle and x is a permutation disjoint from R (i.e. a potential new row). The lower objects consist of Latin rectangles with a distinguished row. This was used to verify all the tables presented in [17] (unpublished).

A special case of our method is described in [31], with examples as diverse as finding non-equivalent cliques in a graph and arcs, caps and flocks in finite geometries.

7. Estimation and Random Generation.

As described above, our method defined a set of rooted trees whose nodes are the unlabelled objects. Using standard methods we can estimate the sizes of the trees without generating them completely. One such method is the celebrated algorithm of Knuth using random paths through the tree.

Another one involves assigning a probability $p_0, p_1, \ldots$ to each generation in the trees (p_0 for the roots, p_1 for the children of roots, etc.). Then we can add a stochastic filter to our generation: as each object of generation i is made, reject it with probability $1-p_i$. It is easily seen that each object of generation i is constructed and accepted with probability exactly $p_0 p_1 \cdots p_i$, so the number of objects accepted divided by $p_0 p_1 \cdots p_i$ is an unbiased estimator of the total number of that generation.

For example, by this means the number of Steiner systems 2-(19,3,1) was estimated to be probably between 1.1×10^{10} and 1.2×10^{10}, about 10 times larger than previously believed [32]. Another example was the estimation of the numbers of $(4,5)-$Ramsey graphs in [25].

More generally, we can make an unbiased estimate of the expectation of any random variable defined on a class of unlabelled objects suitable for our method. Since every object of the same generation has the same probability of being accepted, the expectation over accepted objects is the same as the expectation over all objects (within one generation).

8. Parallelization.

Since our method divides the computation into a clean forest structure, parallelization on a MIMD computer is easy. Non-intersecting subtrees are completely independent, and so can be generated on separate processors if desired.

In most practical applications, adequate partitioning can be obtained by drawing a line across the search forest at some level (order of object) such that the number of objects at that level is significantly larger than the number of processors, but small enough for them to be computed quickly. For example, if we are generating all 11-vertex graphs, we can draw a line at 8 vertices. It takes less than one second to generate all 12346 8-vertex graphs, so this can be done independently by each processor. Each processor counts the 8-vertex graphs $0, 1, 2, \ldots$ as they are made, and proceeds to generate all the descendants of those whose ordinals are congruent to i mod p, where i is the index of the processor and p is the number of processors. In the 11-vertex example, the wastage from repeated computations is much less than one percent, and the elapsed time for 20 processors is about 18 times less than for one processor.

In some rare cases, there may be no "line" satisfying the above criteria. Those cases can usually be handled by using two lines several levels apart. It is also possible to devise a strategy where the line dynamically adjusts its own level, but we shall leave such topics for another time.

9. Software availability.

The current version of `nauty`, as well as the program `makeg6` and similar utilities will always be available via the author's home page: `http://cs.anu.edu.au/~bdm` .

References.

[1] D. Avis, Generating rooted triangulations without repetitions, *Algorithmica*, to appear.

[2] D. Avis and K. Fukuda, Reverse search for enumeration, *Discrete Appl. Math.*, **6** (1996) 21–46.

[3] V. Batagelj, Inductive classes of cubic graphs, in "Collection: Finite and infinite sets", *Colloq. Math. Soc. Janos Bolyai*, **37** (Eger, 1981) 89–101.

[4] S. Brandt, G. Brinkmann and T. Harmuth, The Generation of Maximal Triangle-free Graphs, in preparation.

[5] G. Brinkmann, Fast generation of cubic graphs, *J. Graph Theory*, to appear.

[6] G. Brinkmann, The combinatorial enumeration of hydrocarbon structures, in preparation.

[7] G. Brinkmann and A. W. M. Dress, A constructive enumeration of fullerenes, *J. Algorithms*, to appear.

[8] G. Brinkmann, B. D. McKay and C. Saager, The smallest cubic graphs of girth nine, *Combinatorics Probability and Computing*, **4** (1995) 317–330.

[9] R. D. Cameron, C. J. Colbourn, R. C. Read and N. C. Wormald, Cataloguing the graphs on 10 vertices, *J. Graph Theory*, **9** (1985) 551–562.

[10] J. H. Dinitz, D. Garnick and B. D. McKay, There are 526,915,620 nonisomorphic one-factorizations of K_{12}, *J. Combinatorial Designs*, **2** (1994) 273–285.

[11] I. A. Faradzev, Constructive enumeration of combinatorial objects. *Problemes Combinatoires et Theorie des Graphes Colloque Internat. CNRS 260.* CNRS Paris (1978) 131–135.

[12] R. J. Faudree and B. D. McKay, A conjecture of Erdős and the Ramsey number $r(W_6)$, *J. Combinatorial Math. and Combinatorial Comput.*, **13** (1993) 23–31.

[13] P. W. Fowler, D. Mitchell and G. Brinkmann, Electronic and Steric factors in the stability of proto-fullerene hydrocarbons, in "The chemical physics of the fullerenes 10 (and 5) years later" (ed. W. Andreoni; Kluwer, 1995).

[14] T. Grüner, R. Laue and M. Meringer, Algorithms for group actions: homomorphism principle and orderly generation applied to graphs, preprint 1995.

[15] Jun Hu, A. H. MacDonald and B. D. McKay, Correlations in two-dimensional vortex liquids, *Physical Review B*, **149** (1994) 15263–15270.

[16] A. Kerber, R. Laue, R. Hager and W. Weber, Cataloging graphs by generating them at random, *J. Graph Theory*, **14** (1990) 559–563.

[17] G. Kolesova, C. W. H. Lam and L. Thiel, On the number of 8×8 Latin squares, *J. Combinatorial Theory, Ser. A*, **54** (1990) 143–148.

[18] B. D. McKay, Practical graph isomorphism, *10th. Manitoba Conference on Numer-*

ical Mathematics and Computing (Winnipeg, 1980); *Congressus Numerantium*, **30** (1981) 45–87.

[19] B. D. McKay, nauty User's Guide (version 1.5), *Tech. Rpt. TR-CS-90-02, Dept. Computer Science, Austral. Nat. Univ.* (1990).

[20] B. D. McKay, Small graphs are reconstructible, submitted.

[21] B. D. McKay and S. P. Radziszowski, A new upper bound for the Ramsey number $R(5,5)$, *Australasian J. Combinatorics*, **5** (1991) 13–20.

[22] B. D. McKay and S. P. Radziszowski, The first classical Ramsey number for hypergraphs is computed, Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'91, San Francisco, (1991) 304–308.

[23] B. D. McKay and S. P. Radziszowski, There are no 4-(12,6,6) designs, in Computational and Constructive Design Theory (ed. W. Wallis) (Kluwer Publ., to appear).

[24] B. D. McKay and S. P. Radziszowski, Towards deciding the existence of 2-(22,8,4) designs, *J. Combin. Math. and Combin. Computing*, to appear.

[25] B. D. McKay and S. P. Radziszowski, $R(4,5) = 25$, *J. Graph Theory*, **19** (1995) 309–322.

[26] B. D. McKay and G. F. Royle, Constructing the cubic graphs on up to 20 vertices, *Ars Combinatoria*, **21A** (1986) 129–140.

[27] B. D. McKay and S. Sanjmyatav, Fast generation of cubic graphs by edge addition, in preparation.

[28] B. D. McKay and Zhang K. M., The value of the Ramsey number R(3,8), *J. Graph Theory*, **16** (1992) 99–105.

[29] M. Meringer, Erzeugung regulärer Graphen, Diplomarbeit, Univ. Bayreuth, 1996.

[30] R. C. Read, Every one a winner, *Annals Discrete Math.*, **2** (1978) 107–120.

[31] G. F. Royle, An orderly algorithm and some applications in finite geometry, submitted.

[32] D. R. Stinson and H. Ferch, 2000000 Steiner triple systems of order 19, *Math. Comp.*, **44** (1985), 533–535.