

Enumeration via Permutations

Sage Days 7: Combinatorics

Robert L. Miller

February 5, 2008

Requirements

Suppose that \mathcal{C} is a small category satisfying the following properties (a *small* category is one in which both $\text{ob}(\mathcal{C})$ and $\text{hom}(\mathcal{C})$ are sets).

- There exists a permutation group G , which acts on the set $\text{ob}(\mathcal{C})$, such that two objects $A \cong B$ are isomorphic if and only if there is an element $g \in G$ such that $g \cdot A = B$.
- There is a function of sets $o : \text{ob}(\mathcal{C}) \rightarrow \mathbb{Z}_{\geq 0}$, such that isomorphic objects map to the same integer. Call $o(A)$ the order of A . Also, assume $o^{-1}(n)$ is finite for $n \geq 0$.

The problem to be investigated is to formulate an algorithm which visits exactly one object from each isomorphism class up to a given order $N \geq 0$, called an N -traversal.

For example

- Let $G = S_\infty = \varinjlim_n (S_n)$, i.e. all permutations of finite subsets of \mathbb{N} . Then \mathcal{C} could be the set of all finite simple graphs with vertices in \mathbb{N} , with the order of a graph the number of vertices (alternatively, the number of edges).
- Let $G = \varinjlim_n (S_n \times S_n \times S_n)$, and let $\text{ob}(\mathcal{C})$ be the set of all Latin squares, $n \times n$ matrices with entries in $\{1, 2, \dots, n\}$ such that each row and column contains exactly one of each number. The action of $(g_1, g_2, g_3) \in S_n \times S_n \times S_n$ on a Latin square of size n is defined by g_1 permuting rows, g_2 permuting columns, and g_3 permuting numbers.

N -traversal over \mathcal{C} , naïve method

Algorithm 1

```
for n in  $0 \leq n \leq N$ :  
  for X in  $\mathcal{C}$  such that  $o(X) == n$ :  
    for Y already recorded of order n:  
      if  $X \cong Y$ : reject X, continue  
    record X
```

N -traversal over \mathcal{C} , naïve method

Algorithm 1

```
for n in 0 ≤ n ≤ N:  
  for X in  $\mathcal{C}$  such that  $o(X) == n$ :  
    for Y already recorded of order n:  
      if  $X \cong Y$ : reject X, continue  
    record X
```

Redundancies

- Storage of generated objects
- Visiting every object of \mathcal{C}
- Excessive isomorphism computation

The General Framework

Define the set of *children* $C(X)$ of X to be the set of $Y \in \text{ob}(\mathcal{C})$ such that there exists a monomorphism $X \hookrightarrow Y$ and such that $o(X) + 1 = o(Y)$.

Suppose that \mathcal{C} is such that for every $X \in \text{ob}(\mathcal{C})$, there is a sequence

$$X_0 \hookrightarrow X_1 \hookrightarrow \cdots \hookrightarrow X_{n-1} \hookrightarrow X_n = X,$$

such that $X_{i+1} \in C(X_i)$. Denote this sequence by $\mathcal{X}_{i=1}^n$ or simply \mathcal{X} , and let $T(\mathcal{C})$ be the set of all such sequences.

The General Framework

- Often we will restrict the definition of $C(X)$ to allow only certain kinds of inclusions, to make it easier to compute.
- For example, in the category of finite simple graphs (with order the number of vertices), it is not necessary to compute $C(X)$ if we allow all inclusions. We need only allow inclusions that map a vertex v to itself. This is equivalent to restricting to only subgraphs, instead of subgraph isomorphisms.
- This also greatly reduces the size of $T(\mathcal{C})$.

The Search Tree

Define a map $p : T(\mathcal{C}) \rightarrow T(\mathcal{C})$ as follows. Every element of $T(\mathcal{C})$ is a sequence of inclusions, so set

$$\begin{aligned} p(X_0 \hookrightarrow X_1 \hookrightarrow \dots \hookrightarrow X_{n-1} \hookrightarrow X_n) \\ := X_0 \hookrightarrow X_1 \hookrightarrow \dots \hookrightarrow X_{n-1} \end{aligned}$$

This gives the set $T(\mathcal{C})$ the structure of a rooted forest. In many situations, there will be only one object of order 0, and $T(\mathcal{C})$ will actually be a tree, justifying the notation. Call $T(\mathcal{C})$ the *search tree* for the category \mathcal{C} .

The Search Tree

Note that an object $X \in \text{ob}(\mathcal{C})$ will usually appear many times in the search tree, via different chains of subobjects. Thus there is a difference between “objects” X in \mathcal{C} and “nodes” \mathcal{X} in $T(\mathcal{C})$.

There is a natural surjection $T(\mathcal{C}) \twoheadrightarrow \text{ob}(\mathcal{C})$, taking $\mathcal{X}_{i=1}^n$ to X_n . Define the order on $T(\mathcal{C})$ by $o(\mathcal{X}_{i=1}^n) = n$.

Canonical Labeling

Via the group G , we can think of the objects of our category as *labeled objects*, namely the labels are the things that G is permuting. In the example of graphs, the labels are the vertices themselves. Different permutations of an object are different labelings of the same *unlabeled object*, or isomorphism class.

Canonical Labeling

This motivates the definition of a *canonical labeling* map $CL : \text{ob}(\mathcal{C}) \rightarrow \text{ob}(\mathcal{C})$, characterized by

- $A \cong B$ implies that $CL(A) = CL(B)$.
- $A \cong CL(A)$.

In other words, a canonical labeling is a choice of a unique isomorphism class representative for each class.

Orderly Generation

Suppose we have a canonical labeling map CL that satisfies the following.

- For each $\mathcal{X} \in T(\mathcal{C})$, the canonical labeling of \mathcal{X} is a node of $T(\mathcal{C})$. In other words, given any chain of inclusions

$$X_0 \hookrightarrow X_1 \hookrightarrow \dots \hookrightarrow X_n$$

such that $X_{i+1} \in C(X_i)$, we have $CL(X_{i+1}) \in C(CL(X_i))$. This allows us to define $CL : T(\mathcal{C}) \rightarrow T(\mathcal{C})$.

Orderly Generation

Then we have an algorithm for an N -traversal.

Algorithm 2

```
def orderly(object X):
    if CL(X) != X:
        return
    report X
    if order(X) < N:
        for Y in C(X):
            orderly(Y)
for X in o-1(0):
    orderly(X)
```

Orderly Generation

Proof.

As presented, the algorithm actually generates isomorphism types of *nodes*, not objects. By the assumption, any chain of inclusions is realized as a chain of inclusions of canonical objects. It is obvious from the algorithm that any such chain will be generated, and uniqueness of node types follows from uniqueness of $CL(X)$. \square

Orderly Generation

Proof.

As presented, the algorithm actually generates isomorphism types of *nodes*, not objects. By the assumption, any chain of inclusions is realized as a chain of inclusions of canonical objects. It is obvious from the algorithm that any such chain will be generated, and uniqueness of node types follows from uniqueness of $CL(X)$. \square

Drawbacks

- Isomorphism types of nodes, not objects.
- The assumption is implausibly difficult to efficiently implement.

Orderly Generation

Normally, this assumption is implemented by defining $CL(X)$ in terms of some kind of lexicographic ordering on objects, making $CL(X)$ an extremal element of its isomorphism class. If this is done in a way that also maximizes certain subobjects, then the assumption holds, and this ordering is why the scheme is called orderly generation, although the term orderly can refer to other ideas in the literature.

References

Orderly generation was developed independently by

- I. A. Faradžev, Constructive enumeration of combinatorial objects, in *Problèmes Combinatoires et Théorie des Graphes*, (Université d'Orsay, July 9-13, 1977), CNRS, Paris, 1978, pp. 131–135.
- R. C. Read, Every one a winner; or, How to avoid isomorphism search when cataloguing combinatorial congruences, *Ann. Discrete Math.* 2 (1978), 107–120.

An Algebraic Approach

In this approach, we consider the group G acting on the set $\text{ob}(\mathcal{C})$, and we want an orbit transversal of this action.

Definition

If G is a group acting on two sets Φ and Ψ , a *homomorphism of group actions* is a map $\rho : \Phi \rightarrow \Psi$ such that $\rho(gX) = g\rho(X)$ for all $X \in \Phi$ and all $g \in G$.

I won't say too much about this approach, because it is complicated, and there is an easier (for me) way. However, for an example of how this works...

An Algebraic Approach

Suppose we have a sequence of group actions connected by homomorphisms of group actions (going in either direction)

$$\Phi_1 \xrightarrow{\rho_1} \Phi_2 \xrightarrow{\rho_2} \Phi_3 \xrightarrow{\quad} \dots \xrightarrow{\rho_{n-1}} \Phi_n.$$

If we want an orbit traversal of Φ_1 , and an orbit traversal of Φ_n is easily available, then it is possible to lift or project (depending on direction) the traversal along each ρ_j .

An Algebraic Approach

Here is how lifting works:

Theorem

If G acts on Φ and Ψ , finite sets, and $\rho : \Phi \rightarrow \Psi$ is a homomorphism of group actions, then

- *for $A, B \in \rho(\Phi)$, the orbits on Φ intersected by $\rho^{-1}(A)$ and $\rho^{-1}(B)$ are either equal or disjoint, with equality iff $A \cong B$.*
- *for $A \in \rho(\Phi)$ and $B \in \rho^{-1}(A)$, the automorphism group of A , a subgroup of G , is equal to the union over $C \in \rho^{-1}(A)$ of the sets of isomorphisms from B to C .*

References

- A. Kerber, *Applied Finite Group Actions*, 2nd ed., Springer-Verlag, Berlin, 1999.
- A. Kerber and R. Laue, Group actions, double cosets, and homomorphisms: Unifying concepts for the constructive theory of discrete structures, *Acta Appl. Math.* 52 (1998), 6390.
- R. Laue, Construction of combinatorial objects A tutorial, *Bayreuth. Math. Schr.* 43 (1993), 5396.
- R. Laue, Constructing objects up to isomorphism, simple 9-designs with small parameters, in *Algebraic Combinatorics and Applications* (A. Betten, A. Kohnert, R. Laue, and A. Wassermann, Eds.), Springer-Verlag, Berlin, 2001, pp. 232260.

More Definitions

- We want the same general setup of orderly generation, but with more freedom, and less time spent computing $C(X)$.

More Definitions

- We want the same general setup of orderly generation, but with more freedom, and less time spent computing $C(X)$.
- An *augmentation* is an ordered pair of labeled objects (X, Y) , such that $Y \in C(X)$.

More Definitions

- We want the same general setup of orderly generation, but with more freedom, and less time spent computing $C(X)$.
- An *augmentation* is an ordered pair of labeled objects (X, Y) , such that $Y \in C(X)$.
- An *isomorphism* of augmentations $(W, X) \cong (Y, Z)$ is a permutation $\gamma \in G$ such that $\gamma \cdot W = Y$ and $\gamma \cdot X = Z$.
- Notice that $(X, Y) \cong (X, Z)$ implies that there is a $\gamma \in \text{Aut}(X)$ with $\gamma \cdot Y = Z$.

Weak Canonical Parent

- A *weak canonical parent* of a non-minimal object X is an object $w(X)$, such that $X \in C(w(X))$ and

$$X \cong Y \Rightarrow w(X) \cong w(Y).$$

Weak Canonical Parent

- A *weak canonical parent* of a non-minimal object X is an object $w(X)$, such that $X \in C(w(X))$ and

$$X \cong Y \Rightarrow w(X) \cong w(Y).$$

- As long as the following holds, we have an algorithm for generating all objects of interest: for each n , and every non-minimal object X of order n , there is a node $Y_{i=1}^n$ such that

$$X \cong Y_n \text{ and } Y_{n-1} \cong w(X).$$

Weak Canonical Parent

- We say that a node $X_{i=1}^n$ is generated by *weak canonical augmentation* if $X_{i-1} \cong w(X_i)$ for $i \leq n$.

Weak Canonical Parent

- We say that a node $\mathcal{X}_{i=1}^n$ is generated by *weak canonical augmentation* if $X_{i-1} \cong w(X_i)$ for $i \leq n$.
- Suppose that $\mathcal{X}_{i=1}^n$ and $\mathcal{Y}_{i=1}^n$ are nodes of $T(\mathcal{C})$, generated by weak canonical augmentation, such that $X_n \cong Y_n$. Then by our assumptions,

$$X_{n-1} \cong w(X_n) \cong w(Y_n) \cong Y_{n-1}.$$

This proves the effectiveness of the following algorithm.

Weak Canonical Augmentation

Algorithm 3

```
def weak_traverse(node X):  
    report X  
    possible_Z  $\leftarrow \emptyset$   
    for Z  $\in C(X)$ :  
        if  $X \cong w(Z)$ :  
            possible_Z  $\leftarrow$  possible_Z  $\cup \{Z\}$   
    Remove isomorphs from possible_Z (*)  
    for Z  $\in$  possible_Z:  
        weak_traverse(Z)
```

Canonical Parent

- The idea behind canonical augmentation is instead of ensuring that the objects generated are in canonical form, that the objects are generated, or augmented, in a canonical way.

Canonical Parent

- The idea behind canonical augmentation is instead of ensuring that the objects generated are in canonical form, that the objects are generated, or augmented, in a canonical way.
- A *canonical parent* of a non-minimal object X is an object $m(X)$ that satisfies a better property, which implies that it is a weak canonical parent:

$$\begin{aligned} X \cong Y &\Rightarrow (m(X), X) \cong (m(Y), Y), \text{ i.e.} \\ &\Rightarrow \exists \gamma \in G, \gamma \cdot X = Y \text{ and } \gamma \cdot m(X) = m(Y). \end{aligned}$$

Canonical Parent

- The idea behind canonical augmentation is instead of ensuring that the objects generated are in canonical form, that the objects are generated, or augmented, in a canonical way.
- A *canonical parent* of a non-minimal object X is an object $m(X)$ that satisfies a better property, which implies that it is a weak canonical parent:

$$\begin{aligned}
 X \cong Y &\Rightarrow (m(X), X) \cong (m(Y), Y), \text{ i.e.} \\
 &\Rightarrow \exists \gamma \in G, \gamma \cdot X = Y \text{ and } \gamma \cdot m(X) = m(Y).
 \end{aligned}$$

- We say that a node $\mathcal{X}_{i=1}^n$ is generated by *canonical augmentation* if at each step i , $(X_i, X_{i+1}) \cong (m(X_{i+1}), X_{i+1})$.

Canonical Augmentation I

If we traverse only those nodes which are generated from minimal objects by a chain of canonical augmentations, then note if $X \cong Y$ are both generated, then we have

$$(p(X), X) \cong (m(X), X) \cong (m(Y), Y) \cong (p(Y), Y),$$

which in particular implies that $p(X) \cong p(Y)$. If isomorphs have already been rejected on the parents, we can conclude that $p(X) = p(Y) =: Z$. Since $(Z, X) \cong (Z, Y)$, there is a $\gamma \in \text{Aut}(Z)$ such that $\gamma \cdot X = Y$. Thus, isomorph rejection is established in computing $\text{Aut}(Z)$.

Canonical Augmentation II

Algorithm 4

```
def traverse(node X):  
    report X  
    CX ← C(X)  
    for each orbit of CX under Aut(X):  
        select one representative Z  
        if (Z,p(Z)) ≅ (Z,m(Z)):  
            traverse(Z)
```

Implementing m given C

- Given a canonical labeling map CL , we can often easily define m . For example:

Implementing m given C

- Given a canonical labeling map CL , we can often easily define m . For example:
- In the category of finite simple graphs, let γ be the permutation taking G to $CL(G)$, and define G' to be $CL(G)$ minus the last vertex. Finally, define $m(G)$ to be $\gamma^{-1}G'$.

Already implemented in Sage

```
sage: for g in graphs(4):  
...     print g.characteristic_polynomial()  
x^4  
x^4 + x^2  
x^4  
x^4 + x^2  
x^4 + x^2  
x^4 + 1  
x^4 + x^2 + 1  
x^4 + 1  
x^4  
x^4 + x^2  
x^4 + 1
```

References

Canonical augmentation and canonical labeling:

- B. D. McKay, Practical graph isomorphism, *Congr. Numer.* 30 (1981), 45–87.
- B. D. McKay, Isomorph-free exhaustive generation, *J. Algorithms* 26 (1998), 306–324.

A good general reference for classification:

- P. Kaski, P. R. J. Östergård, *Classification Algorithms for Codes and Designs*, Springer-Verlag, Berlin, 2006.