[Sto12a, Stolee] and [Sto12b, Stolee]

Computational Combinatorics and Canonical Deletions

Hassam

Outline

Section 1

Motivation

Generating Objects

- We want to count *things*
- We don't want to miss any
- We don't want to count more than we need to

Generalizing Combinatoric Search

- Start at a base object
- Augment the object
- Watch out for symmetries (*isomorphisms*)

Finding all Permutations

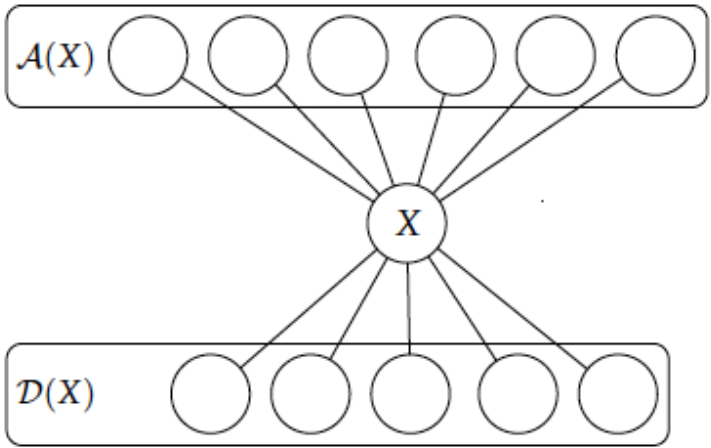
Let's find all permutations of the list [1, 2, 3]

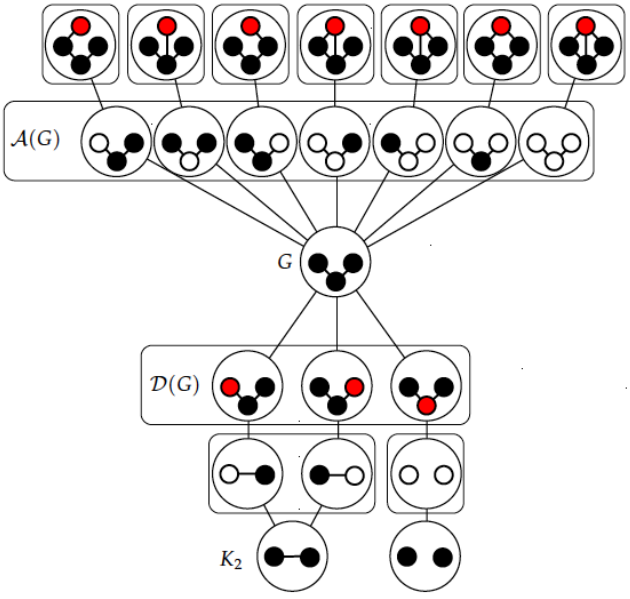
- What's our base object?
- Empty list: []
- What do we add next?
  - Items: 1, 2, 3
- Generate:
  - [1], [2], [3]
  - [1, 2], [1, 3], [2, 1], [2, 3], [3, 1], [3, 2]
  - [1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]

What breaks if we do this with subsets?

Section 2

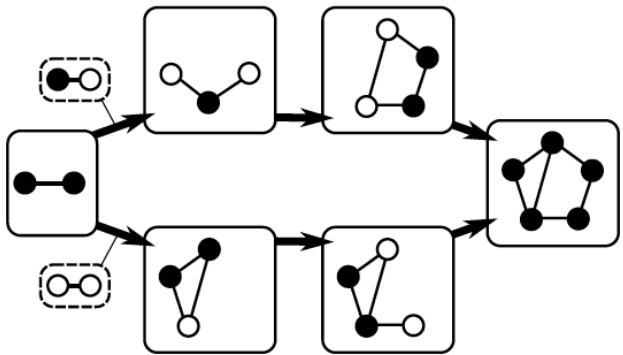
Augmentations and Deletions


$\mathcal{A}(X)$
X
$\mathcal{D}(X)$


$\mathcal{A}(G)$
G
$\mathcal{D}(G)$
K_2

Removing symmetries (isomorphisms)

What if we filtered out local isomorphisms? Is that sufficient?



Section 3

Canonical Deletions

The Problem

We have a way to label every graph and check if they are isomorphic, but this is slow, especially as our search space expands. Going up our list of augmentations, our search space explodes.

What if we worked down our search space?

Canonical Labels

In the process of checking whether two graphs are isomorphic, we generate a canonical label. We can define a canonical labelling as a function such that if $g \cong g'$ then $c(g) = c(g')$, and each element in the co-domain has a "name".

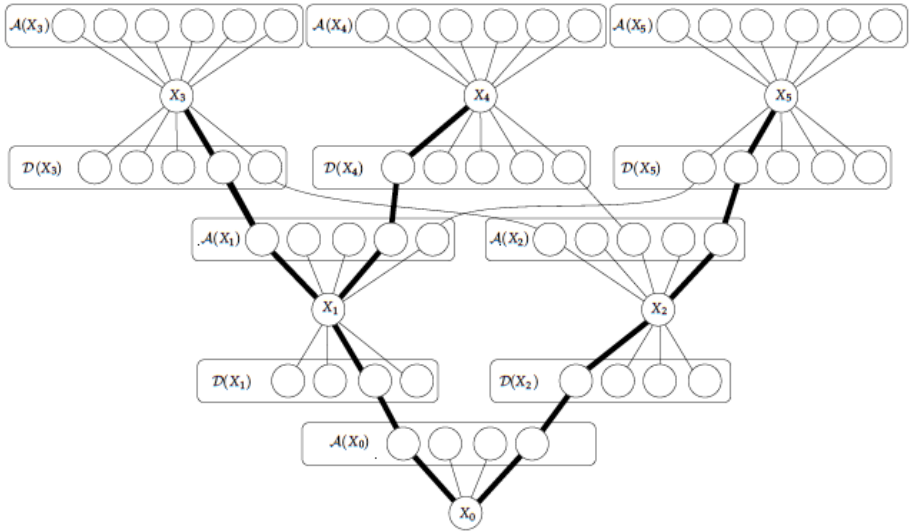
What's an obvious canonical labelling for subsets?

Canonical Deletions

The key idea: Every single canonical labeling has a specific set of deletions and augmentations associated with it. What if we chose one of the deletions to be the correct one?

A Subset Example

- Let's define the canonical labelling for a set to be the elements of the set in lexicographic order.
- Let's define the canonical deletion for a set to be removing it's smallest lexicographic element.
- What's the canonical deletion of $\{3, 4, 1, 2\}$?
- $\{2, 3, 4\}$


$\mathcal{A}(X_3)$
$\mathcal{A}(X_4)$
$\mathcal{A}(X_5)$
X_3
X_4
X_5
$\mathcal{D}(X_3)$
$\mathcal{D}(X_4)$
$\mathcal{D}(X_5)$
$\mathcal{A}(X_1)$
$\mathcal{A}(X_2)$
X_1
X_2
$\mathcal{D}(X_1)$
$\mathcal{D}(X_2)$
$\mathcal{A}(X_0)$
X_0

Canonical Deletion Algorithm

```
    CanonicalDeletion(n, X):
 1:  if Bad(X):                ⟨⟨Short Circuit Optimization⟩⟩
 2:      return
 3:  if Solution(X):
 4:      output X
 5:  for all augmentations A ∈ 𝒜(X), up to isomorphism:
 6:      D ← δ(A)              ⟨⟨Get the corresponding deletion⟩⟩
 7:      Y ← 𝒟⁻¹(D)            ⟨⟨Find the object that corresponds to that deletion⟩⟩
 8:      D′ ← del(Y)           ⟨⟨Compute its canonical deletion⟩⟩
 9:      if D ≅ D′:            ⟨⟨Check that they match⟩⟩
10:          call CanonicalDeletion(n, Y)
11:  return
```

A Specific Example

- Let us consider a specific example: *Generating Graphs* with n nodes
- Restrictions:
  - $\Delta(G) \leq 4$: Nodes have maximum degree 4
  - $\chi(G) \leq 3$: Nodes have chromatic color of at most 3

A Specific Example

```
GraphCanonicalDeletion(n, G):
1:  if Δ(G) ≥ 5 or χ(G) ≥ 4 or n(G) > n:
2:      return
3:  if n(G) ≡ n and Δ(G) ≤ 4 and χ(G) ≤ 3:
4:      Output G
5:  for all S ⊆ V(G):
6:      H ← G + v_S            ⟨⟨Augment the graph⟩⟩
7:      (H, v') ← del(H)       ⟨⟨Compute its canonical deletion⟩⟩
8:      if v_S and v' are the same vertex:
9:          call CanonicalDeletion(n, H)
10: return
```

So what?

What guarantees do we have?

1. Every object that has the desired property we are looking for is visited only once
2. We can use pruning to avoid visiting any objects that don't have this property more than once
3. Most importantly, this process is entirely local. We can parallelize it.

Section 4

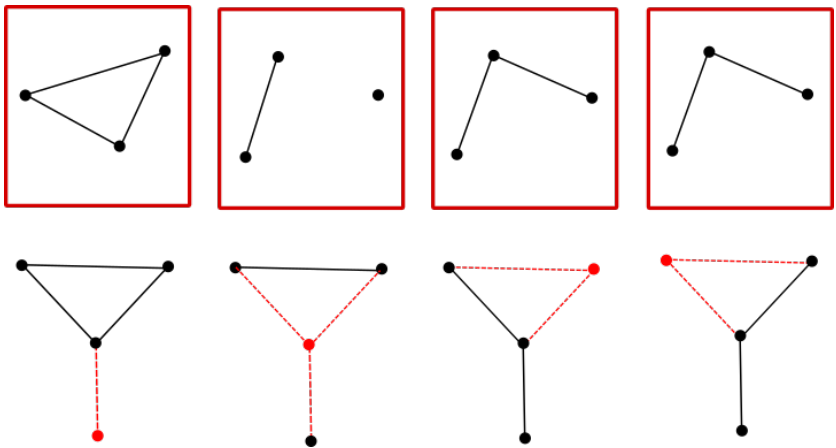
The Reconstruction Conjecture

An ~~Open~~(?) Problem in Graph Theory

The reconstruction conjecture was posed by Kelly and Ulam as something to play with while Ulam was working on the atomic bomb. There are some papers claiming to have proven this conjecture true, but they go above my head and I can't find any reputable source verifying them.

Graph Decks

Given a graph G, its deck is the multi-set of graphs H such that H is isomorphic to a single vertex deleted subgraph of G.



Are decks all you need?

In other words: if G and G' have isomorphic decks, are G and G' isomorphic?

Naive Testing

We could generate every graph of order n and do pairwise comparisons of their decks. This is not a lot of programming, sage, geng, etc. Generating graphs is not too bad, we will probably have to do this anyway. This is a lot of comparisons though.

Testing smarter

Suppose we had two distinct graphs with the same deck, then those two graphs must come from the same set of augmentations/deletions. To check that a n vertex graph is reconstructible, we only need to compare it to the augmentations that come from its canonical deletion!

In practice, we go from n^2 comparisons to approximately $n \log n$ comparisons.

Results

McKay, when he first created the Nauty library, was able to show that the theorem holds for all graphs of up to order 11. This was in 1997. Can we do better using modern hardware? I think we can. [McK97]

Update: in 2021, McKay repeated this inquiry, with a few additions, using modern computers. He was able to show the reconstruction theorem holds for all graphs up to order 13. It took him 1.5 years of compute-time though. [McK21]

Bibliography


Brendan D. McKay.
Small graphs are reconstructible.
Australas. J Comb., 15:123–126, 1997.


Brendan D. McKay.
Reconstruction of small graphs and digraphs, 2021.


Derrick Stolee.
Introduction to canonical deletion, August 2012.
https://computationalcombinatorics.wordpress.com/2012/08/13/canonical-deletion/.


Derrick Stolee.
Small graphs are reconstructible, August 2012.
https://computationalcombinatorics.wordpress.com/2012/08/29/small-graphs-are-reconstructible/.