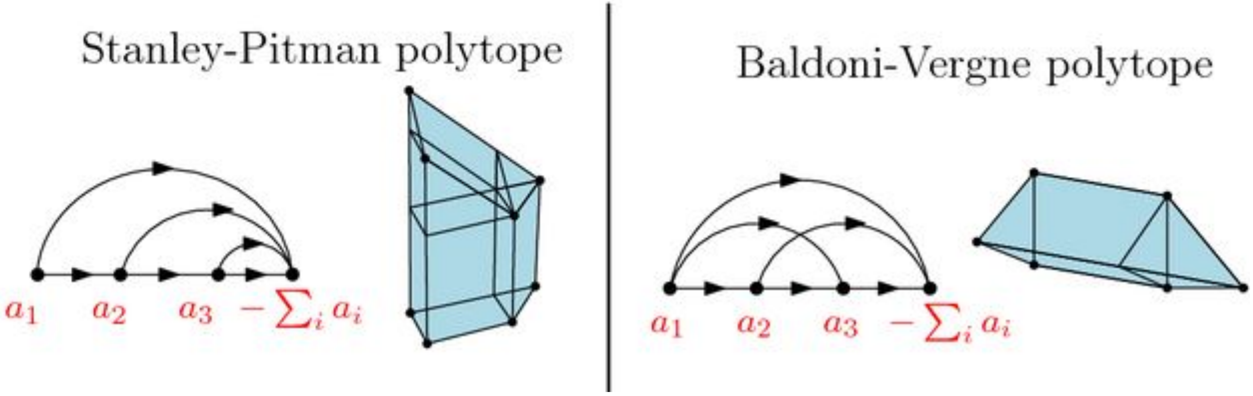
The Geometry and Limits of Young Partition Flow Polytopes

By Advay Goel

What Are Flow Polytopes?

- Special shapes that exist in higher dimensions
    - 2-D, 3-D, 9-D, 29-D, etc.



Why Do We Study Them?

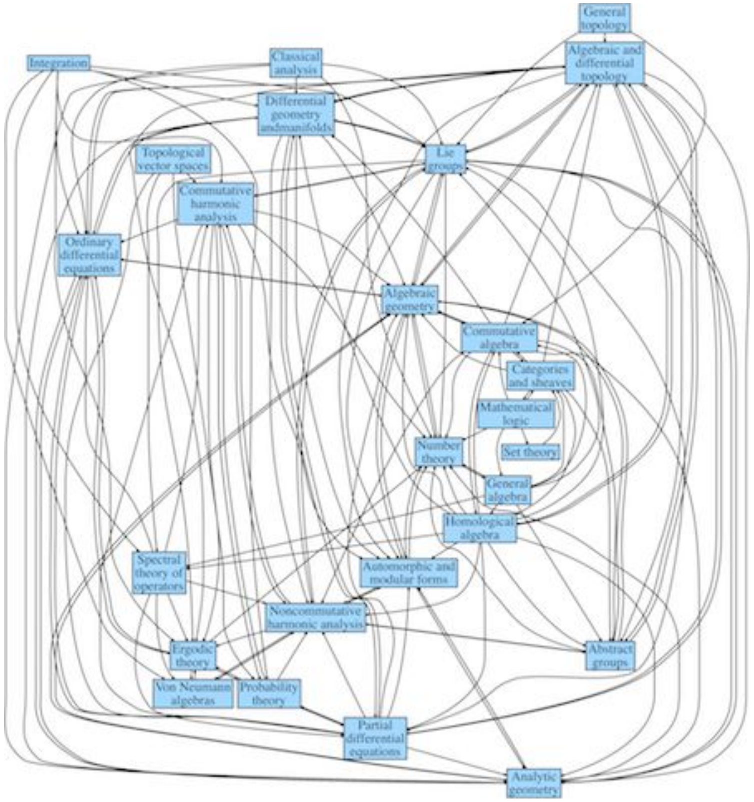
- Artificial Intelligence algorithms make smart approximations to save time
    - Combinatorial Optimization
- Oftentimes, combinatorial optimization problems can be modelled by flow polytopes

Understand flow polytopes → Improve combinatorial optimization methods → Make AI algorithms more accurate and efficient

The Research Problem

- How do we connect flow polytopes to other fields of math?
- How can we use any newly formed connections to gain more insight about flow polytopes?

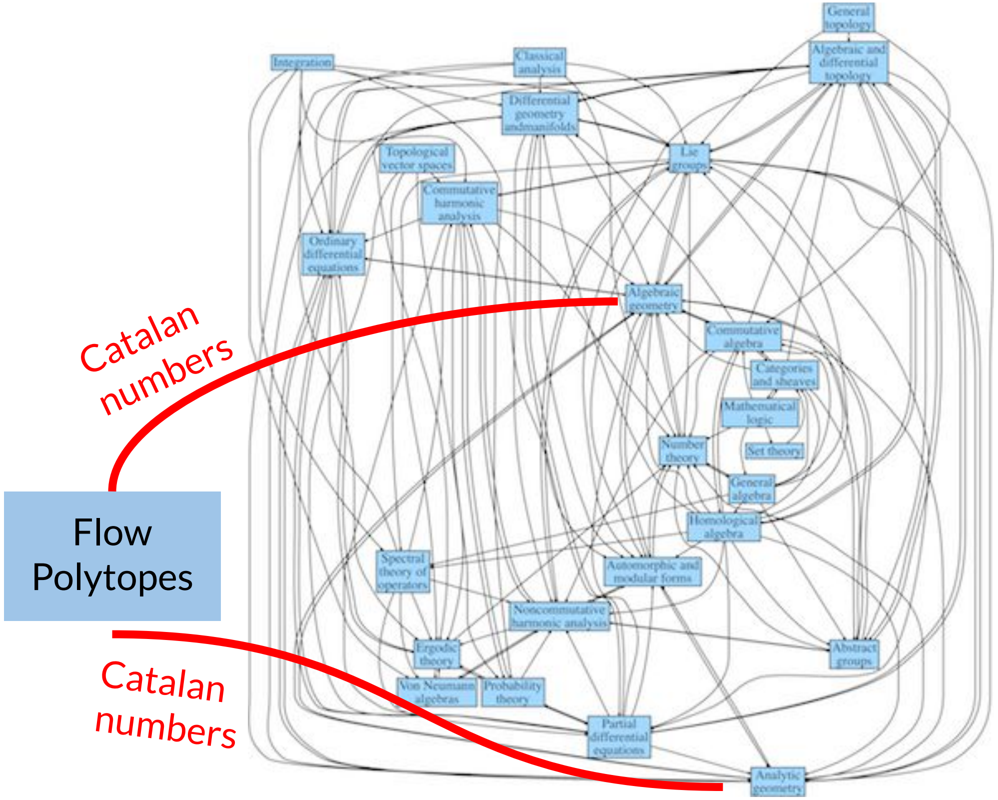
Flow Polytopes



Volumes

- 1999: Chan-Robbins-Yuen Conjecture published
    - Catalan Numbers

- If we understand how and why the Catalan numbers appear, we can draw connections using the Catalan numbers as a bridge



My Solution

- Triangles are the building blocks of 2-D shapes
- This idea extends to higher dimensions
  - Simplex
- What do the simplex structures of flow polytopes look like and how does this affect the volume/relate to Catalan numbers?

```java
    //this is the main recursive function used to get the volume
    static String getVolume(FlowPolytope f) {
        String volume = "";
        //nInequality is a data structure I constructed. It's there because the cases, when joined into one line, form a huge inequality with many (n) parts.
        //the list of nInequalities polytopeDefinition is just all of the inequalities in the flow polytope
        ArrayList<nInequality> polytopeDefinition = new ArrayList((ArrayList<nInequality>) f.getInequalities());

        //base case 1: if the flow polytope is empty, return an empty string (no volume)
        if(polytopeDefinition.size() == 0) {
            return "";
        }
        //base case 2: if the flow polytope has only 1 inequality, that inequality must be type A --> the volume is 1/x! * a_1^(x) where x is the # of terms on the LHS of the Type A
        if (polytopeDefinition.size() == 1) {
            return " + \\frac{1}{" +factorial(polytopeDefinition.get(0).getTerms().get(0).size()) + "} a_1^{" + polytopeDefinition.get(0).getTerms().get(0).size() +  "}";
        }

        //number of inequalities in the flow polytope
        int totalInequalities = polytopeDefinition.size();

        //we work with the inequalities from bottom to up. that's why "currentInequality" is the last inequality in f.
        nInequality currentInequality = f.getInequalities().get(totalInequalities -1 );

        //this list just contains all of the cases for currentInequality. It calls the getCases() function which is defined below
        ArrayList<nInequality> cases = new ArrayList(getCases(currentInequality));

            //in the following section, we solve each case separately.
            //We see what it adds to the volume and how it affects the upper inequalities.
            //Based on that, we then create a new flow polytope called "newF" that has the updated upper inequalities based on the case and we remove currentInequalities from it
            //this keeps going on recursively so that we are able to cover all of the subcases for each major case.
            for (nInequality ineq: cases) {

                //inequalitiesLeft is all of the inequalities left in polytope f excluding the current inequality.
```

Problems | Javadoc | Declaration | Console

```
<terminated> Algorithm [Java Application] /Users/premg/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.2.v20220201-1208/jre/bin/java  (Feb 27, 2023, 11:24:31 PM – 11:24:34 PM) [pid: 26901]
\frac{1}{24} a_2^{4} \left ( \frac{1}{120} a_1^{5}\right ) + \frac{1}{362880} a_1^{9} +  a_2 \left ( \frac{1}{40320} a_1^{8}\right ) + \frac{1}{2} a_2^{2} \left ( \frac{1}{5040} a_1^{7}\right )
```

Final Results

- Using my building block (simplex) analysis and code, I was able to shed light on why the Catalan numbers appear

- Now, researchers can use this to form bridges between flow polytopes and other fields and use that information to improve combinatorial optimization techniques