Isomorph-Free Exhaustive Generation in SAT Solving

Curtis Bright

University of Windsor

February 17, 2022

Tutorial at the Dagstuhl Seminar

New Perspectives in Symbolic Computation and Satisfiability Checking

Roadmap

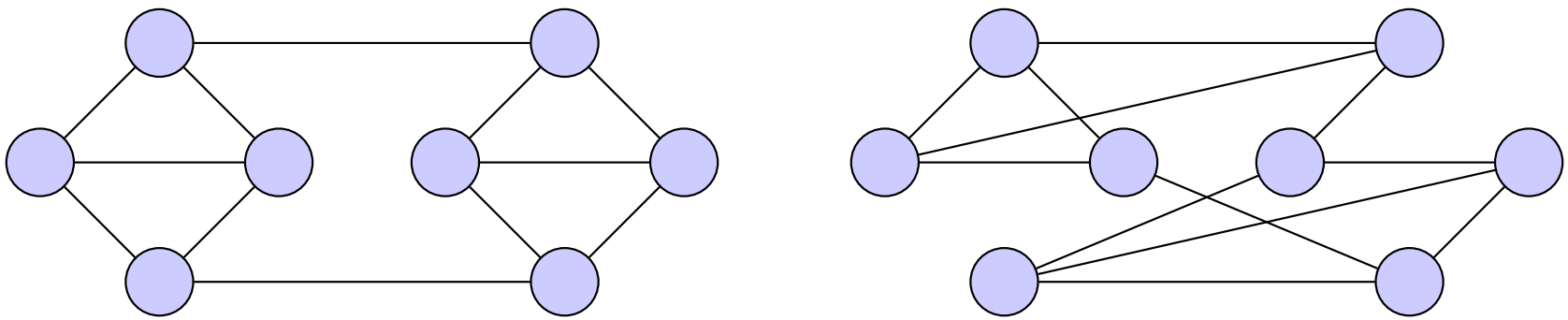
Past: What is isomorph-free exhaustive generation?

Present: Bringing isomorph-free generation to SAT solvers.

Future: Release the untapped potential of isomorph-free SAT solving!

Isomorphisms

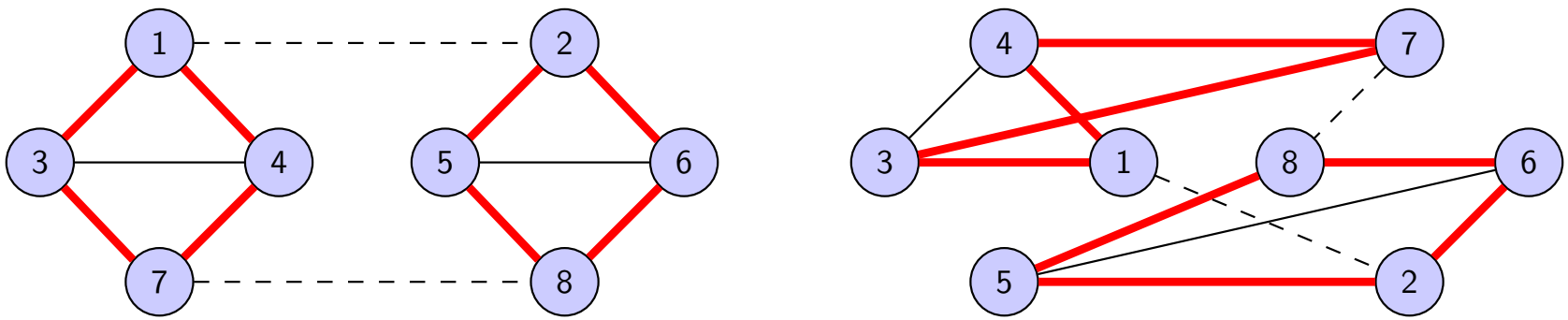
Two combinatorial objects are isomorphic if one can be transformed into the other through a mapping that preserves the structure of the object.



Two graphs both with eight vertices, but different edges.

Isomorphisms

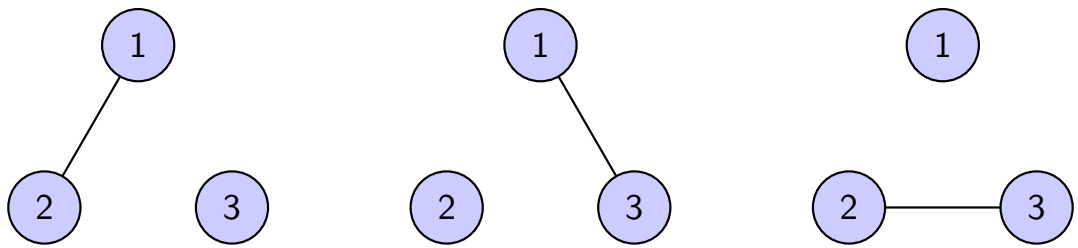
Two combinatorial objects are isomorphic if one can be transformed into the other through a mapping that preserves the structure of the object.



Morally speaking, they are the same graph!

Isomorphisms

When generating combinatorial objects we really only care about generating them *up to isomorphism*. Unfortunately, objects usually have many isomorphic representations.



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The Importance of Isomorph-Free Generation

For example, a graph with n vertices can have up to $n!$ distinct isomorphic adjacency matrices. This makes the size of the search space for graphs *much* larger than it needs to be.

To exhaustively generate combinatorial objects it is of utmost importance to detect and remove isomorphic copies of objects as early as possible.

Methods for Isomorph-Free Generation

There are a number of methods for isomorph-free exhaustive generation. In this tutorial I'll cover two approaches:

Recorded objects: All intermediate objects are recorded up to isomorphism as the search progresses. (Folklore method)

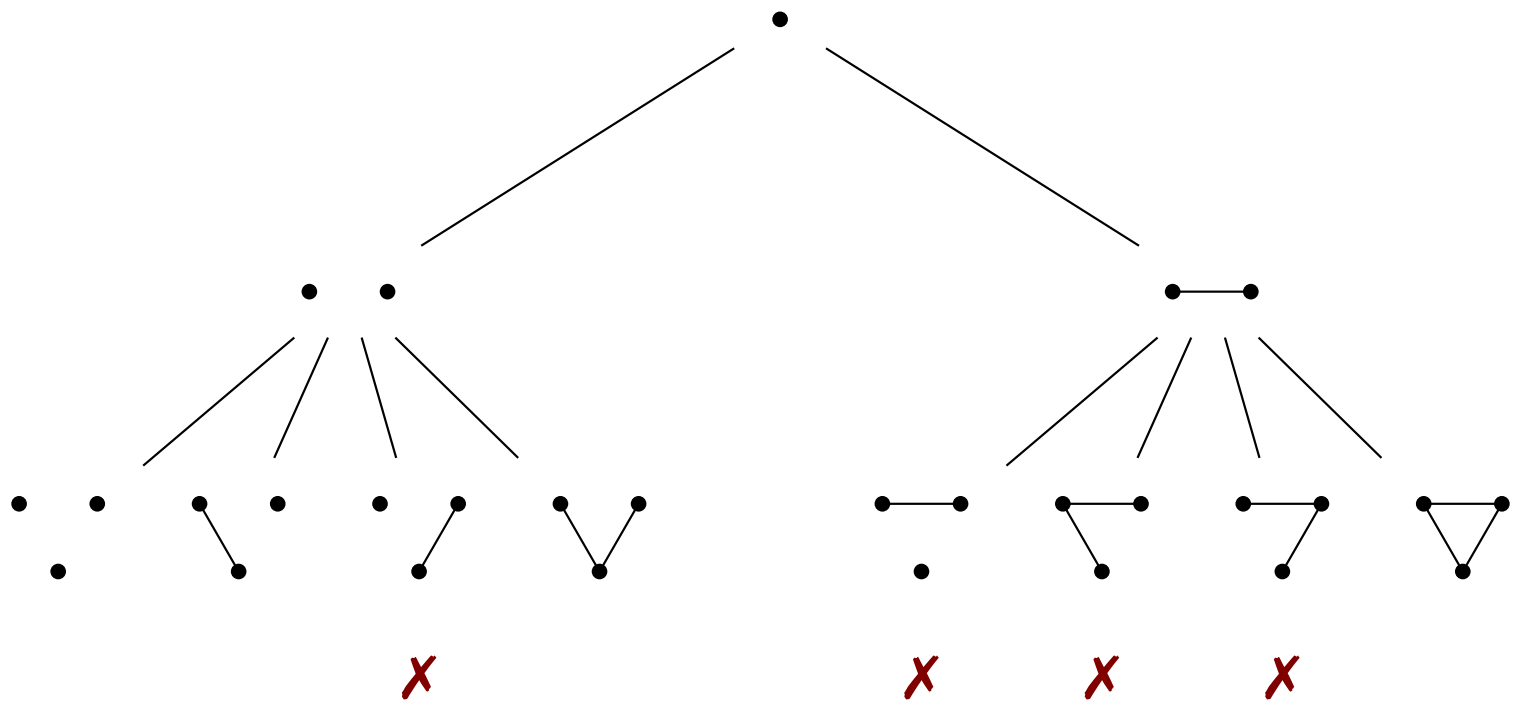
Orderly generation: Only "canonical" intermediate objects are recorded as the search progresses. Developed independently by Igor Faradžev and Ronald Read in 1978.

Recorded Objects Method

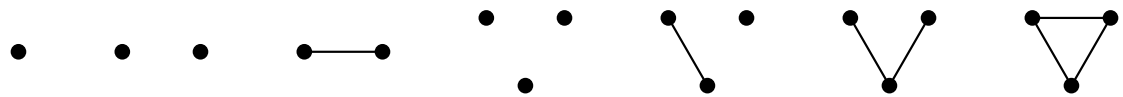
As the search progresses all intermediate objects are *recorded*.

An intermediate object is immediately rejected if it is ever found to be isomorphic to a previously recorded intermediate object.

Example: Generating Graphs Using Recorded Objects



Recorded objects:



Implementing Recorded Objects

To implement the "recorded objects" method we need a fast way of checking if an intermediate object is isomorphic to a previously recorded object.

In practice, a "certificate" of the isomorphism class of each new object is computed.[1]

A hash table is used to store the certificates of each object. If the hash of a certificate is *not* in the table then the object is genuinely new and recorded.

[1] B. McKay, A. Piperno. Practical Graph Isomorphism, II. *Journal of Symbolic Computation*, 2014.

Implementing Recorded Objects II

If a hash *is* in the table of recorded objects this does not imply the object is genuinely new as there may be a hash collision.

Only if the certificate itself has previously been recorded can an object safely be rejected.

Thus, it is not enough to only store hashes—the certificates themselves must be stored.

Pros and Cons of Recorded Objects

- Relatively easy to implement, assuming that you have access to a isomorphism certificate generator.

- Requires storing the certificates of every recorded object and these can be relatively large.

- Does not parallelize well, since all processors need access to the table of recorded objects.

Orderly Generation Method

Only "canonical" intermediate objects are recorded. The notion of canonicity must be defined so that:

1. Every isomorphism class has exactly one canonical representative.

2. If an object is canonical then its parent in the search tree is also canonical.

In other words, if an object is *not* canonical then all of its children are not canonical. Thus, all intermediate noncanonical objects can be rejected.

Canonicity Example

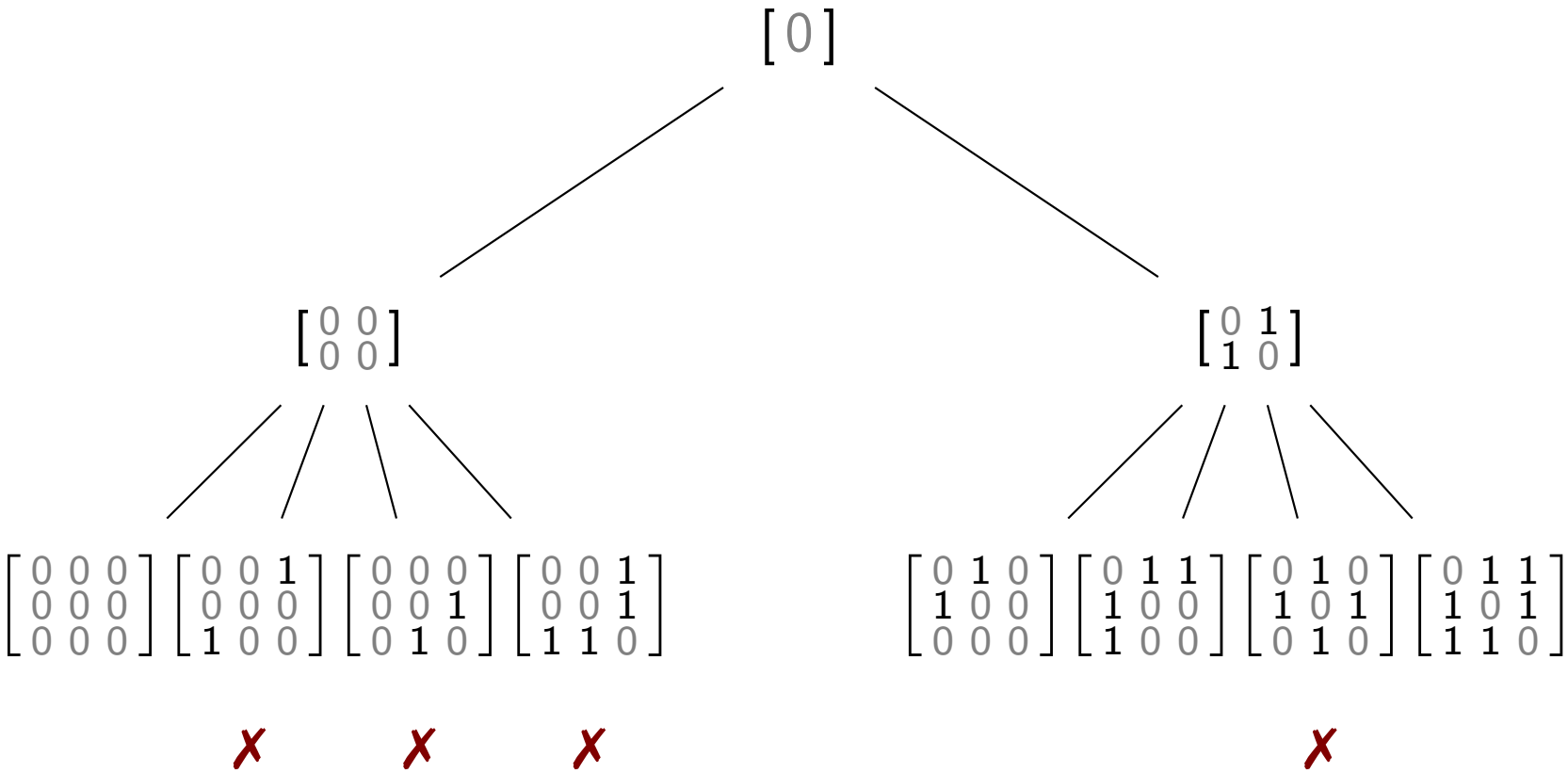
An adjacency matrix of a graph can be considered *canonical* if its rows concatenated together are lexicographically greatest (among all of the adjacency matrices isomorphic to the same graph).

For example,

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \text{ and } \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

are all isomorphic adjacency matrices but only the first is canonical.

Example: Orderly Generation of Graphs



Canonical adjacency matrices:

$[0]$ $\begin{bmatrix}0&0\\0&0\end{bmatrix}$ $\begin{bmatrix}0&1\\1&0\end{bmatrix}$ $\begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}$ $\begin{bmatrix}0&1&0\\1&0&0\\0&0&0\end{bmatrix}$ $\begin{bmatrix}0&1&1\\1&0&0\\1&0&0\end{bmatrix}$ $\begin{bmatrix}0&1&1\\1&0&1\\1&1&0\end{bmatrix}$

Implementing Orderly Generation

To perform orderly generation we need a canonicity checking method which in general is a difficult problem.

However, verifying that a matrix is *noncanonical* is often easy—it requires finding a single permutation of the vertices which gives a lexicographically larger adjacency matrix.

Implementing Orderly Generation II

In practice, a backtracking algorithm can be used to determine canonicity by iterating through permutations on the vertices:

- Permute the vertices of the graph.
- Compare the rows of the permuted adjacency matrix to the rows of the original adjacency matrix:
  - If they are *greater* then matrix is not canonical.
  - Otherwise backtrack and try a new permutation.
- The matrix is canonical if there are no more permutations.

It is possible to backtrack early (after only a few vertices have been permuted) if the first rows of a *smaller* matrix is uncovered.

Pros and Cons of Orderly Generation

- Canonicity testing is often fast, but it can sometimes can be relatively slow (especially when a matrix is canonical).

- Does not require memory to record intermediate objects as the search progresses.

- Parallelizes easily. Separate processors can search separate parts of the search space without needing to communicate.

SAT Approaches

The satisfiability community has their own extensive literature on methods for dealing with highly symmetric search spaces.

A typical approach is to add "symmetry breaking" constraints that remove as many isomorphic solutions as possible.

For example, you can order the rows of an adjacency matrix of a graph lexicographically.[2] However, typically many distinct isomorphic representations still exist, like $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$.

[2] M. Codish, A. Miller, P. Prosser, P. Stuckey. Constraints for symmetry breaking in graph representation. *Constraints*, 2019.

SAT with Isomorph-free Exhaustive Generation

Remarkably, there seems to be almost no work using isomorph-free exhaustive generation methods in a SAT solver.

I am not the first to notice this.[3] I hope this will change as a result of initiatives like the SC-square project.

I will now talk about two applications where using isomorph-free generation inside a SAT solver is extremely useful.

[3] T. Junttila, M. Karppa, P. Kaski, J. Kohonen. An adaptive prefix-assignment technique for symmetry reduction. *Journal of Symbolic Computation*, 2020.

Application I: Lam's Problem

Based on work with
Kevin Cheung, Brett Stevens, Ilias Kotsireas, and Vijay Ganesh.

History

Mathematicians tried to derive Euclid's "parallel postulate" from his other axioms for geometry for over two thousand years.

History

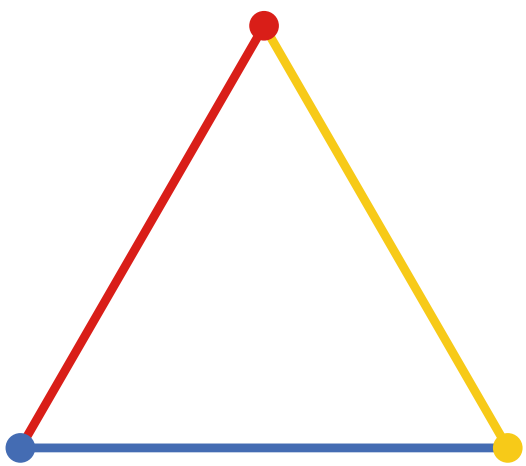
Mathematicians tried to derive Euclid's "parallel postulate" from his other axioms for geometry for over two thousand years.

The discovery of alternative geometries in the 1800s showed this is impossible!

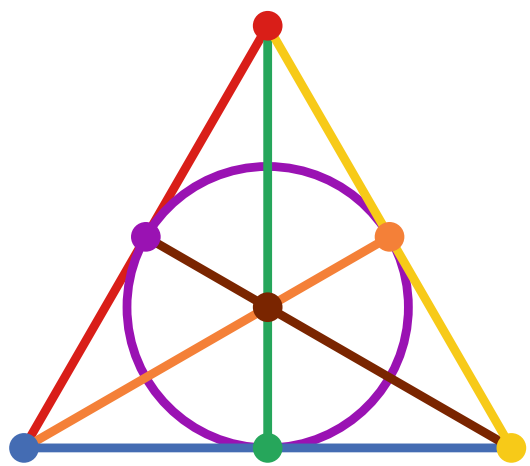
Finite Projective Planes

Finite projective planes satisfy the following axioms:

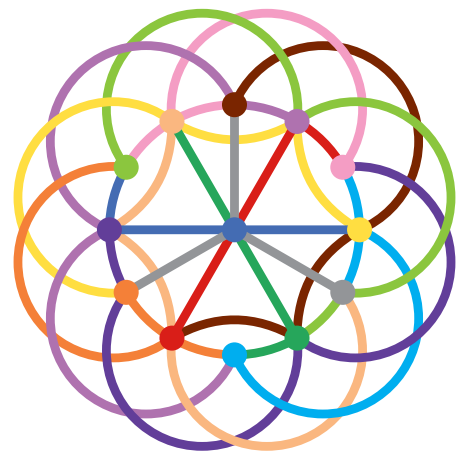
- Every pair of points define a unique line.
- Every pair of lines meet at a unique point.
- Every line contains $n + 1$ points for some *order* n.



order 1



order 2



order 3

Projective Planes of Small Orders

1	2	3	4	5	6	7	8	9	10
✓	✓	✓	✓	✓	✗	✓	✓	✓	✗

Lam's problem

Computer Science team solves centuries-old math problem

And they had to search through a thousand trillion combinations to do it

Simply put . . .

Whew! To complete a mathematical investigation as complicated as the one recently accomplished by a team from the faculty of Engineering and Computer Science, every human being on earth would have to do 50,000 complex calculations.

The team, made up of Computer Science's Clement Lam, John McKay, Larry Thiel and Stanley Swiercz, took three years to solve a problem which had stumped mathematicians since the 1700s.

The problem: To find out whether "a finite projective plane of the order of 10" can exist.

Charles Bélanger

Resolution of Lam's Problem

Lam et al.[4] used custom-written software to show that a projective plane of order ten does not exist.

We must simply trust their searches ran to completion—the authors were upfront that mistakes were a real possibility.

We generated the first certifiable resolution of Lam's problem[5] and found they had missed some intermediate objects.

---

[4] C. Lam, L. Thiel, S. Swiercz. The Nonexistence of Finite Projective Planes of Order 10. *Canadian Journal of Mathematics*, 1989.

[5] C. Bright, K. Cheung, B. Stevens, I. Kotsireas, V. Ganesh. A SAT-based Resolution of Lam's Problem. *AAAI 2021*.

SAT Encoding

A projective plane of order n is equivalent to a quad-free $(0,1)$-matrix with $n+1$ ones in each row and column. A *quad-free* matrix contains no rectangle with 1s in the corners.

1	1	0
1	0	1
0	1	1

order 1

1	1	0	1	0	0	0
0	1	1	0	1	0	0
0	0	1	1	0	1	0
0	0	0	1	1	0	1
1	0	0	0	1	1	0
0	1	0	0	0	1	1
1	0	1	0	0	0	1

order 2

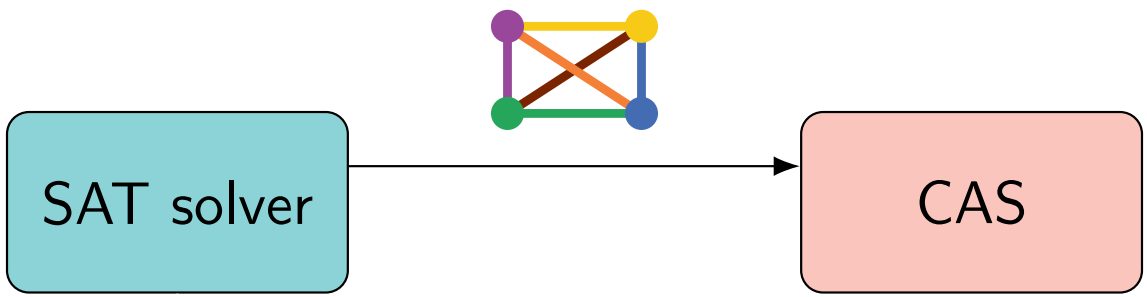
1	0	0	0	1	0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	1	0	1	0	0	0
0	1	0	0	0	1	1	0	0	1	0	0	0
1	0	0	0	0	1	0	1	0	0	1	0	0
0	1	0	1	0	0	0	0	1	0	1	0	0
0	0	1	0	1	0	1	0	0	0	1	0	0
1	0	0	1	0	0	1	0	0	0	0	1	0
0	1	0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	1	0	0	1	0	0	1	0
0	0	0	1	1	1	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	1	0	0	0	1
1	1	1	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	1	1	1

order 3

These constraints are encoded in Boolean logic along with symmetry breaking constraints. However, many isomorphic intermediate solutions (partial projective planes) still exist.

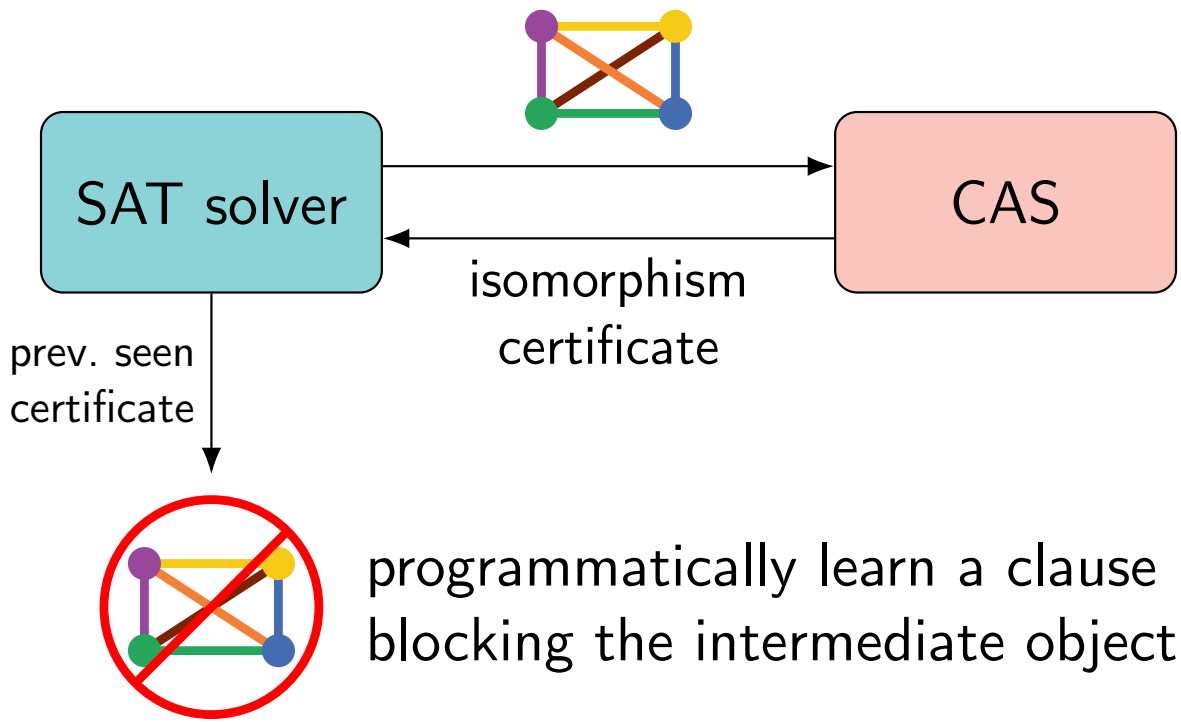
Recorded Objects Isomorph Rejection in SAT

During the search the SAT solver will find partial solutions (by finding complete definitions for the first few lines of the plane)...



Recorded Objects Isomorph Rejection in SAT

During the search the SAT solver will find partial solutions (by finding complete definitions for the first few lines of the plane)...



Lam's Problem Results

In the hardest case of Lam's problem, the first step is to find all possibilities for the first 19 points of the plane. These SAT instances have over 56 million solutions.

It can quickly be determined that only 0.5 million of these are distinct up to isomorphism, but this indicates many unbroken symmetries remain...

Using isomorph rejection *during* the search (instead of at the end) generates the solutions **150 times** faster.

Application II: Kochen–Specker Systems

Based on work with
Brian Li and Vijay Ganesh.

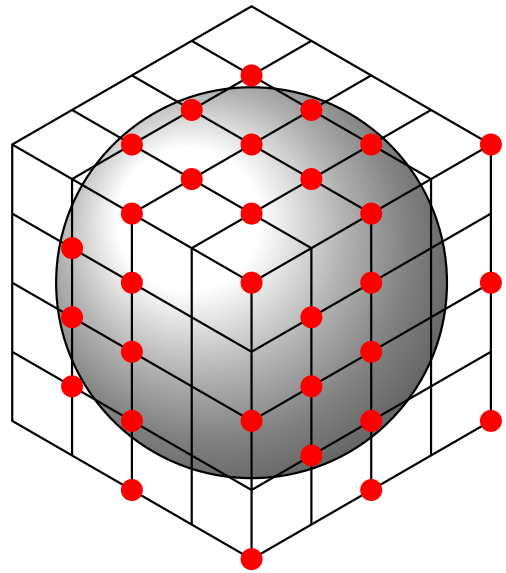
The Free Will Theorem

In 2004, John Conway and Simon Kochen proved the "Free Will Theorem". It roughly states that if humans have free will then so do elementary particles.

Their proof uses a configuration of three dimensional vectors called a Kochen–Specker (KS) system.

Example KS System

If an agent is *free* to choose to measure the spin of a particle in the following 31 directions...



...then the particle is *free* to choose its spin in these directions.

Subject to the laws of quantum mechanics it is impossible to consistently assign outcomes for the spin in all 31 directions.

Image Source: S. Uijlen, B. Westerbaan

Can We Do Better?

It is unknown if there are fewer than 31 directions with this property.

The current best known result is that at least 22 directions are required.[6]

This was shown by translating a hypothetical 21-direction KS system into a graph and using isomorph-free exhaustive generation on 21-vertex graphs. The computation took 75 CPU years.

[6] S. Uijlen, B. Westerbaan. A Kochen-Specker System Has at Least 22 Vectors. *New Generation Computing*, 2016.

Reduction to SAT

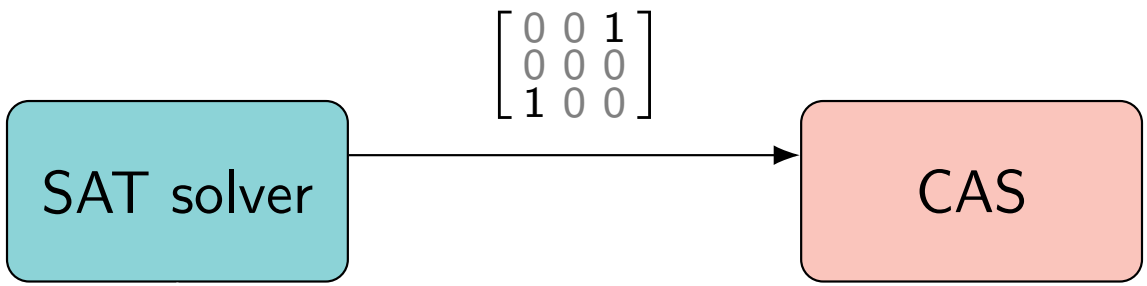
With some cleverness, many of the properties a "KS graph" must satisfy can be reduced to Boolean logic.

We found that a SAT solver performs better than isomorph-free generation. However, a SAT solver generates many copies of the same graph.

Thus, we use a hybrid SAT and isomorph-free generation approach.

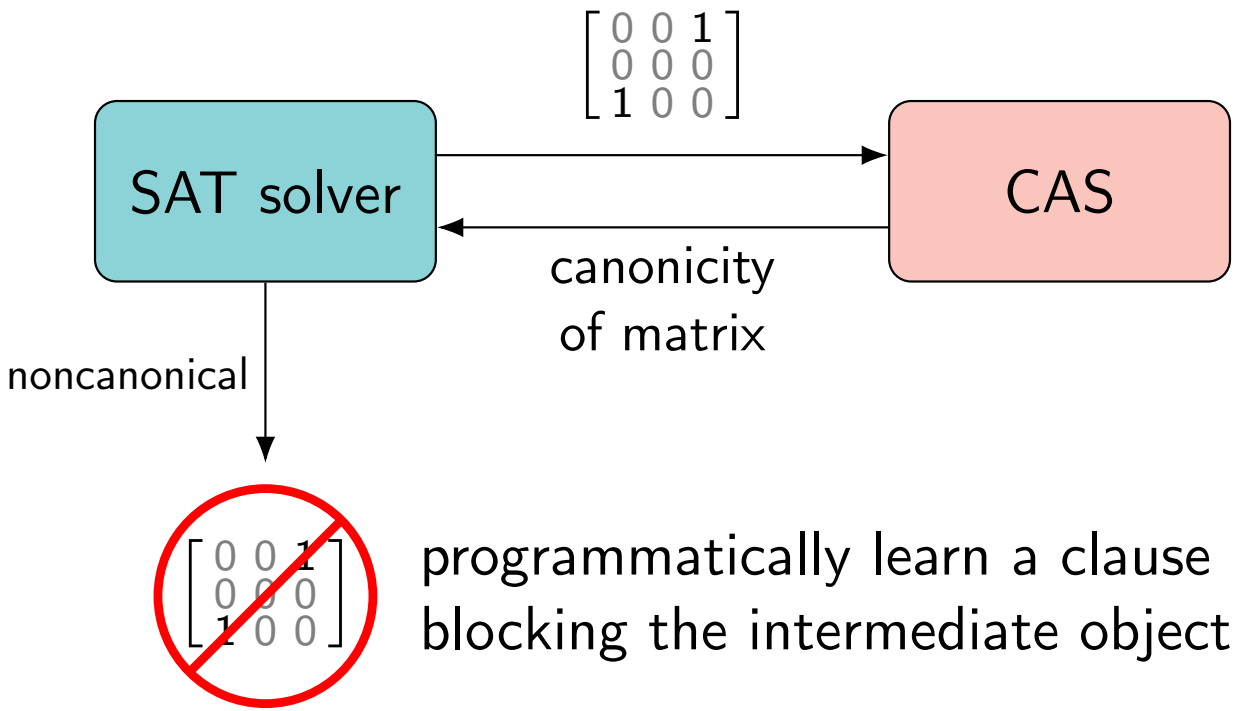
Orderly Generation in SAT

During the search the SAT solver will find partial solutions (complete definitions for the edges in some subgraphs)...



Orderly Generation in SAT

During the search the SAT solver will find partial solutions (complete definitions for the edges in some subgraphs)...



Skipping Canonicity Checking

To save on canonicity tests it is safe to skip canonicity checking (and *not* block the object).

For example, hashes of canonical matrices can be recorded and any object with the same hash can be accepted.

This doesn't require much extra memory and even if hash collisions occur it never results in solutions being undetected.

KS Search Results

The speedup factor that we found when using SAT-based orderly generation in the search for KS systems of a given order:

order	speedup factor
16	6.5
17	13.6
18	37.8
19	104.5

The order 21 case was resolved in 25.7 CPU days (over 1000 times faster than the previous search).

KS Search Results

The speedup factor that we found when using SAT-based orderly generation in the search for KS systems of a given order:

order	speedup factor
16	6.5
17	13.6
18	37.8
19	104.5

The order 21 case was resolved in 25.7 CPU days (over 1000 times faster than the previous search).

The order 22 case was resolved in 5.3 CPU years. No KS system was found, so a *KS system must have at least 23 directions*.

Future Work: Better Parallelization

How best to split the search space when using parallelization? The obvious approach of splitting by finding all canonical subgraphs of a given order results in poor performance.

The cube-and-conquer method offers miraculous results for some combinatorial problems by splitting the search space into "cubes" solved by SAT solvers.

During splitting, we blocked noncanonical graphs of small order—but for large orders there are simply too many to block. Can the splitting employ isomorph-free generation more effectively?

A Promising Future!

I hope I've convinced you that SAT and isomorph-free generation methods deserve to be combined—despite little work pursuing this.

We have bolted together two methodologies that complement each other well. Can they be combined in other more effective ways?

There is still much to be done and great potential to be unlocked!