GraphSAT

-- a novel decision problem combining SAT with graph theory

Vaibhav Karve

University of Illinois at Urbana-Champaign
Department of Mathematics

vaibhavkarve.github.io/logic_seminar.pdf

Context

Joint work with Anil Hirani, UIUC Math. Work on GraphSAT started in 2017.

My talk will cover research from

- (**K.**, Anil Hirani) The complete set of minimal simple graphs that support unsatisfiable 2-CNFs. doi.org/10.1016/j.dam.2019.12.017
- (PhD thesis, unfinished) Graphical structure of unsatisfiable boolean formulas
- `graphsat` : a python package for scientific computations with CNFs, mult-hypergraphs, and GraphSAT

Acknowledgment:

- A part of this project was funded by the UIUC Campus Research Board's Arnold O. Beckman Award RB10150.
- Another part was funded by UIUC Math's David G. Bourgin Fellowship for Fall 2020.

SAT aka Boolean Satisfiability

Given a formula of the form

$$(a \vee \neg b \vee \neg c) \wedge (b \vee \neg c \vee d) \wedge (\neg a \vee \neg d)$$

SAT: is there a truth assignment for variables a, b, c, d that satisfies the formula?

If yes, then we say the formula is Satisfiable. If not, it is Unsatisfiable.

But SAT is hard

$$(a \vee \neg b \vee \neg c) \wedge (b \vee \neg c \vee d) \wedge (\neg a \vee \neg d)$$

Bruteforce approach: check all truth assignments

a	b	c	d	formula
⊤	⊤	⊤	⊤	⊥
⊤	⊤	⊤	⊥	⊤
⊤	⊤	⊥	⊤	⊥
⊤	⊤	⊥	⊥	⊤
...	...	...	...	...
...	...	...	...	...
⊥	⊥	⊥	⊥	⊤

Hence the formula is Satisfiable

n variables results in 2^n truth-assignments

SAT can be delegated to computers

$$(a \vee \neg b \vee \neg c) \wedge (b \vee \neg c \vee d) \wedge (\neg a \vee \neg d)$$

```
Python 3.9.0 (default) [GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> from graphsat import cnf, sat

>>> x = cnf.cnf([(1,-2,-3), (2,-3,4), (-1,-4)])
>>> sat.cnf_bruteforce_satcheck(x)
True
```

Complexity classes

P = all decision problems that can be solved[†] in time that is a polynomial function of the input length.

NP = all decision problems where, if the answer is YES[‡], then the proofs are verifiable[†] in polynomial time. $P \subseteq NP$.

co-NP = all decision problems where, if the answer is NO, then the proofs are verifiable[†] in polynomial time. $P \subseteq$ co-NP.

P might or might not be equal to NP. NP might or might not be equal to co-NP.

[†]by a deterministic Turing machine

‡for SAT, YES means satisfiable and NO means unsatisfiable

Complexity of SAT

1SAT ∈ P $(a) \wedge (\neg b) \wedge (\neg c) \wedge (d)$

2SAT ∈ P $(a \vee \neg b) \wedge (b \vee c) \wedge (\neg b \vee d) \wedge (\neg a \vee \neg d)$

... because of resolution and transitive closure.

3SAT ∈ NP \ co-NP $(a \vee \neg b \vee \neg c) \wedge (b \vee \neg c \vee d) \wedge (\neg a \vee c \vee \neg d)$

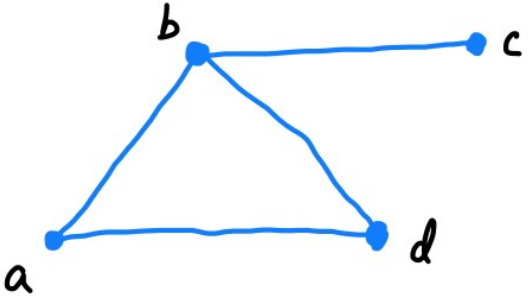
... we can always verify the answer in polynomial time.

... ⇝ 6SAT ⇝ 5SAT ⇝ 4SAT ⇝ 3SAT (reduction in polynomial time)

GraphSAT definition

Bundle up similar formulas and represent them as graphs.

$$(a \vee b) \wedge (b \vee c) \wedge (b \vee d) \wedge (a \vee d)$$
$$(a \vee b) \wedge (b \vee c) \wedge (b \vee d) \wedge (a \vee \neg d)$$
$$(a \vee b) \wedge (b \vee c) \wedge (b \vee d) \wedge (\neg a \vee d)$$
$$(a \vee b) \wedge (b \vee c) \wedge (b \vee d) \wedge (\neg a \vee \neg d)$$
$$(a \vee b) \wedge (b \vee c) \wedge (b \vee \neg d) \wedge (a \vee d)$$
$$\vdots$$
$$(\neg a \vee \neg b) \wedge (\neg b \vee \neg c) \wedge (\neg b \vee \neg d) \wedge (\neg a \vee \neg d)$$

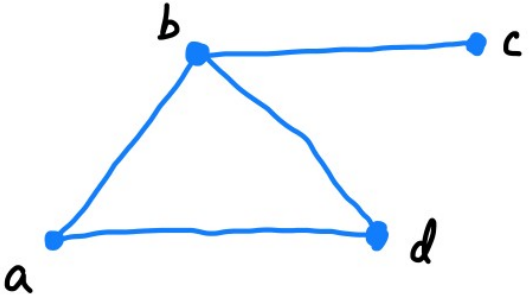


GraphSAT definition

Bundle up similar formulas and represent them as graphs.

A graph is satisfiable if every formula corresponding to the graph is satisfiable.

A graph is unsatisfiable if some formula corresponding to the graph is unsatisfiable.



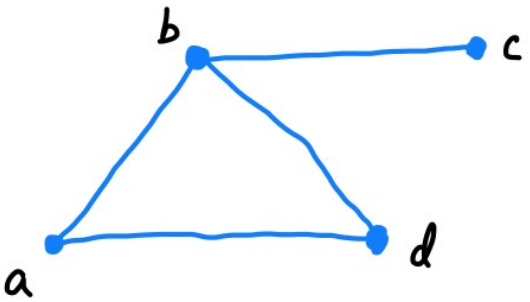
GraphSAT definition

```
Python 3.9.0 (default) [GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> from graphsat import graph, sat
>>> g = graph.graph([(1,2), (1,4), (2,3), (2,4)])

>>> len(sat.cnfs_from_graph(g)) == 4**len(g)
True

>>> sat.graph_bruteforce_satcheck(g)
True
```



Graphs allowed in GraphSAT

- ☑ Edges of size = 2 i.e simple edges

- ☑ Edges of size = 1 i.e. self-loops

  ... because $(a) \wedge (\neg b) \wedge (\neg c) \wedge (d)$

- ☑ Edges of size = 3 i.e. hyperedges

  ... because $(a \vee \neg b \vee \neg c) \wedge (b \vee \neg c \vee d) \wedge (\neg a \vee c \vee \neg d)$

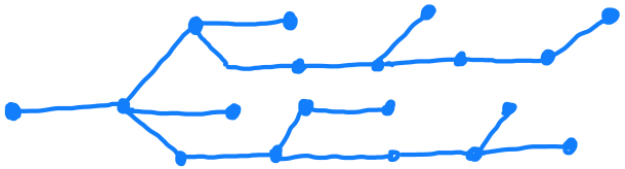
- ☑ Multi-edges

  ... because $(a \vee \neg b) \wedge (\neg a \vee b) \wedge (b \vee \neg c)$

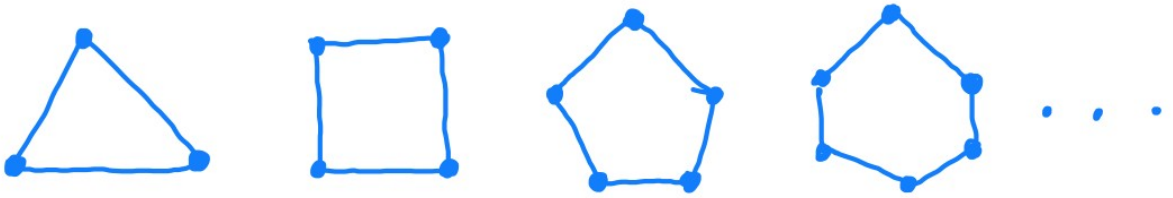
- ☑ Looped-multi-hyperedges

Families of known satisfiable graphs

Every tree is sat

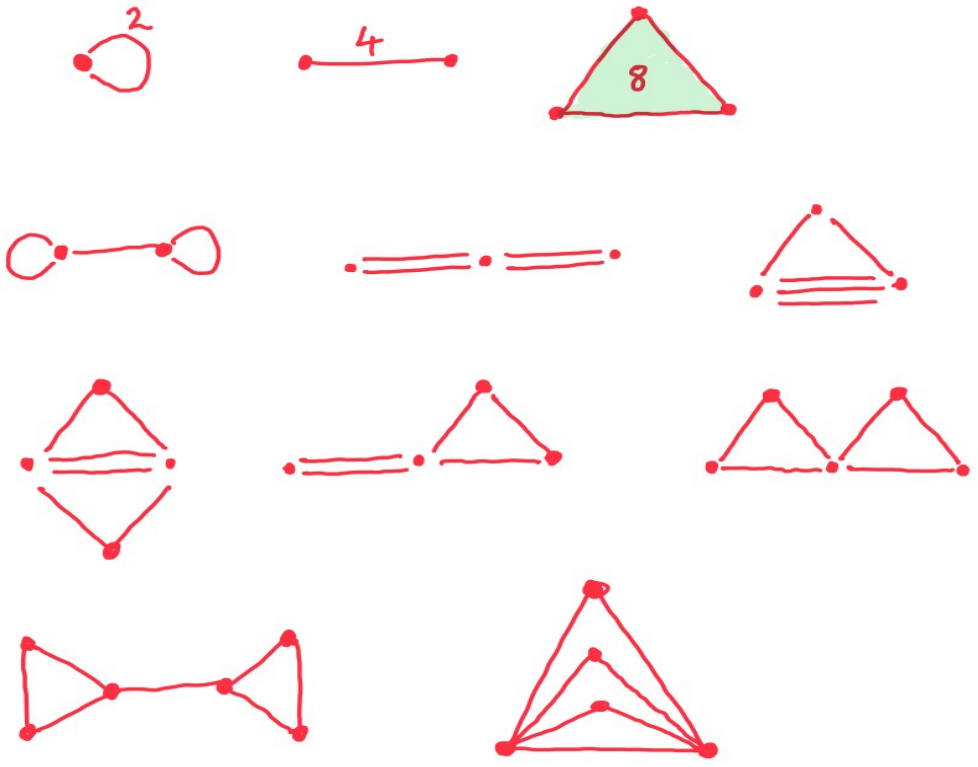


Every cycle is sat



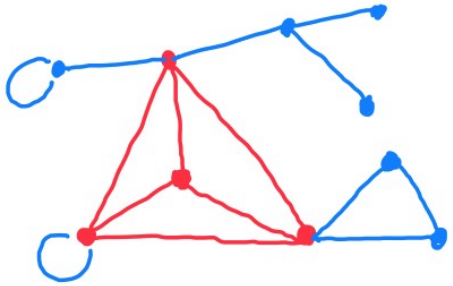
Closed under disjoint union

Zoo of known unsatisfiable graphs

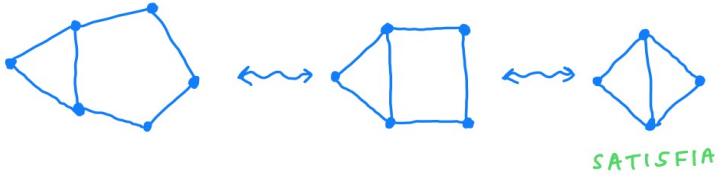


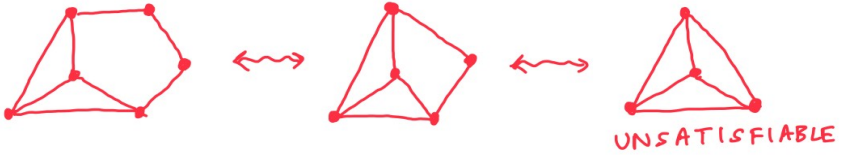
Structure theorems for 2GraphSAT

If a subgraph is unsatisfiable then so is the supergraph.



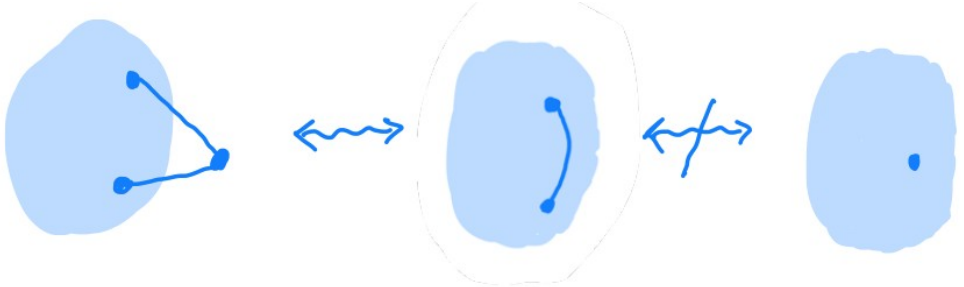
2GraphSAT is closed under path-smoothing.



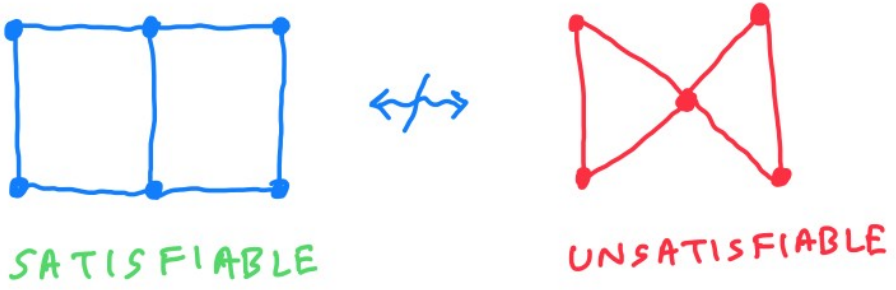


Structure theorems for 2GraphSAT

2GraphSAT is closed under Topological Minoring.



But it is not closed under Minoring.



Minoring is particularly desirable

1) Graph minor theorem [Neil Robertson, Paul Seymour]

   Every family of graphs closed under minors can be defined by a finite set of forbidden minors.

   (minoring defines a weak-quasi-ordering on the set of all graphs)

   For example, Wagner-Kuratowski theorem states that a graph is planar iff its minors do not include $\mathbf{K}_5$ or $\mathbf{K}_{3,3}$.

Minoring is particularly desirable

1) Graph minor theorem [Neil Robertson, Paul Seymour]

   Every family of graphs closed under minors can be defined by a finite set of forbidden minors.

2) Nameless theorem [HW]

   A finite list of forbidden minors yields a polynomial-time algorithm.

For example, we can write an **P**olynomial-time algorithm to check if a graph is planar.

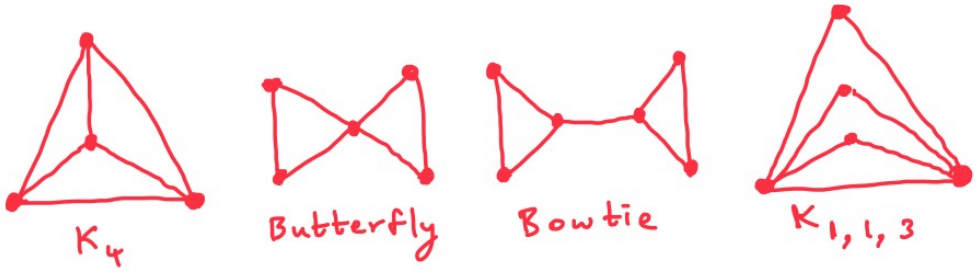
No such guarantees exist for 2GraphSAT because 2GraphSAT is not minor-closed.

Plot-twist

We still found a finite list of obstructions to satisfiability!

Complete list of 2GraphSAT obstructions [K., Hirani, 2019]

A simple graph is satisfiable iff it does not have the any of the following 4 topological minors:



Additionally, this gives us a **P**olynomial time algorithm for 2GraphSAT.

3GraphSAT

The bad news …

- Hypergraphs + Edge-multiplicities
- No immediately obvious generalization of minors, topological minors, edge-smoothing, edge-contraction.
- Pure hypergraphs $\rightsquigarrow$ (Looped+Simple+Hyper+Multi)-graphs

$2SAT \in P$	**$2GraphSAT \in P$**
$3SAT \in NP \setminus co\text{-}NP$	**$3GraphSAT \notin NP$** and **$3GraphSAT \notin co\text{-}NP$**
$4SAT \rightsquigarrow 3SAT \in P$	**$4GraphSAT \rightsquigarrow 3GraphSAT \notin P$**

3GraphSAT

The good news ...

```
Python 3.9.0 (default) [GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> from graphsat import mhgraph, sat
>>> g = mhgraph.mhgraph([(1,2,3), (2,3,4), (1,4), (1,4), (2,3)])

>>> len(sat.cnfs_from_mhgraph(g))
1024

>>> sat.mhgraph_bruteforce_satcheck(g)
True

>>> %time sat.mhgraph_bruteforce_satcheck(g)
CPU times: user 1.29 s, sys: 0 ns, total: 1.29 s
Wall time: 1.28 s

>>> %time sat.mhgraph_pysat_satcheck(g)
CPU times: user 75.8 ms, sys: 4 ms, total: 79.8 ms
Wall time: 92.1 ms
```

Global and local interaction

Local condition:

every vertex/variable can only take True/False values.

Global structure:

(un)satisfiability depends on all edges/connections.

GraphSAT is a global problem that can be solved locally using term rewriting.

Term rewriting

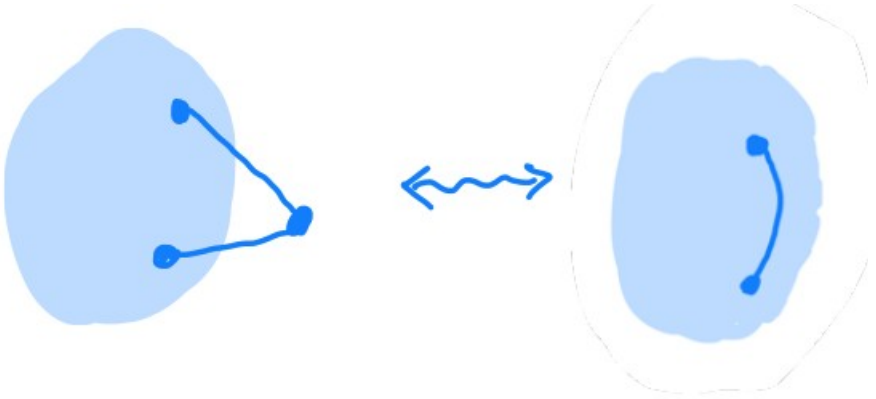
For example,

the MIU formal system from Gödel, Escher, Bach: an Eternal Golden Braid, by Douglas Hofstadter.

Nr.	Formal rule[note 1]	Informal explanation	Example
1.	x `I` → x `IU`	Add a `U` to the end of any string ending in `I`	`MI` to `MIU`
2.	`M` x → `M` xx	Double the string after the `M`	`MIU` to `MIUIU`
3.	x `III` y → x `U` y	Replace any `III` with a `U`	`MUIIIU` to `MUUU`
4.	x `UU` y → xy	Remove any `UU`	`MUUU` to `MU`

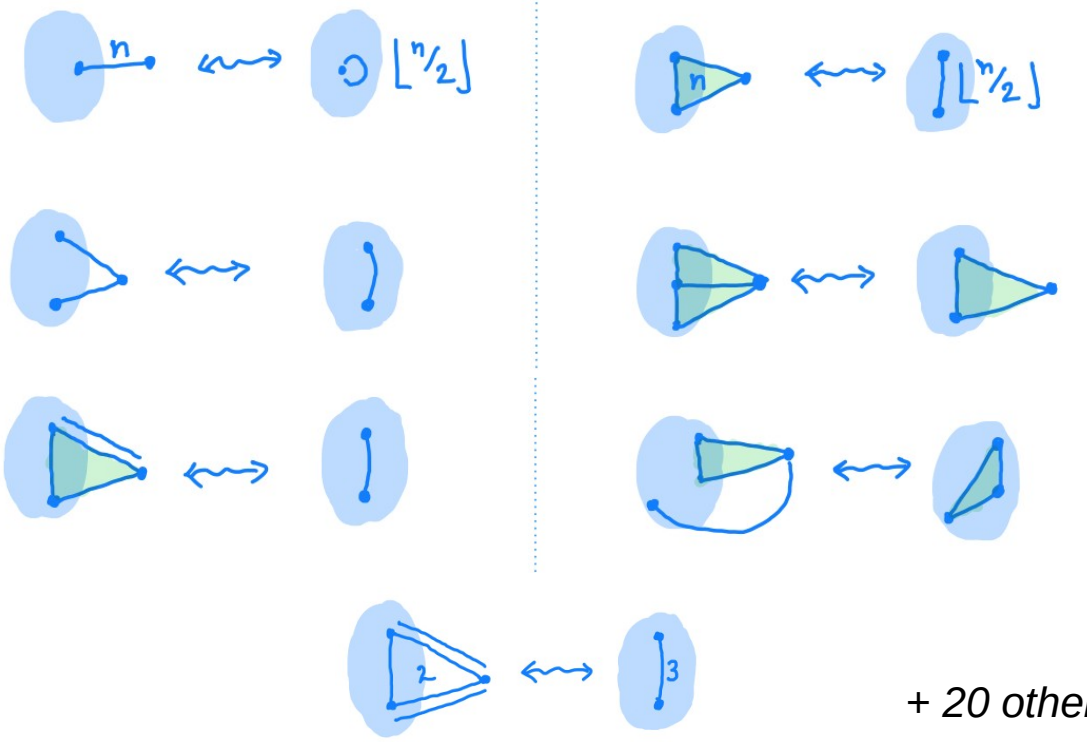
Local graph rewriting

Topological minoring is a local graph rewrite



Local graph rewriting

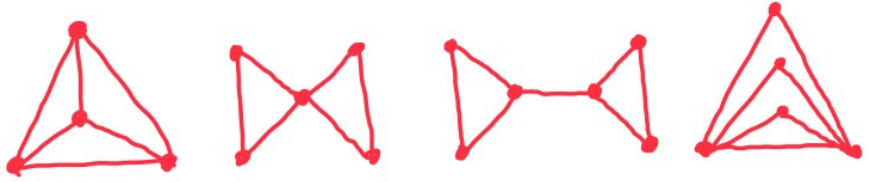
Lots of new graph rewriting operations



+ 20 other rewriting operations

Forbidden graphs

Unsatisfiable simple graphs

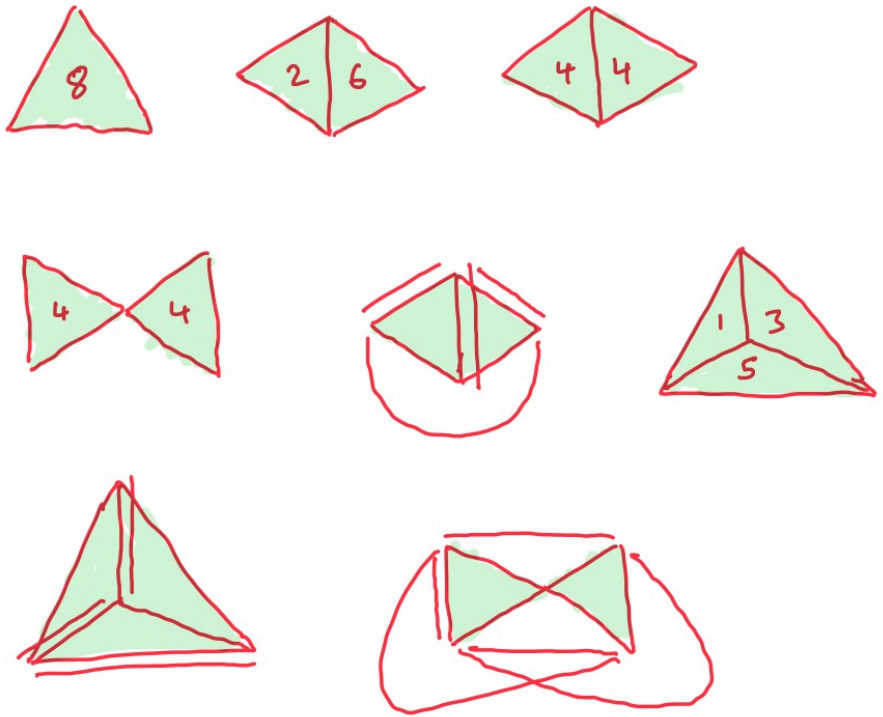


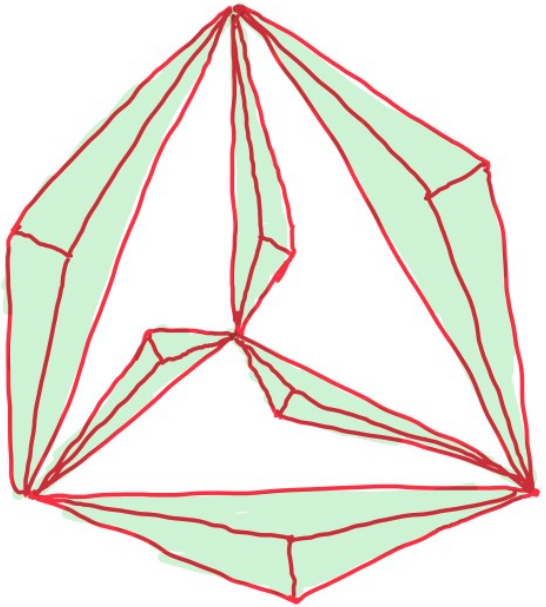
Unsatisfiable multi-graphs

Unsatisfiable multi-hypergraphs...

Forbidden graphs

Unsatisfiable multi-hypergraphs





+ ~1000 other multi-hypergraphs

Forbidden graphs

Unsatisfiable multi-hypergraphs

```
(1, 2)⁴
(1, 2)²,(2, 3)²
Tri(1, 2, 3) + (1,4),(2,4),(3,4)
Tri(1, 2, 3) + Tri(3, 4, 5)
Tri(1, 2, 3) + Tri(4, 5, 6) + (3,4)
Tri(1, 2, 3) + (1,4),(1,5),(2,4),(2,5)
Tet(1, 2, 3, 4) + 3-Tri(5, 1, 2, 3) + 3-Tri(6, 1, 2, 3) + 3-Tri(7, 1, 2, 3)
3-Tri(3, 1, 2, 5) + 3-Tri(4, 1, 2, 5) + 3-Tri(6, 1, 2, 5) + (3,4,6)
Tet(1, 3, 4, 6) + 3-Tri(1, 2, 3, 5) + (1,2,4),(1,2,6),(1,5,6),(2,4,5)
Tet(1, 2, 4, 6) + 3-Tri(1, 2, 3, 5) + (1,3,4),(1,5,6),(2,4,5),(3,4,5)
3-Tri(1, 2, 3, 4) + 3-Tri(1, 2, 5, 6) + (1,3,5),(1,4,6),(2,4,5),(3,4,5),(4,5,6)
3-Tri(1, 2, 3, 4) + 3-Tri(1, 2, 5, 6) + (1,3,5),(1,4,6),(2,3,6),(3,4,5),(4,5,6)
3-Tri(1, 2, 3, 4) + 3-Tri(1, 2, 5, 6) + 3-Tri(5, 2, 3, 4) + (1,4,6),(4,5,6)
Tet(1, 2, 3, 4) + 3-Tri(5, 1, 2, 3) + (1,2,6),(1,4,5),(3,4,5),(3,4,6)
Tet(1, 2, 3, 4) + 3-Tri(1, 3, 5, 6) + (1,2,6),(1,4,5),(2,3,6),(3,4,6)
Tet(1, 2, 3, 4) + 3-Tri(1, 3, 5, 6) + 3-Tri(6, 2, 3, 4) + (1,4,5)
Tet(1, 2, 3, 4) + 3-Tri(1, 3, 5, 6) + (1,4,5),(2,5,6),(3,4,6),(4,5,6)
Tet(1, 2, 3, 4) + Tet(1, 3, 5, 6) + (1,4,5),(3,4,6),(4,5,6)
Tet(2, 3, 4, 5) + (1,2,3),(1,2,4),(1,3,5),(1,4,5),(2,3,6),(2,5,6),(3,4,6)
3-Tri(1, 3, 5, 6) + 3-Tri(2, 1, 3, 4) + (1,4,5),(1,4,6),(2,3,5),(3,4,5),(3,4,6)
3-Tri(1, 3, 5, 6) + 3-Tri(2, 1, 3, 4) + 3-Tri(4, 3, 5, 6) + (1,4,5),(2,3,5)
Tet(1, 4, 5, 6) + 3-Tri(2, 1, 3, 4) + (1,3,5),(1,3,6),(2,3,6),(3,4,6)
Tet(1, 3, 5, 6) + 3-Tri(1, 2, 4, 5) + (1,2,3),(2,3,4),(3,4,5),(3,4,6)
Tet(1, 2, 3, 4) + 3-Tri(1, 3, 5, 6) + (1,2,6),(1,4,5),(3,4,5),(3,4,6)
Tet(1, 2, 3, 6) + 3-Tri(3, 4, 5, 6) + (1,2,4),(1,3,5),(1,4,5),(2,3,4)
Tet(3, 4, 5, 6) + 3-Tri(1, 2, 3, 6) + (1,2,4),(1,3,5),(1,4,5),(2,3,4)
Tet(1, 2, 3, 4) + 3-Tri(1, 2, 5, 6) + 3-Tri(5, 1, 3, 4) + (3,4,6)
Tet(1, 2, 3, 4) + 3-Tri(6, 1, 2, 5) + 3-Tri(5, 1, 3, 4) + (3,4,6)
Tet(1, 2, 3, 4) + 3-Tri(6, 1, 2, 5) + (1,3,5),(1,4,5),(2,3,5),(3,4,6)
Tet(1, 2, 3, 4) + 3-Tri(6, 1, 2, 5) + (1,3,5),(1,4,5),(2,3,6),(3,4,6)
Tet(1, 2, 3, 4) + 3-Tri(1, 2, 5, 6) + (1,3,5),(1,4,5),(2,3,6),(3,4,6)
Tet(1, 2, 3, 5) + (1,2,4),(1,2,6),(1,3,4),(1,4,5),(1,5,6),(2,3,6),(3,4,6)
Tet(1, 2, 4, 5) + 3-Tri(1, 2, 3, 6) + (1,3,4),(1,3,5),(1,5,6),(2,5,6)
Tet(1, 2, 3, 6) + 3-Tri(1, 2, 4, 5) + (1,3,4),(1,3,5),(1,5,6),(2,5,6)
Tet(1, 2, 3, 4) + Tet(1, 2, 5, 6) + (1,3,5),(1,4,5),(3,5,6)
Tet(3, 4, 5, 6) + 3-Tri(1, 2, 3, 5) + (1,2,4),(1,2,6),(1,4,5),(2,3,4)
Tet(1, 2, 3, 5) + 3-Tri(4, 1, 2, 5) + (1,2,6),(1,3,4),(2,5,6),(3,4,5)
Tet(1, 2, 3, 5) + 3-Tri(4, 1, 2, 5) + (1,2,6),(1,3,4),(2,5,6),(3,4,6)
Tet(1, 2, 3, 5) + 3-Tri(1, 2, 4, 6) + 3-Tri(4, 2, 3, 5) + (2,5,6)
Tet(1, 2, 3, 5) + Tet(1, 2, 4, 6) + (2,3,4),(2,4,5),(2,5,6)
Tet(1, 2, 3, 5) + 3-Tri(1, 2, 4, 6) + (1,3,4),(2,4,5),(2,5,6),(3,4,5)
Tet(1, 2, 3, 5) + Tet(1, 2, 4, 6) + 3-Tri(5, 2, 4, 6)
Tet(1, 2, 3, 5) + 3-Tri(1, 2, 4, 6) + 3-Tri(5, 2, 4, 6) + (3,4,6)
Tet(1, 2, 3, 4) + 3-Tri(5, 2, 3, 6) + (1,2,6),(1,3,5),(1,4,5),(2,4,5)
Tet(1, 3, 4, 5) + (1,2,3),(1,2,4),(1,2,6),(2,3,5),(2,4,5),(2,5,6),(3,4,6)
3-Tri(1, 2, 3, 4) + 3-Tri(5, 2, 4, 6) + (1,2,6),(1,3,5),(1,4,6),(2,3,5),(3,4,5)
Tet(1, 2, 3, 5) + 3-Tri(1, 2, 4, 6) + (1,4,5),(2,4,5),(2,5,6),(3,4,6)
3-Tri(1, 2, 3, 6) + 3-Tri(4, 1, 2, 5) + (1,3,5),(1,4,6),(2,3,5),(2,5,6),(3,4,6)
Tet(3, 4, 5, 6) + 3-Tri(1, 2, 3, 5) + (1,2,4),(1,2,6),(1,5,6),(2,4,5)
3-Tri(1, 2, 3, 4) + 3-Tri(2, 1, 5, 6) + (1,3,5),(2,3,6),(2,4,5),(3,4,5),(4,5,6)
Tet(1, 2, 4, 6) + 3-Tri(4, 3, 5, 6) + 3-Tri(3, 1, 2, 5) + (1,4,5)
Tet(1, 2, 3, 5) + 3-Tri(6, 1, 2, 3) + (1,2,4),(1,5,6),(2,4,5),(3,4,5)
Tet(1, 2, 4, 5) + 3-Tri(4, 2, 3, 6) + (1,2,3),(1,2,6),(1,3,5),(1,5,6)
```

Conclusion and Work in Progress

GraphSAT combines many different problems and can therefore be approached from different perspective.

- We need a hypergraph analogue of the graph minor theorem.

  Graph minor theorem [Neil Robertson, Paul Seymour]

  Every family of graphs closed under minors can be defined by a finite set of forbidden minors.

- We are looking to prove that the list of forbidden hypergraphs is finite.

- We are looking for ways to turn this into an algorithm for SAT.