Graphical Structure of Unsatisfiable Boolean Formulae

PhD thesis defence

Vaibhav Karve
Department of Mathematics
University of Illinois at Urbana-Champaign

2021-04-08

Outline

Introduction

Definitions

2GraphSAT

Local Rewriting in Graphs

Reduction rules

Computational results

Conclusion

Key results

2GraphSAT

- a new graph decision problem
- invariant under graph homeomorphism (topological minoring)
- a complete (and finite) list of minimal obstructions (simple graphs)

Local rewriting theorem

$$G[v] \;=\; \mathrm{sphere}(G,v),\mathrm{star}(G,v)[v] \;=\; \bigcup_{\substack{g_i\ h_i\ :\ \mathrm{Graph} \\ g_i, h_i\ =\ \mathrm{link}(G,v)}} \mathrm{sphere}(G,v),(g_i \vee h_i)$$

3GraphSAT

- graph rewrite/reduction rules that preserve satisfiability
- systematic computer-aided search on looped-multi-hypergraphs
- an incomplete list of minimal obstructions
- a Python package called `graphsat`

CNFs

Definition (CNF)

A boolean formula is in *Conjunctive Normal Form* if it is a conjunction of disjunctions of variables and their negations.

CNFs

Definition (CNF)

A boolean formula is in *Conjunctive Normal Form* if it is a conjunction of disjunctions of variables and their negations.

For example,

$$(x_1 \text{ OR } x_2 \text{ OR } \text{ NOT } x_3) \text{ AND } (x_3 \text{ OR } x_4) \text{ AND } (\text{ NOT } x_4)$$

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$$

$$\Big\{\{x_1, x_2, \neg x_3\},\ \{x_3, x_4\},\ \{\neg x_4\}\Big\}$$

CNFs

Definition (CNF)

A boolean formula is in *Conjunctive Normal Form* if it is a conjunction of disjunctions of variables and their negations.

For example,

$$(x_1 \text{ OR } x_2 \text{ OR } \text{ NOT } x_3) \text{ AND } (x_3 \text{ OR } x_4) \text{ AND } (\text{ NOT } x_4)$$

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$$

$$\Big\{\{x_1, x_2, \neg x_3\},\ \{x_3, x_4\},\ \{\neg x_4\}\Big\}$$

- Variable $\equiv$ an element of a countably infinite set, i.e. $x_1, x_2, \ldots$
- Literal $\equiv$ a variable or its negation, i.e. $x_1, \neg x_1, x_2, \neg x_2, \ldots$
- Clause $\equiv$ a nonempty set of literals
- CNF $\equiv$ a nonempty set of clauses

SAT

Definition (Boolean Satisfiability)

A CNF is *satisfiable* if there exists a satisfying truth-assignment. Otherwise, it is *unsatisfiable*.

SAT

Definition (Boolean Satisfiability)

A CNF is *satisfiable* if there exists a satisfying truth-assignment. Otherwise, it is *unsatisfiable*.

For example,

- $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$ satisfies $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$
- $[x_1 := \top,\ x_3 := \top,\ x_4 := \top]$ does not satisfy $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$

SAT

Definition (Boolean Satisfiability)

A CNF is *satisfiable* if there exists a satisfying truth-assignment. Otherwise, it is *unsatisfiable*.

For example,

- $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$ satisfies $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$
- $[x_1 := \top,\ x_3 := \top,\ x_4 := \top]$ does not satisfy $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$

If we insist on constructing an unsatisfiable CNF —

- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$

SAT

Definition (Boolean Satisfiability)

A CNF is *satisfiable* if there exists a satisfying truth-assignment. Otherwise, it is *unsatisfiable*.

For example,

- $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$ satisfies $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$
- $[x_1 := \top,\ x_3 := \top,\ x_4 := \top]$ does not satisfy $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$

If we insist on constructing an unsatisfiable CNF —

- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$ is satisfiable because $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$

SAT

Definition (Boolean Satisfiability)

A CNF is *satisfiable* if there exists a satisfying truth-assignment. Otherwise, it is *unsatisfiable*.

For example,

- $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$ satisfies $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$
- $[x_1 := \top,\ x_3 := \top,\ x_4 := \top]$ does not satisfy $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$

If we insist on constructing an unsatisfiable CNF —

- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$ is satisfiable because $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$
- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$

SAT

Definition (Boolean Satisfiability)

A CNF is *satisfiable* if there exists a satisfying truth-assignment. Otherwise, it is *unsatisfiable*.

For example,

- $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$ satisfies $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$
- $[x_1 := \top,\ x_3 := \top,\ x_4 := \top]$ does not satisfy $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$

If we insist on constructing an unsatisfiable CNF —

- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$ is satisfiable because $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$
- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$ is satisfiable because $[x_1 := \bot,\ x_2 := \bot,\ x_3 := \top,\ x_4 := \bot]$

SAT

Definition (Boolean Satisfiability)

A CNF is *satisfiable* if there exists a satisfying truth-assignment. Otherwise, it is *unsatisfiable*.

For example,

- $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$ satisfies $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$
- $[x_1 := \top,\ x_3 := \top,\ x_4 := \top]$ does not satisfy $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$

If we insist on constructing an unsatisfiable CNF —

- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$ is satisfiable because $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$
- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$ is satisfiable because $[x_1 := \bot,\ x_2 := \bot,\ x_3 := \top,\ x_4 := \bot]$
- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$

SAT

Definition (Boolean Satisfiability)

A CNF is *satisfiable* if there exists a satisfying truth-assignment. Otherwise, it is *unsatisfiable*.

For example,

- $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$ satisfies $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$
- $[x_1 := \top,\ x_3 := \top,\ x_4 := \top]$ does not satisfy $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$

If we insist on constructing an unsatisfiable CNF —

- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$ is satisfiable because $[x_1 := \top,\ x_3 := \top,\ x_4 := \bot]$
- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$ is satisfiable because $[x_1 := \bot,\ x_2 := \bot,\ x_3 := \top,\ x_4 := \bot]$
- $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$ is unsatisfiable.

SAT

History

- Cook & Levin
- NP-completeness

Definition (SAT decision problem)

- Instance: A boolean formula in conjunctive normal form.
- Question: Is the given formula satisfiable?
- Certificate: If yes, then the certificate is a truth assignment. If no, then there is no certificate.

It is easier to verify that a CNF is satisfiable, than to prove that it is unsatisfiable.

SAT using a computer

To verify that $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$ is satisfiable —

```
from cnf import cnf, Cnf
from sat import cnf_pysat_satcheck as sat

x : Cnf = cnf([[1, 2, -3], [3, 4], [-4]])
print(sat(x))
```

```
True
```

SAT using a computer

To verify that $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4)$ is satisfiable —

```
from cnf import cnf, Cnf
from sat import cnf_pysat_satcheck as sat

x : Cnf = cnf([[1, 2, -3], [3, 4], [-4]])
print(sat(x))
```

```
True
```

To prove that $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$ is unsatisfiable —

```
x2 : Cnf = cnf([[1, 2, -3], [3, 4], [-4], [-1, -3, 4], [-2, -3, 4]])
print(sat(x2))
```

```
False
```

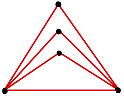
Graphs

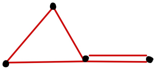
- Vertex $\equiv$ an element of a countably infinite set, i.e. points/nodes.
- Edge $\equiv$ a nonempty set of vertices, i.e. loops, simple connections, hyperedges.
- Graph $\equiv$ a nonempty set of edges, i.e. a looped-multi-hypergraph.

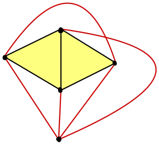
Graphs

- Vertex $\equiv$ an element of a countably infinite set, i.e. points/nodes.
- Edge $\equiv$ a nonempty set of vertices, i.e. loops, simple connections, hyperedges.
- Graph $\equiv$ a nonempty set of edges, i.e. a looped-multi-hypergraph.

Examples







Graphs | CNFs

Each graph "supports" a set of CNFs. Each CNF supported on a graph has the same underlying structure.

$(v_1 v_2 v_3), (v_3 v_4), (v_4)$

$$(x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_4)$$
$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_4)$$
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_3 \vee x_4) \wedge (x_4)$$
$$\vdots$$
$$(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_4)$$

GraphSAT | SAT

Each graph supports a set of sentences. We can define a notion of satisfiability for the graph using the CNFs it supports.

Definition (Satisfiability of graphs)

If every CNF supported on a graph is satisfiable, then the graph itself is *satisfiable*. If any CNF supported on a graph is unsatisfiable, then the graph is *unsatisfiable*.

GraphSAT | SAT

Each graph supports a set of sentences. We can define a notion of satisfiability for the graph using the CNFs it supports.

Definition (Satisfiability of graphs)

If every CNF supported on a graph is satisfiable, then the graph itself is *satisfiable*. If any CNF supported on a graph is unsatisfiable, then the graph is *unsatisfiable*.

SAT decision problem

- Instance: A CNF.
- Question: Is the given formula satisfiable?
- Certificate: If yes, then the certificate is a truth assignment.
  If not, then there is no certificate.

It is easier to verify that a CNF is satisfiable, than to prove that it is unsatisfiable.

GraphSAT decision problem

- Instance: A looped-multi-hypergraph.
- Question: Is the given graph satisfiable?
- Certificate: If not, then the certificate is an unsatisfiable CNF supported on the graph.
  If yes, then there is no certificate.

It is easier to verify that a graph is unsatisfiable, than to prove that it is satisfiable.

GraphSAT | SAT

These problems have several variants –

1. 2SAT $\equiv$ SAT restricted to CNFs that have clauses of size at most 2. 2SAT is in complexity class P.

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3)$$

2. 3SAT $\equiv$ SAT restricted to CNFs that have clauses of size at most 3. 3SAT is in complexity class NP-complete.

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \neg x_4)$$

3. 2GraphSAT $\equiv$ GraphSAT restricted to graphs with edges of size at most 2. We proved that 2GraphSAT is in complexity class P.

$$v_1 v_2,\ v_2 v_3$$

4. 3GraphSAT $\equiv$ GraphSAT restricted to graphs with edges of size at most 3. We do not yet know its complexity class.

$$v_1 v_2 v_3,\ v_2 v_3 v_4$$

2GraphSAT is in complexity class P

This is work that was presented in the Prelim exam.

Theorem (Polynomial time algorithm for 2GraphSAT)

Let G be a looped-multi-graph with n vertices. There exists an algorithm that can decide whether G is satisfiable in $\mathcal{O}(n)$ steps.

Proof sketch.

1. Show that G is unsatisfiable if and only if it contains at least one of four special graphs as a topological minor. We label this set of four special graphs M_4.
   This needs a proof.

2GraphSAT is in complexity class P

This is work that was presented in the Prelim exam.

Theorem (Polynomial time algorithm for 2GraphSAT)

Let G be a looped-multi-graph with n vertices. There exists an algorithm that can decide whether G is satisfiable in $\mathcal{O}(n)$ steps.

Proof sketch.

1. Show that G is unsatisfiable if and only if it contains at least one of four special graphs as a topological minor. We label this set of four special graphs M_4.
   This needs a proof.
2. Then, our algorithm just searches G for the four special topological minors from M_4.
   This is a consequence of 1.

2GraphSAT is in complexity class P

This is work that was presented in the Prelim exam.

Theorem (Polynomial time algorithm for 2GraphSAT)

Let G be a looped-multi-graph with n vertices. There exists an algorithm that can decide whether G is satisfiable in $\mathcal{O}(n)$ steps.

Proof sketch.

1. Show that G is unsatisfiable if and only if it contains at least one of four special graphs as a topological minor. We label this set of four special graphs M_4.
   This needs a proof.
2. Then, our algorithm just searches G for the four special topological minors from M_4.
   This is a consequence of 1.
3. There exists an $\mathcal{O}(n)$-algorithm for graph decision problem $\textsc{TopologicalMinor}(g, G)$.
   »» N. Robertson and P.D. Seymour. "Graph Minors XIII. The Disjoint Paths Problem". In: *Journal of Combinatorial Theory, Series B* 63.1 (1995), pp. 65–110. ISSN: 0095-8956
4. Since M_4 is finite, the algorithm still runs in $\mathcal{O}(n)$ steps.

□

2GraphSAT is homeomorphism-invariant

Definition (Topological minors and Homeomorphism)

Graph G_1 is a *topological minor* of graph G_2 if a subgraph of G_2 can be obtained from a series of edge-subdivisions of G_1.

If G_2 itself can be obtained from a series of edge-subdivisions of G_1, then G_2 and G_1 are *homeomorphic*.

2GraphSAT is homeomorphism-invariant

Definition (Topological minors and Homeomorphism)

Graph G_1 is a *topological minor* of graph G_2 if a subgraph of G_2 can be obtained from a series of edge-subdivisions of G_1.

If G_2 itself can be obtained from a series of edge-subdivisions of G_1, then G_2 and G_1 are *homeomorphic*.

Example (Topological minors and Homeomorphism)

2GraphSAT is homeomorphism-invariant

Definition (Topological minors and Homeomorphism)

Graph G_1 is a *topological minor* of graph G_2 if a subgraph of G_2 can be obtained from a series of edge-subdivisions of G_1.

If G_2 itself can be obtained from a series of edge-subdivisions of G_1, then G_2 and G_1 are *homeomorphic*.

Example (Topological minors and Homeomorphism)

Lemma (Invariance under homeomorphism)

If G_1 and G_2 are homeomorphic graphs, then G_1 is unsatisfiable if and only if G_2 is unsatisfiable.

Proof sketch.

2GraphSAT is homeomorphism-invariant

Definition (Topological minors and Homeomorphism)

Graph G_1 is a *topological minor* of graph G_2 if a subgraph of G_2 can be obtained from a series of edge-subdivisions of G_1.

If G_2 itself can be obtained from a series of edge-subdivisions of G_1, then G_2 and G_1 are *homeomorphic*.

Example (Topological minors and Homeomorphism)

Lemma (Invariance under homeomorphism)

If G_1 and G_2 are homeomorphic graphs, then G_1 is unsatisfiable if and only if G_2 is unsatisfiable.

Proof sketch.

1. Assume WLOG that G_2 is a single edge-subdivision of G_1.

2GraphSAT is homeomorphism-invariant

Definition (Topological minors and Homeomorphism)

Graph G_1 is a *topological minor* of graph G_2 if a subgraph of G_2 can be obtained from a series of edge-subdivisions of G_1.

If G_2 itself can be obtained from a series of edge-subdivisions of G_1, then G_2 and G_1 are *homeomorphic*.

Example (Topological minors and Homeomorphism)

Lemma (Invariance under homeomorphism)

If G_1 and G_2 are homeomorphic graphs, then G_1 is unsatisfiable if and only if G_2 is unsatisfiable.

Proof sketch.

1. Assume WLOG that G_2 is a single edge-subdivision of G_1.
2. For every unsatisfiable CNF supported on G_1, construct an unsatisfiable CNF on G_2.

2GraphSAT is homeomorphism-invariant

Definition (Topological minors and Homeomorphism)

Graph G_1 is a *topological minor* of graph G_2 if a subgraph of G_2 can be obtained from a series of edge-subdivisions of G_1.

If G_2 itself can be obtained from a series of edge-subdivisions of G_1, then G_2 and G_1 are *homeomorphic*.

Example (Topological minors and Homeomorphism)

Lemma (Invariance under homeomorphism)

If G_1 and G_2 are homeomorphic graphs, then G_1 is unsatisfiable if and only if G_2 is unsatisfiable.

Proof sketch.

1. Assume WLOG that G_2 is a single edge-subdivision of G_1.
2. For every unsatisfiable CNF supported on G_1, construct an unsatisfiable CNF on G_2.
3. For every unsatisfiable CNF supported on G_2, construct an unsatisfiable CNF on G_1.

$\square$

2GraphSAT is homeomorphism-invariant

This means we only need to consider graphs up to homeomorphisms!

Example (Satisfiability of graph families)

1. All tree graphs are satisfiable.

2GraphSAT is homeomorphism-invariant

This means we only need to consider graphs up to homeomorphisms!

Example (Satisfiability of graph families)

1. All tree graphs are satisfiable.
2. Triangles are satisfiable $\implies$ all cycle graphs are satisfiable.

2GraphSAT is homeomorphism-invariant

This means we only need to consider graphs up to homeomorphisms!

Example (Satisfiability of graph families)

1. All tree graphs are satisfiable.
2. Triangles are satisfiable $\implies$ all cycle graphs are satisfiable.
3. A graphs is satisfiable if and only if each of its connected components is satisfiable.

2GraphSAT is homeomorphism-invariant

This means we only need to consider graphs up to homeomorphisms!

Example (Satisfiability of graph families)

1. All tree graphs are satisfiable.
2. Triangles are satisfiable $\implies$ all cycle graphs are satisfiable.
3. A graphs is satisfiable if and only if each of its connected components is satisfiable.
4. A graph containing an unsatisfiable subgraph is unsatisfiable.

2GraphSAT is homeomorphism-invariant

This means we only need to consider graphs up to homeomorphisms!

Example (Satisfiability of graph families)

1. All tree graphs are satisfiable.
2. Triangles are satisfiable $\implies$ all cycle graphs are satisfiable.
3. A graphs is satisfiable if and only if each of its connected components is satisfiable.
4. A graph containing an unsatisfiable subgraph is unsatisfiable.

Example (Some unsatisfiable graphs)

What is M_4?

Theorem (The complete set of minimal unsatisfiable simple graphs)

Theorem (The complete set of minimal unsatisfiable looped-multi-graphs)

Proof sketch.

Combinatorial enumeration of all graphs based on number of independent cycles.

- 0 cycles: tree graph $\implies$ always satisfiable. 1 cycle: cycle graph $\implies$ always satisfiable.
- 2 cycles: we show that either this is satisfiable, or has a butterfly topological minor, or a bow-tie topological minor.
- 3 cycles: we show that it always has a topological minor that is in M_4
- 4 cycles: any such graph has a 3 cycle subgraph. Revert to previous case.

Analogy for understanding local graph rewriting

	Math analogy	Physics analogy
CNF	function	state of particle
Graph	family of functions	superposition of many states

Analogy for understanding local graph rewriting

	Math analogy	Physics analogy
CNF	function	state of particle
Graph	family of functions	superposition of many states
CNF evaluated at a variable	function evaluated at a point	observing a collapsed state
Graph evaluated at a vertex	??	sum over all states/paths/histories

Analogy for understanding local graph rewriting

	Math analogy	Physics analogy
CNF	function	state of particle
Graph	family of functions	superposition of many states
CNF evaluated at a variable	function evaluated at a point	observing a collapsed state
Graph evaluated at a vertex	??	sum over all states/paths/histories

Example (Cnf evaluated at a variable)

x : CNF $= (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_4)$
x_1 : Variable

$$\begin{aligned} x[x_1] &= (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_4)[x_1] \\ &= (x_1 \vee \neg x_2)[x_1] \wedge (x_2 \vee x_3)[x_1] \wedge (\neg x_4)[x_1] \\ &= (\top \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_4) \\ &= (x_2 \vee x_3) \wedge (\neg x_4) \end{aligned}$$

Evaluating at a variable and its negation keeps CNF-satisfiability unchanged!

Local graph rewriting

Theorem (Local graph rewriting)

Let G be a graph and let v be a vertex of G. Evaluating G at v yields the following set of CNFs:

$$G[v] = sphere(G, v), star(G, v)[v] = \bigcup_{\substack{g_i \ h_i \ : \ Graph \\ g_i, h_i \ = \ link(G,v)}} sphere(G, v), (g_i \vee h_i)$$

Local rewriting examples

```python
from mhgraph import mhgraph, MHGraph, vertex, Vertex
from graph_rewrite import local_rewrite

G : MHGraph = mhgraph([[1, 2], [1, 3], [1, 4], [1, 5]])
v : Vertex = vertex(1)

local_rewrite(G, v, True)
```

	(S),(3, 4),(3, 5),(2, 5),(2, 4)	(S),(2, 5),(2, 3),(2, 4)	(S),(3, 4),(3, 5),(2, 3)	(S),(4, 5),(3, 4),(2, 5),(2, 3)	(S),(2, 5),(3, 5),(4, 5)
0	(-S)(3,4)(3,-5)(-2,4)(-2,-5)	(-S)(2,3)(2,-5)(2,-4)	(-S)(3,4)(3,5)(-2,3)	(S)(-2,3)(3,-4)(-4,-5)(-2,-5)	(S)(4,5)(3,5)(2,5)
1	(-S)(-3,5)(-2,5)(-2,-4)(-3,-4)	(S)(2,5)(2,3)(2,-4)	(S)(3,5)(3,-4)(-2,3)	(-S)(3,4)(-2,3)(4,-5)(-2,-5)	(S)(2,-5)(3,-5)(-4,-5)
2	(S)(2,4)(2,5)(-3,4)(-3,5)	(S)(2,-5)(2,-3)(2,-4)	(-S)(3,5)(3,-4)(-2,3)	(S)(3,4)(2,3)(4,5)(2,5)	(-S)(4,5)(2,5)(-3,5)
3	(-S)(-3,4)(-3,5)(-2,5)(-2,4)	(S)(-2,5)(-2,4)(-2,-3)	(S)(3,4)(3,5)(-2,3)	(-S)(-2,3)(3,-4)(-4,-5)(-2,-5)	(-S)(4,5)(-3,5)(-2,5)
4	(-S)(3,5)(3,-4)(-2,5)(-2,-4)	(S)(-2,4)(-2,-5)(-2,-3)	(-S)(3,-5)(3,-4)(-2,3)	(S)(-4,5)(-2,5)(-3,-4)(-2,-3)	(-S)(3,5)(-4,5)(-2,5)
5	(S)(3,4)(2,4)(2,-5)(3,-5)	(-S)(2,-5)(2,-3)(2,-4)	(-S)(3,4)(2,3)(3,5)	(-S)(-3,4)(2,-3)(4,-5)(2,-5)	(-S)(2,-5)(-3,-5)(-4,-5)
6	(-S)(3,5)(2,5)(3,-4)(2,-4)	(-S)(2,5)(2,3)(2,-4)	(S)(2,3)(3,5)(3,-4)	(S)(2,5)(-4,5)(2,-3)(-3,-4)	(-S)(4,-5)(-3,-5)(-2,-5)
7	(-S)(3,-5)(3,-4)(-2,-4)(-2,-5)	(S)(2,3)(2,-5)(2,-4)	(-S)(-3,4)(-3,5)(-2,-3)	(S)(-4,5)(-2,3)(3,-4)(-2,5)	(-S)(2,5)(3,5)(-4,5)
8	(S)(3,4)(2,4)(3,5)(2,5)	(-S)(2,4)(2,-5)(2,-3)	(S)(3,4)(2,3)(3,5)	(-S)(3,4)(2,3)(2,-5)(4,-5)	(S)(-4,-5)(-3,-5)(-2,-5)
9	(-S)(3,4)(3,5)(-2,5)(-2,4)	(S)(-2,3)(-2,-5)(-2,-4)	(-S)(2,3)(3,5)(3,-4)	(-S)(-4,-5)(-3,-4)(-2,-5)(-2,-3)	(S)(2,-5)(-3,-5)(-4,-5)
10	(S)(-3,-5)(-2,-4)(-3,-4)(-2,-5)	(-S)(2,5)(2,-3)(2,-4)	(S)(3,4)(2,3)(3,-5)	(-S)(2,3)(2,-5)(3,-4)(-4,-5)	(S)(3,5)(-4,5)(-2,5)
11	(S)(-3,4)(-2,4)(-3,-5)(-2,-5)	(-S)(-2,3)(-2,-5)(-2,-4)	(-S)(2,3)(3,-5)(3,-4)	(S)(3,4)(2,3)(2,-5)(4,-5)	(S)(4,-5)(-3,-5)(-2,-5)
12	(-S)(2,5)(2,-4)(-3,5)(-3,-4)	(S)(-2,3)(-2,5)(-2,-4)	(-S)(-3,4)(2,-3)(-3,5)	(S)(2,3)(2,5)(-4,5)(3,-4)	(S)(3,-5)(-4,-5)(-2,-5)
13	(-S)(2,4)(-3,4)(2,-5)(-3,-5)	(S)(2,4)(2,5)(2,3)	(-S)(2,-3)(-3,-5)(-3,-4)	(S)(4,5)(2,5)(-3,4)(2,-3)	(-S)(2,-5)(4,-5)(-3,-5)
14	(-S)(3,4)(2,4)(2,-5)(3,-5)	(-S)(-2,5)(-2,-3)(-2,-4)	(S)(-3,4)(-3,-5)(-2,-3)	(S)(-3,4)(2,-3)(4,-5)(2,-5)	(S)(2,5)(-4,5)(-3,5)
15	(S)(-3,4)(-3,5)(-2,5)(-2,4)	(-S)(-2,-5)(-2,-3)(-2,-4)	(S)(-3,5)(-3,-4)(-2,-3)	(S)(4,5)(-3,4)(-2,5)(-2,-3)	(S)(4,5)(2,5)(-3,5)
16	(S)(3,5)(3,-4)(-2,5)(-2,-4)	(-S)(2,4)(2,5)(2,-3)	(-S)(-3,-5)(-3,-4)(-2,-3)	(-S)(-3,4)(4,-5)(-2,-5)(-2,-3)	(S)(4,5)(3,5)(-2,5)
17	(S)(-3,5)(-2,5)(-2,-4)(-3,-4)	(S)(2,4)(2,3)(2,-5)	(S)(-3,4)(-3,5)(-2,-3)	(S)(2,-5)(2,-3)(-4,-5)(-3,-4)	(-S)(2,-5)(3,-5)(4,-5)
18	(S)(3,4)(3,-5)(-2,4)(-2,-5)	(-S)(-2,4)(-2,-5)(-2,-3)	(S)(-3,-5)(-3,-4)(-2,-3)	(-S)(3,4)(2,3)(4,5)(2,5)	(-S)(2,-5)(3,-5)(-4,-5)
19	(-S)(2,4)(2,5)(-3,4)(-3,5)	(-S)(-2,5)(-2,4)(-2,-3)	(-S)(2,-3)(-3,5)(-3,-4)	(S)(3,4)(-2,3)(4,-5)(-2,-5)	(-S)(4,5)(2,5)(3,5)
20	(S)(2,4)(-3,4)(2,-5)(-3,-5)	(S)(-2,3)(-2,4)(-2,-5)	(-S)(-3,4)(2,-3)(-3,-5)	(-S)(-4,5)(-2,3)(3,-4)(-2,5)	(S)(2,-5)(3,-5)(4,-5)
21	(S)(2,5)(2,-4)(-3,5)(-3,-4)	(S)(-2,3)(-2,5)(-2,4)	(-S)(-3,5)(-3,-4)(-2,-3)	(-S)(2,-5)(2,-3)(-4,-5)(-3,-4)	(-S)(4,5)(3,5)(-2,5)
22	(S)(2,-4)(2,-5)(3,-5)(3,-4)	(-S)(2,4)(2,3)(2,-5)	(-S)(-3,4)(-3,-5)(-2,-3)	(S)(-3,4)(4,-5)(-2,-5)(-2,-3)	(S)(3,-5)(4,-5)(-2,-5)

Aside: graph disjunction

Let G_1 and G_2 be graphs. The disjunction of G_1 and G_2 is defined to be the following set of CNFs –

$$G_1 \vee G_2 = \{x_1 \vee x_2 \mid (x_1 : \mathrm{CNF}) \in G_1,\ (x_2 : \mathrm{CNF}) \in G_2\}$$

Table 3: Graph disjunctions where size of $\boldsymbol{h_1}$ + size of $\boldsymbol{h_2}$ is 2.

h_1		h_2		$h_1 \vee h_2$
a	$\vee$	a	$=$	$\top \cup a$
a	$\vee$	b	$=$	ab
a	$\vee$	bc	$=$	abc
a	$\vee$	ab	$=$	$\top \cup ab$
ab	$\vee$	cd	$=$	$abcd$
ab	$\vee$	ac	$=$	$\top \cup abc$
ab	$\vee$	ab	$=$	$\top \cup ab$

Table 4: Graph disjunctions where size of $\boldsymbol{h_1}$ + size of $\boldsymbol{h_2}$ is 3.

h_1		h_2		$h_1 \vee h_2$
a	$\vee$	a^2	$=$	a
a	$\vee$	b^2	$=$	a
a	$\vee$	$(ab)^2$	$=$	$\top \cup a \cup ab$
a	$\vee$	(b, ab)	$=$	$a \cup ab$
ab	$\vee$	c^2	$=$	ab
ab	$\vee$	(a, c)	$=$	$ab \cup abc$
ab	$\vee$	a^2	$=$	ab
ab	$\vee$	(a, b)	$=$	$\top \cup ab$
ab	$\vee$	(c, ac)	$=$	$ab \cup abc$
ab	$\vee$	(c, ab)	$=$	$ab \cup abc$
ab	$\vee$	(a, cd)	$=$	$ab \cup abcd$
ab	$\vee$	(a, ac)	$=$	$\top \cup ab \cup abc$
ab	$\vee$	(a, bc)	$=$	$\top \cup ab \cup abc$
ab	$\vee$	(a, ab)	$=$	$\top \cup ab$
ab	$\vee$	(ab, cd)	$=$	$ab \cup abcd$
ab	$\vee$	(ab, ac)	$=$	$\top \cup ab \cup abc$
ab	$\vee$	$(ac)^2$	$=$	$\top \cup ab \cup abc$
ab	$\vee$	(ac, bc)	$=$	$\top \cup ab \cup abc$
ab	$\vee$	$(ab)^2$	$=$	$\top \cup ab$

Table 5: Subset-superset pairs for graph disjunctions where size of $\boldsymbol{h_1}$ + size of $\boldsymbol{h_2}$ is at most 3.

Subset		h_1		h_2		Superset
		a	$\vee$	(b, c)	$\subset$	(ab, ac)
ab	$\subset$	a	$\vee$	(a, b)	$\subset$	$ab \cup (a, ab)$
		a	$\vee$	(b, cd)	$\subset$	(ab, acd)
abc	$\subset$	a	$\vee$	(a, bc)	$\subset$	$abc \cup (a, abc)$
ab	$\subset$	a	$\vee$	(b, ac)	$\subset$	$ab \cup (ab, ac)$
		a	$\vee$	(b, bc)	$\subset$	(ab, abc)
$\top \cup a \cup ab$	$\subset$	a	$\vee$	(a, ab)	$\subset$	$\top \cup a \cup ab \cup (a, ab)$
		a	$\vee$	(bc, de)	$\subset$	(abc, ade)
		a	$\vee$	(bc, bd)	$\subset$	(abc, abd)
		a	$\vee$	$(bc)^2$	$\subset$	$(abc)^2$
acd	$\subset$	a	$\vee$	(ab, cd)	$\subset$	$acd \cup (ab, acd)$
$ab \cup abc$	$\subset$	a	$\vee$	(ab, bc)	$\subset$	$ab \cup abc \cup (ab, abc)$
$\top \cup ab \cup ac$	$\subset$	a	$\vee$	(ab, ac)	$\subset$	$\top \cup ab \cup ac \cup (ab, ac)$
		ab	$\vee$	(c, d)	$\subset$	(abc, abd)
		ab	$\vee$	(c, de)	$\subset$	$(abc, abde)$
abc	$\subset$	ab	$\vee$	(c, cd)	$\subset$	$abc \cup (abc, abcd)$
abc	$\subset$	ab	$\vee$	(c, ad)	$\subset$	$abc \cup (abc, abd)$
		ab	$\vee$	(cd, ef)	$\subset$	$(abcd, abef)$
		ab	$\vee$	(cd, ce)	$\subset$	$(abcd, abce)$
		ab	$\vee$	$(cd)^2$	$\subset$	$(abcd)^2$
$abcd$	$\subset$	ab	$\vee$	(cd, ac)	$\subset$	$abcd \cup (abc, abcd)$
$\top \cup abc \cup abd$	$\subset$	ab	$\vee$	(ac, ad)	$\subset$	$\top \cup abc \cup abd \cup (abc, abd)$
$\top \cup abc \cup abd$	$\subset$	ab	$\vee$	(ac, bd)	$\subset$	$\top \cup abc \cup abd \cup (abc, abd)$
$abcd$	$\subset$	ab	$\vee$	(ac, cd)	$\subset$	$abcd \cup (abc, abcd)$

GraphSAT strategies

Given a looped-multi-hypergraph, we wish to ascertain its satisfiability status.

Strategy: Bruteforce

For a given graph G, check very CNF x_i supported on G. For every x_i, check every truth-assignment.

$$G \text{ is sat} \iff \forall x_i \in G,\ x_i \text{ is sat.}$$

G	CNF count
$v_1 v_2$	4
$v_1 v_2 v_3$	8
$(v_1 v_2 v_3), (v_1 v_3 v_4), (v_1 v_2 v_4)$	512
$(v_1 v_2 v_3), (v_1 v_3 v_4), (v_1 v_2 v_4), (v_3 v_4 v_5), (v_3 v_5 v_6), (v_4 v_5 v_6)$	262,144

GraphSAT strategies

Strategy: Apply reduction rules

Idea: simplify the graph before subjecting it to the bruteforce strategy.

For example, assume that we are given the following reduction rule:

$$X, (v_1 v_2 v_3), (v_1 v_2 v_4) \quad \longmapsto \quad X, (v_2 v_3 v_4)$$

where X is any graph not having edges incident on the vertex v_1.

GraphSAT strategies

Strategy: Apply reduction rules

Idea: simplify the graph before subjecting it to the bruteforce strategy.

For example, assume that we are given the following reduction rule:

$$X, (v_1 v_2 v_3), (v_1 v_2 v_4) \quad \longmapsto \quad X, (v_2 v_3 v_4)$$

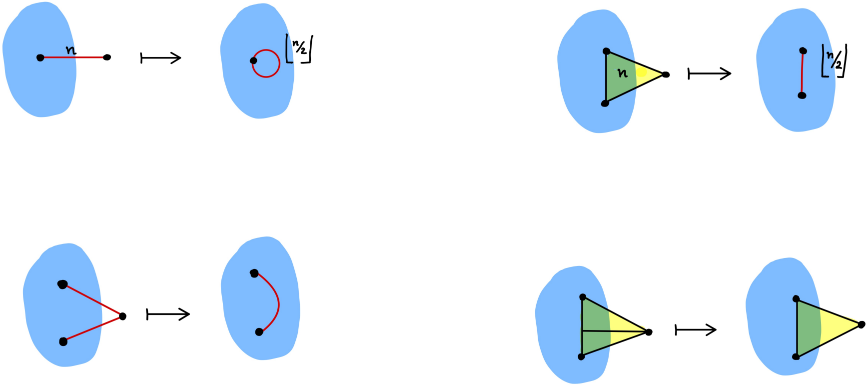
where X is any graph not having edges incident on the vertex v_1.

Proof sketch of reduction rule.

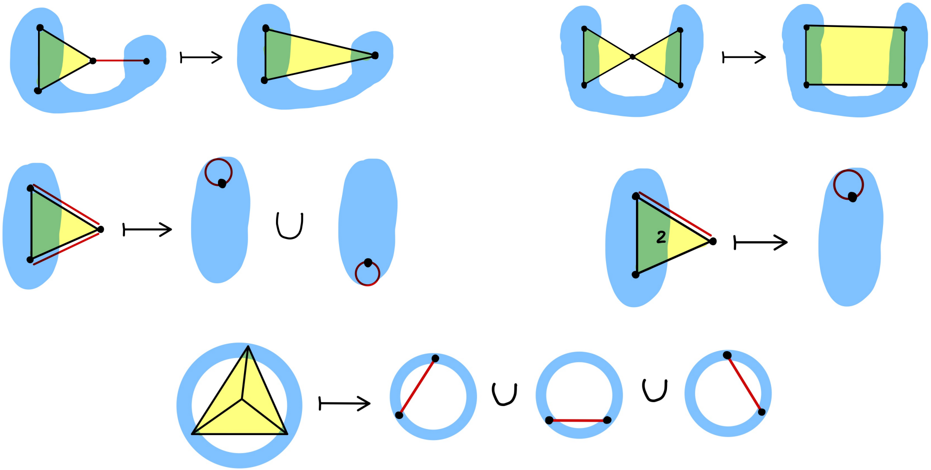
$$G[v] \;=\; \mathrm{sphere}(G, v), \mathrm{star}(G, v)[v] \;=\; \bigcup_{\substack{g_i \;\; h_i \;:\; \mathrm{Graph} \\ g_i, h_i \;=\; \mathrm{link}(G, v)}} \mathrm{sphere}(G, v), (g_i \vee h_i)$$

$$\begin{aligned} (X, 123, 124)[1] &= X, (23 \vee 24) \\ &\sim X, (\top \cup 234) \\ &= X \cup X, 234 \\ &\sim X, 234 \end{aligned}$$

Graph reduction rules



Graph reduction rules

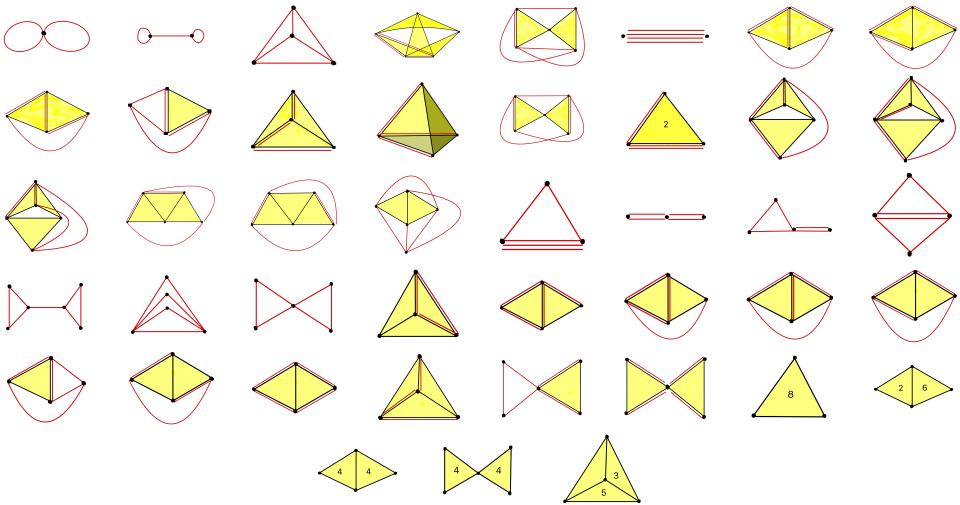


Minimal unsatisfiable looped-multi-hypergraphs

1. Start with all looped-multi-hypergraphs sorted smallest to largest.
2. Pick one and apply all known reduction rules to it.
3. Sat-check the reduced core of the graph.
4. If satisfiable, then pick the next one.
5. If unsatisfiable, then add to list of "minimal criminals".

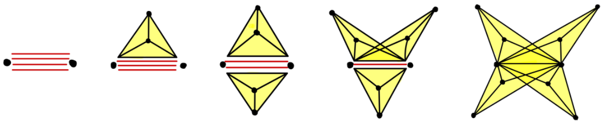
Number of connected graphs with less than 7 vertices	143
Number of minimal unsatisfiable irreducible simple graphs	4
Number of connected graphs with less than 6 vertices	10080
Number of minimal unsatisfiable irreducible L-M-H-graphs	202
Number of connected graphs with less than 7 vertices	48,364,386
Number of minimal unsatisfiable irreducible L-M-H-graphs	??

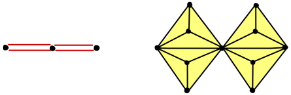
A selection of unsatisfiable looped-multi-hypergraphs



Thickening of edges

One of the reduction rules is $X, (v_1 v_2 v_3), (v_1 v_2 v_4), (v_2 v_3 v_4) \longmapsto X, (v_3 v_4)$,
where X is a graph having no edges incident on the vertices v_1 and/or v_2.
Applying this rule in reverse yields an easy way of creating (un)satisfiable hypergraphs.



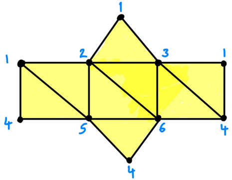


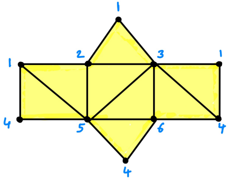
(Un)satisfiable triangulations

1. Tetrahedron is satisfiable.

(Un)satisfiable triangulations

1. Tetrahedron is satisfiable.
2. Triangulation of a triangular prism –





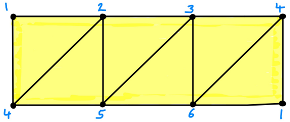

```
import mhgraph as mhg
import graph_rewrite as grw

prism1: mhg.MHGraph
prism1 = mhg.mhgraph([[1,2,3], [1,2,5], [1,3,4], [1,4,5], [2,3,6], [2,5,6], [3,4,6], [4,5,6]])
grw.decompose(prism1)

prism2: mhg.MHGraph
prism2 = mhg.mhgraph([[1,2,3], [1,2,5], [1,3,6], [1,4,5], [1,4,6], [2,3,6], [2,5,6], [4,5,6]])
grw.decompose(prism2)
```

Output:

```
(1, 2, 3),(1, 2, 5),(1, 3, 4),(1, 4, 5),(2, 3, 6),(2, 5, 6),(3, 4, 6),(4, 5, 6) is SAT
(1, 2, 3),(1, 2, 5),(1, 3, 6),(1, 4, 5),(1, 4, 6),(2, 3, 6),(2, 5, 6),(4, 5, 6) is SAT
```

(Un)satisfiable triangulations

3. Triangulation of a Möbius strip



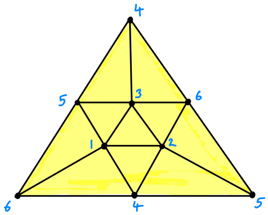

```python
import mhgraph as mhg
import graph_rewrite as grw

mobius_strip: mhg.MHGraph
mobius_strip = mhg.mhgraph([[1, 2, 4], [1, 4, 6], [2, 3, 5], [2, 4, 5], [3, 4, 6], [3, 5, 6]])
grw.decompose(mobius_strip)
```

Output:

```
(1, 2, 4),(1, 4, 6),(2, 3, 5),(2, 4, 5),(3, 4, 6),(3, 5, 6) is SAT
```

(Un)satisfiable triangulations

4. Triangulation of a Real projective plane ($\mathbb{RP}^2$)




```
import mhgraph as mhg
import graph_rewrite as grw

rp2: mhg.MHGraph
rp2 = mhg.mhgraph([[1, 2, 3], [3, 2, 6], [4, 6, 1], [4, 1, 2], [5, 2, 6],
                   [5, 6, 1], [1, 5, 3], [3, 6, 4], [4, 2, 5], [5, 3, 4]])
grw.decompose(rp2)
```

Output:

(1, 2, 3),(3, 2, 6),(4, 6, 1),(4, 1, 2),(5, 2, 6),(5, 6, 1),(1, 5, 3),(3, 6, 4),(4, 2, 5),(5, 3, 4) is SAT

(Un)satisfiable triangulations

5. Triangulation of a Klein bottle

Figure: The "242 triangulation" of a Klein bottle and its unsatisfiable subgraph.

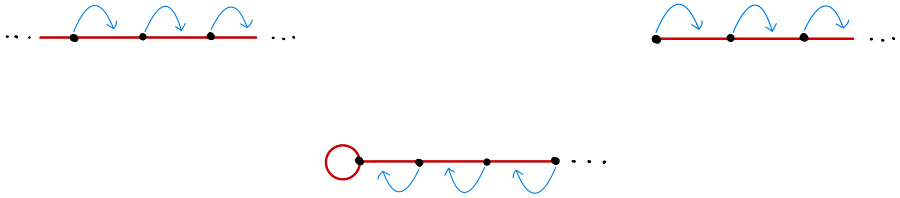
6. Triangulation of a Torus

Figure: Minimal triangulation of a torus and its unsatisfiable subgraph.

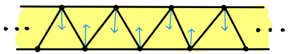
Satisfiable infinite graphs

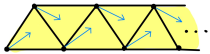
Recall,

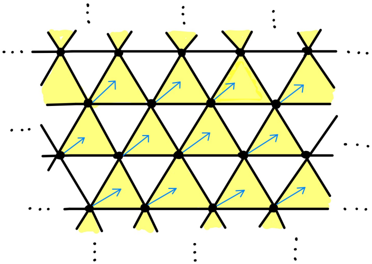
- Vertex $\equiv$ an element of a countably infinite set, i.e. points/nodes.
- Edge $\equiv$ a nonempty set of vertices, i.e. loops, simple connections, hyperedges.
- Graph $\equiv$ a nonempty set of edges, i.e. a looped-multi-hypergraph.



Satisfiable infinite graphs







Key results

1. Set of all satisfiable simple graphs
   $= $ Graphs forbidding $\{K_4,\ \text{butterfly graph},\ \text{bow-tie graph},\ K_{1,1,3}\}$ as topological minors.

2. Set of all satisfiable looped-multi-graphs
   $=$ Language of 2GraphSAT decision problem
   $=$ Graphs forbidding $\{K_4,\ \text{double-loop graph},\ \text{dumbbell graph},\ ab^4\}$ as topological minors.

3. There is a P-time algorithm for 2GraphSAT.

4. A list of graph reduction rules that preserve graph satisfiability.

5. The graph local rewriting theorem –

$$g[v] \;=\; \text{sphere}(g,v), \text{star}(g,v)[v] \;=\; \bigcup_{\substack{g_i\ h_i\ :\ \text{Graph} \\ g_i, h_i\ =\ \text{link}(g,v)}} \text{sphere}(g,v), (g_i \vee h_i)$$

6. An incomplete list of known unsatisfiable looped-multi-hypergraphs.

7. A Python package called `graphsat`.
   - : vaibhavkarve/graphsat
   - Supports Python version 3.9+, released under the GNU-GPL-v3.0 open license.
   - Functional-style, test-driven, literate programming, static type-checked.
   - Algorithms for SAT, GraphSAT, local rewriting, reduction of graphs under known rules, and searching for minimal unsatisfiable hypergraphs.

Conjectures and future work

1. We showed that the complexity class of 2GraphSAT is P. The complexity class for 3GraphSAT is not known. Moreover, the effect of local graph rewriting on 3GraphSAT is not known. Does local rewriting make the problem easier, or does it leave the complexity unchanged?

2. We have an incomplete list of unsatisfiable looped-multi-hypergraph.
   - **Conjecture 1**: The number of essential sat-invariant graph reduction rules is finite.
   - **Conjecture 2**: Each sat-invariant reduction rule is searchable in polynomial time.
   - **Conjecture 3**: The number of minimal unsatisfiable graphs under these reduction rules is also finite.

   If conjectures 1, 2, and 3 hold then we can conclude that 3GraphSAT is in P. This would give us an easy P-time heuristic check for 3SAT, simplifying some 3SAT cases. However, this will not affect the complexity class of 3SAT.

3. All unsatisfiable infinite graphs known so far have a finite unsatisfiable subgraph. Is there an unsatisfiable infinite graph whose every finite subgraph is satisfiable?
   Conjecture: No.

Conjectures and future work

4. Let $G_{a,b}$ denote the complete a-uniform hypergraph on b vertices. We can also think of this as the $(a-1)$-skeleton of a $(b-1)$-simplex.
   For example, we know that $G_{2,4} = K_4$ is unsatisfiable, while $G_{2,3} = C_3$ is satisfiable.

	b = 1	b = 2	b = 3	b = 4	b = 5	b = 6	b = 7	···
a = 1	sat	sat	sat	sat	sat	sat	sat	
a = 2	-	sat	sat	**unsat**	unsat	unsat	unsat	
a = 3	-	-	sat	sat	**unsat**	unsat	unsat	
a = 4	-	-	-	sat	sat	**sat**	??	
a = 5	-	-	-	-	sat	sat	??	
a = 6	-	-	-	-	-	sat	sat	
a = 7	-	-	-	-	-	-	sat	
···	-	-	-	-	-	-	-	

5. The generalized rule for n triangular hyperedges meeting at a common free vertex is not known. We know the reduction rule only for $n = 3$.

$$X, 123, 124, 134 \longmapsto X, 23 \cup X, 24 \cup X, 34$$

6. We have not explored random instances of GraphSAT, or graph analogues of random SAT.

Acknowledgments

Anil Hirani
$\wedge$ UIUC Campus Research Board
$\wedge$ University of Illinois at Urbana-Champaign
$\wedge$ Department of Mathamatics
$\wedge$ Merit Program for Emerging Scholars
$\wedge$ IISER-Kolkata
$\wedge$ Saugata Bandyopadhyay
$\wedge$ Emily
$\wedge$ Sungwoo
$\wedge$ Madhura
$\wedge$ Akshita
$\wedge$ TeX
$\wedge$ coffee
$\wedge$ Shekhar
$\wedge$ Anupama
$\wedge$ Nathan Dunfield
$\wedge$ Yuliy Baryshnikov
$\wedge$ Patryk Szuta
$\wedge$ `mem.math.illinois.edu`
$\wedge$ Richard Sowers
$\wedge$ Richard Laugesen
$\wedge$ Derrek
$\wedge$ Ravi
$\wedge$ Rishabh
$\wedge$ Vivek
$\wedge$ Emacs
$\wedge$ LaTeX
$\wedge$ $\neg$ SARS-Cov-2
$\wedge$ Madhavi
$\wedge$ Sanjay
$\wedge$ Anush Tserunyan
$\wedge$ David G. Bourgin family
$\wedge$ Illinois Geometry Lab
$\wedge$ Jennifer McNeilly
$\wedge$ Philipp Hieronymi
$\wedge$ Pranay Goel
$\wedge$ Simone
$\wedge$ Ciaran
$\wedge$ Neelotpal
$\wedge$ Mohit
$\wedge$ Org-mode
$\wedge$ Python
$\wedge$ Shreya
$\wedge$ Suresh
$\wedge$ Sukanya

References

[1] Niklas Eén and Niklas Sörensson. "An Extensible SAT-solver". In: *Theory and Applications of Satisfiability Testing*. Ed. by Enrico Giunchiglia and Armando Tacchella. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 502–518. ISBN: 978-3-540-24605-3.

[2] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. "PySAT: A Python Toolkit for Prototyping with SAT Oracles". In: *SAT*. 2018, pp. 428–437. DOI: 10.1007/978-3-319-94144-8_26. URL: https://doi.org/10.1007/978-3-319-94144-8_26.

[3] Vaibhav Karve and Anil N. Hirani. *GitHub: vaibhavkarve/graphsat*. Version v0.1.0. Apr. 2021. DOI: 10.5281/zenodo.4662169. URL: github.com/vaibhavkarve/graphsat.

[4] Vaibhav Karve and Anil N. Hirani. "The complete set of minimal simple graphs that support unsatisfiable 2-CNFs". In: *Discrete Applied Mathematics* 283 (2020), pp. 123–132. ISSN: 0166-218X. DOI: 10.1016/j.dam.2019.12.017.

[5] N. Robertson and P.D. Seymour. "Graph Minors XIII. The Disjoint Paths Problem". In: *Journal of Combinatorial Theory, Series B* 63.1 (1995), pp. 65–110. ISSN: 0095-8956.

[6] Neil Robertson and P.D. Seymour. "Graph minors I. Excluding a forest". In: *Journal of Combinatorial Theory, Series B* 35.1 (1983), pp. 39–61. ISSN: 0095-8956.

[7] Neil Robertson and P.D. Seymour. "Graph Minors XX. Wagner's conjecture". In: *Journal of Combinatorial Theory, Series B* 92.2 (2004). Special Issue Dedicated to Professor W.T. Tutte, pp. 325–357. ISSN: 0095-8956.

URLs

General notes on GraphSAT	vaibhavkarve.github.io/satisfiability
A copy of the thesis	vaibhavkarve.github.io/satisfiability/thesis.pdf
A copy of these slides	vaibhavkarve.github.io/satisfiability/slides.pdf
Code repository	github.com/vaibhavkarve/graphsat