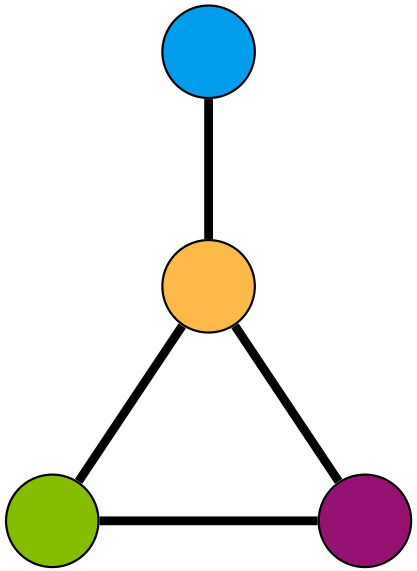
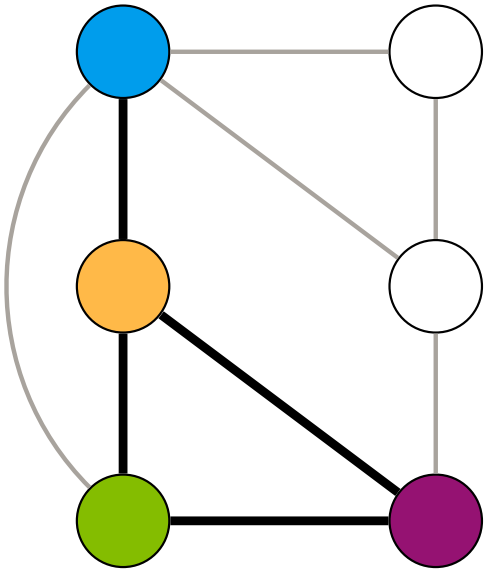
Finding Little Graphs Inside Big Graphs

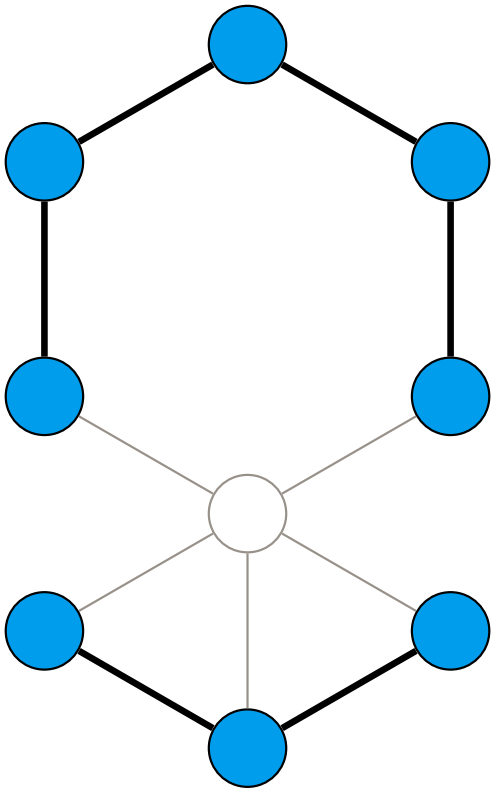
Ciaran McCreesh

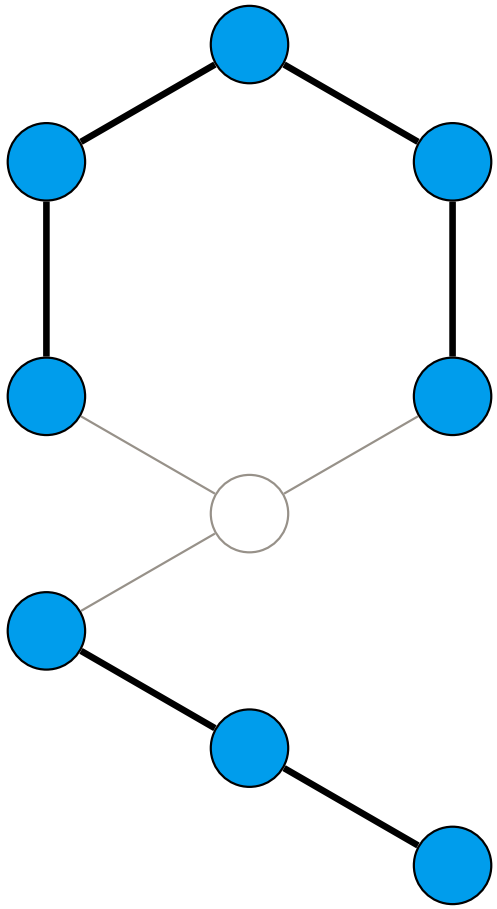
Subgraph Isomorphism



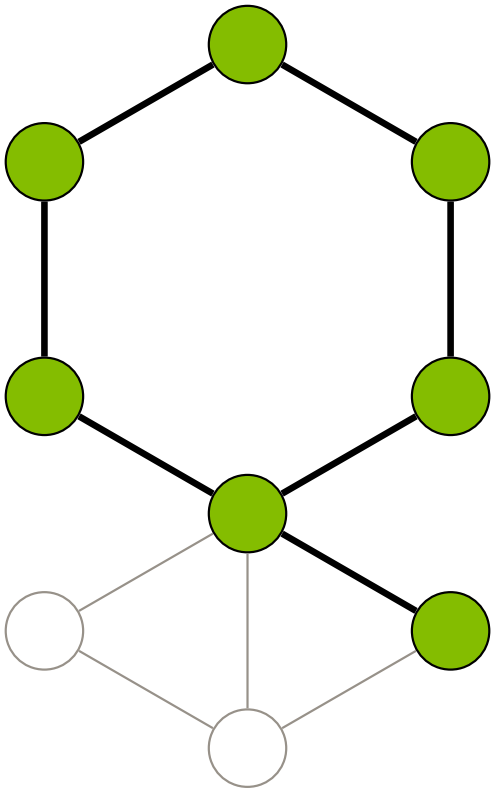


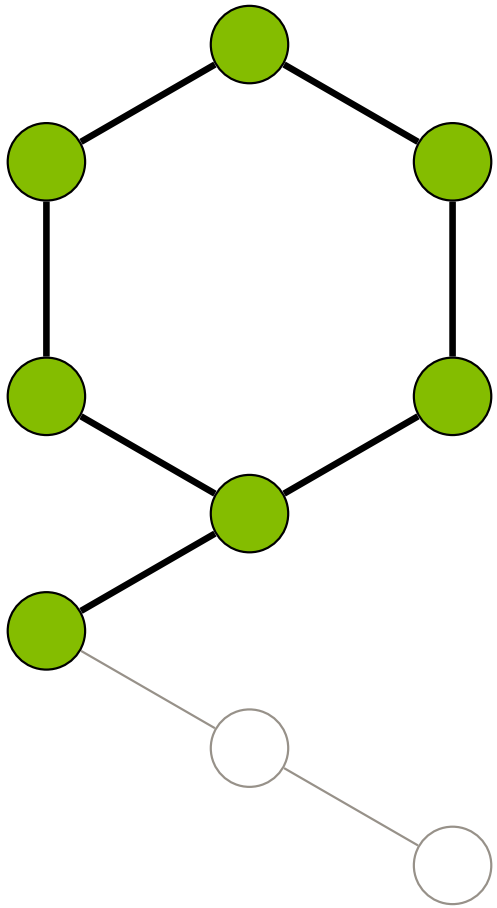
Maximum Common Induced Subgraph



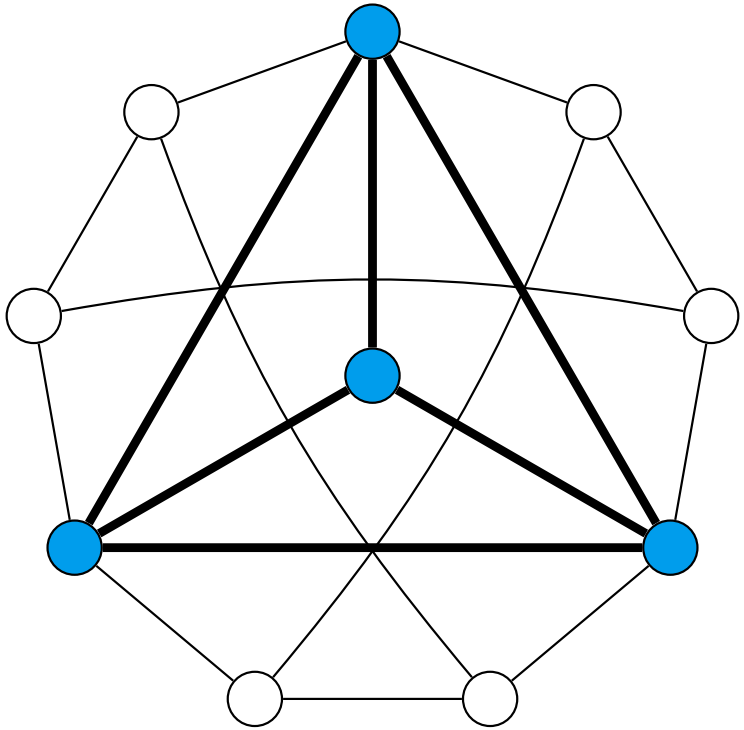


Maximum Common Induced Connected Subgraph





Maximum Clique



In Theory…

- Subgraph finding is hard.
- Subgraph counting is hard.
- Approximate subgraph finding is hard.

In Practice…

- We have good *solvers* for subgraph problems.
- Some applications involve solving thousands of subgraph isomorphism queries per second.
- We can solve clique on larger graphs than we can solve all-pairs shortest path.[1]

[1] Terms and conditions apply.

Popular Families of Subgraph Isomorphism Algorithm

- Connectivity-based:
  - VF2 (2004), VF3 (2017)
  - RI (2013)
- Constraint programming:
  - Ullman (1976)
  - LAD (AIJ 2010), SND (CP 2014), PathLAD (LION 2016)
  - Glasgow (CP 2015, LION 2016, CPAIOR 2019, ...)

Connectivity Algorithms

- Pick a pattern vertex and a target vertex.
- Recursively try to grow a mapping by looking at vertices connected to vertices that have already been used.

A Quick Introduction to Constraint Programming

- A declarative way of describing (hard) problems.
- A set of variables, each of which has a (finite) domain of values.
- A set of constraints (in any form we like, so arbitrary arity, non-linear, etc).
- Combining inference and clever backtracking search, give each variable a value from its domain, such that all constraints are respected.

Subgraph Finding, as a Constraint Program

- A variable for each pattern vertex. The domains are all of the target vertices.
- At least two sets of constraints:
  - Adjacent pairs of vertices must be mapped to adjacent pairs of vertices.
  - Injectivity, known as "all different".

Further Constraints We Can Deduce About Graphs

- A pattern vertex of degree k cannot be mapped to a target vertex of degree $k - 1$ or smaller.
  - We can also reason about neighbourhood degree sequences.
- Two pattern vertices that are distance d apart cannot be mapped to a pair of target vertices that are further than d apart.
  - We can also reason about the number of distinct short simple paths between two vertices.

Intelligent Backtracking Search

- It's a good idea to solve the hardest part of the problem first.
- For CP algorithms, branch on the smallest domain first, tie-breaking on the highest degree, and start by trying the highest degree target vertex first.
  - Then do more sneaky things involving slight randomisation, restarts, parallel search, ...
- For connectivity algorithms: try high degree, or rarest label?

The Glasgow Subgraph Solver

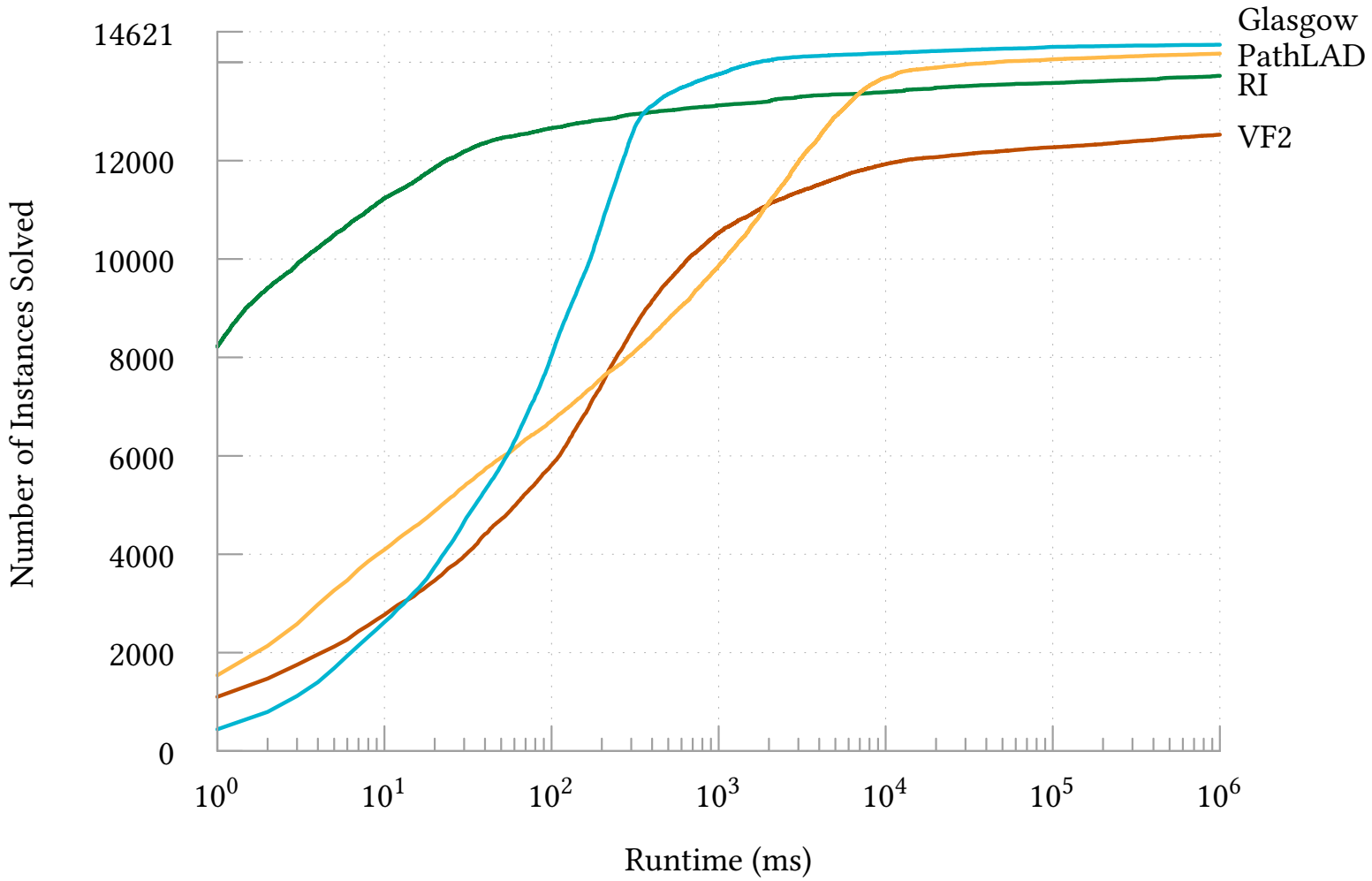
`https://github.com/ciaranm/glasgow-subgraph-solver`

- Subgraph isomorphism, and all its variants (induced / non-induced, homomorphism, locally injective, labels, side constraints, directed, ...).
- Also special algorithms for clique.

Benchmark Instances

- 14,621 instances from Christine Solnon's collection:
  - Randomly generated with different models.
  - Real-world graphs.
  - Computer vision problems.
  - Biochemistry problems.
  - Phase transition instances.
- At least...
  - $\geq$ 2,110 satisfiable.
  - $\geq$ 12,322 unsatisfiable.
- A lot of them are very easy for good algorithms.

Horse Race!


Glasgow
PathLAD
RI
VF2
Number of Instances Solved
14621
12000
10000
8000
6000
4000
2000
0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
Runtime (ms)

Easy Conclusion!

- CP is best!

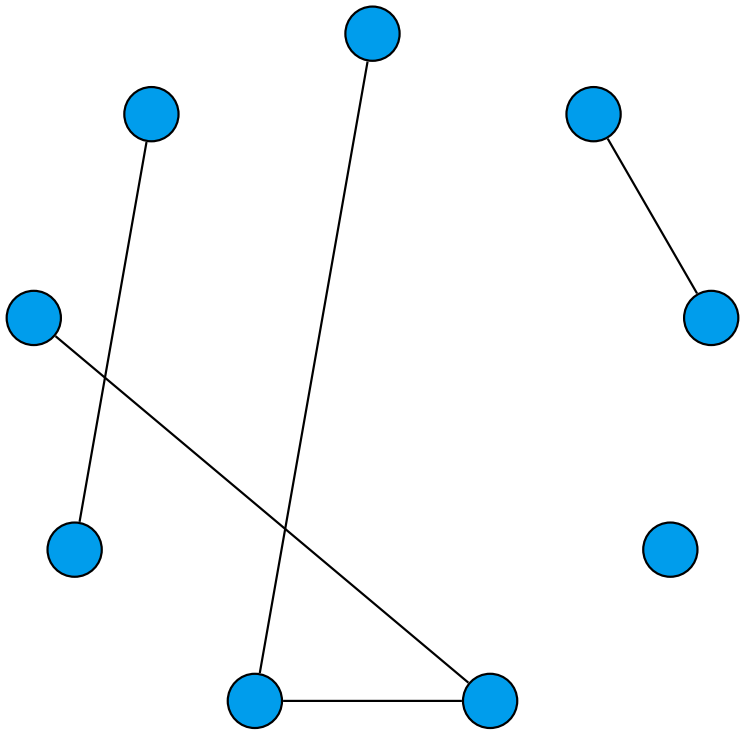
An Observation about Certain Datasets

- All of the randomly generated instances from the MIVIA suites are satisfiable.
- The target graphs are randomly generated, and patterns are made by selecting random connected subgraphs and permuting them.
- These instances are usually rather easy…
- Many papers use *only* these instances for benchmarking.

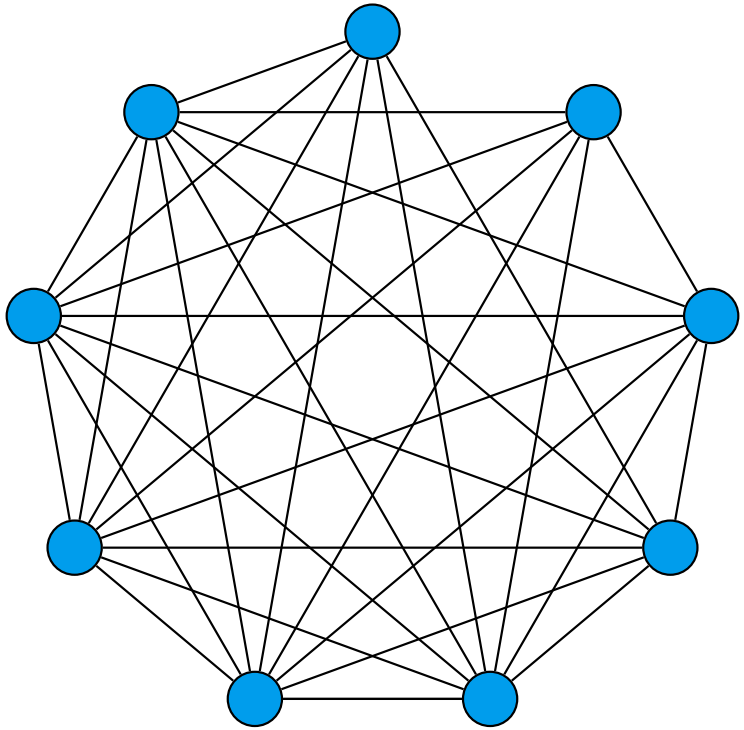
A Different Easy Conclusion!

- CP is slow! RI is best!

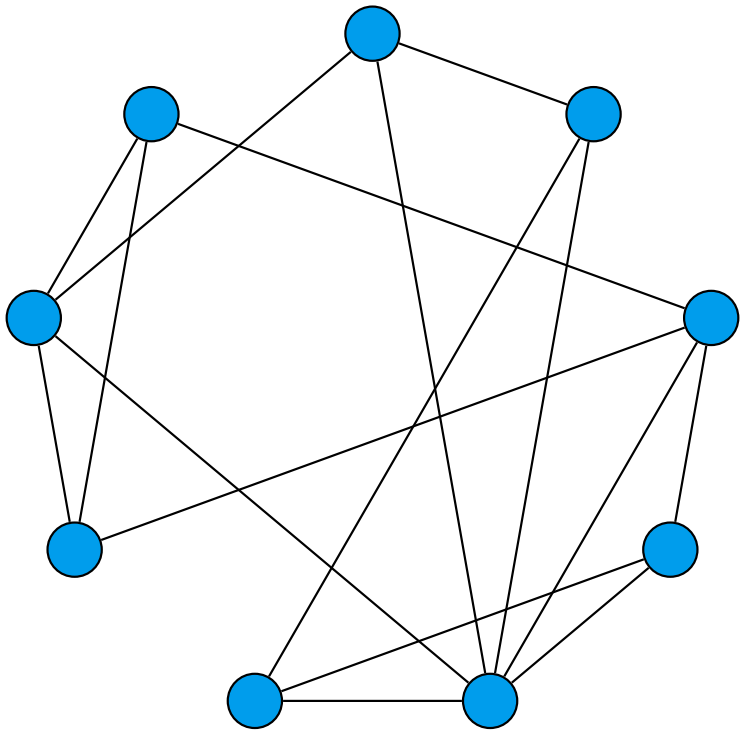
Is Clique-Finding Hard?



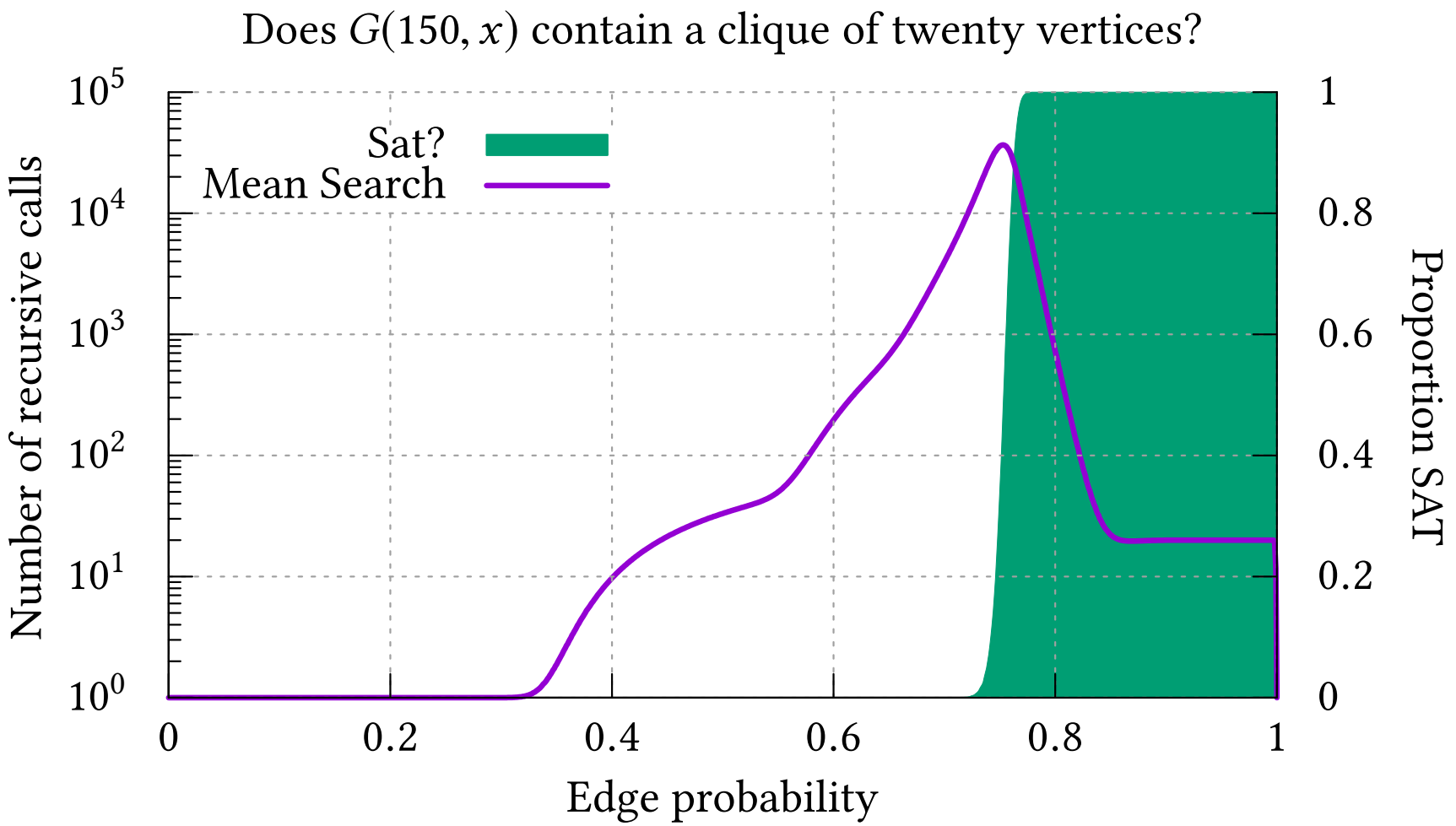
Is Clique-Finding Hard?



Is Clique-Finding Hard?



Cliques in Random Graphs



Intuition

- High density means lots of occurrences, so wherever we look, it's easy to find one of them.[2]
- Low density means no occurrences, and we can quickly show we run out of edges after doing a bit of branching.
- If we expect there to be just one solution, it's really hard to find it if it exists, and really hard to rule it out if it doesn't exist.

[2] This statement is technically a massive lie.

So What?

- This has practical implications in solver design, and in designing systems built around solvers.
- There's a lot more to hardness than worst-case complexity analysis.
- Understanding how algorithms behave is important.
- Maybe we should try doing some science?

http://github.com/ciaranm

ciaran.mccreesh@gmail.com