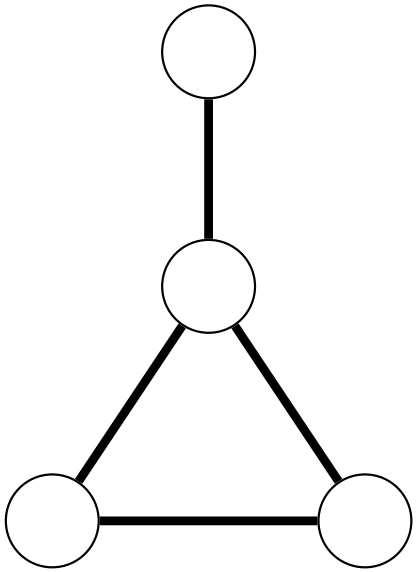
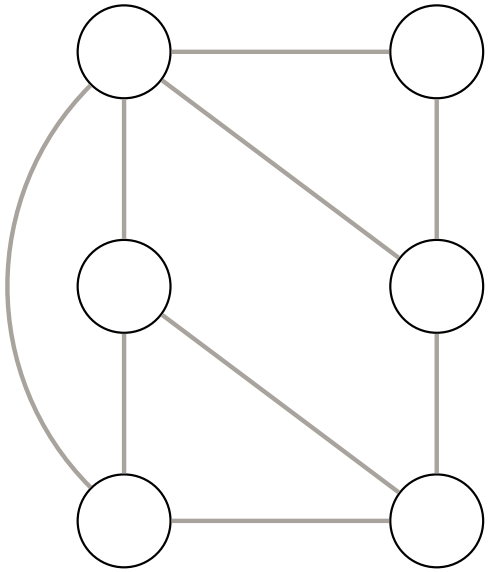
The Glasgow Subgraph Solver

Ciaran McCreesh
Patrick Prosser
James Trimble

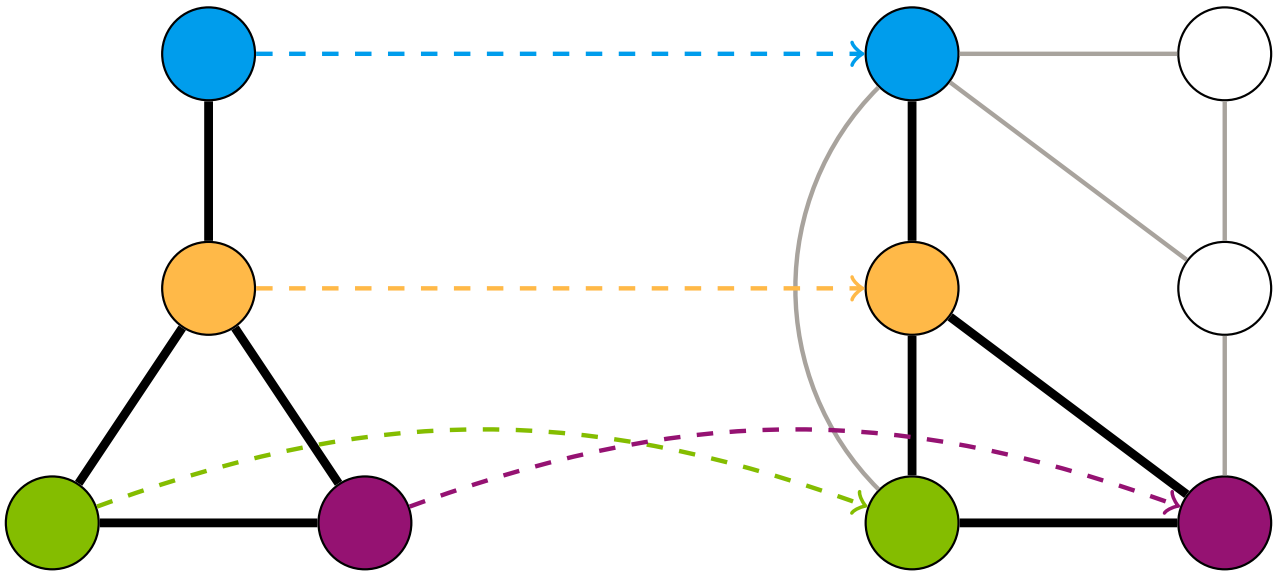
University of Glasgow

Subgraph Isomorphism





Subgraph Isomorphism



Who Else Cares?

- Chemistry, biochemistry, and drug design (graphs are molecule fragments or proteins).
- Computer vision.
- Compilers (instruction generation, code rewriting).
- Plagiarism and malware detection.
- Livestock epidemiology (contact and trade graphs).
- Designing mechanical lock systems.

What Exactly is the Problem?

- Non-induced or induced?
- Directed edges? Multi-edges? Loops?
- Injective? Locally injective? Non-injective?
- Labels on vertices? Labels on edges?
- Nested graphs?
- Wildcards? Partial matching?
- Connected?

In Theory…

- Subgraph finding is hard.[1]
- Subgraph counting is hard.
- Approximate subgraph finding is hard.

[1] For an arbitrary pattern.

In Practice...

- We have good *solvers* for subgraph problems.
- We can solve clique on larger graphs than we can solve all-pairs shortest path.[2]

[2] Terms and conditions apply.

The Glasgow Subgraph Solver

/ciaranm/glasgow-subgraph-solver

- MIT licence.
- A constraint programming solver, but specifically for subgraph problems.
- Subgraph isomorphism and its many variants (induced / non-induced, homomorphism, locally injective, labels, side constraints, directed, …), maximum common subgraph, and clique.

Key Features

- Really good performance on “hard” instances, e.g. large pattern graphs.
- Fairly easy to add support for new problem variants.
- Absolutely definitely totally 100% bug-free.

The Rest of this Talk

- Finding subgraphs, from a constraint programming perspective.
- A bit about the implementation.
- Ongoing and future work.

The Idea

- A set of *variables*, each of which has a finite *domain* of possible values.
- A set of *constraints*.
- Give each variable a value from its domain, respecting all constraints.
- Solvers based upon *inference* and *intelligent backtracking search*.

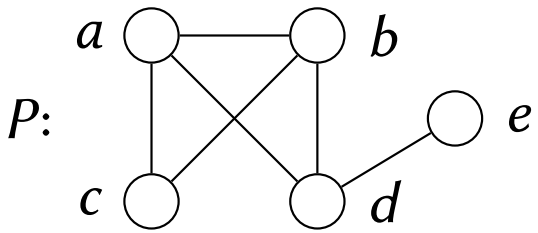
Graph Mapping Problems

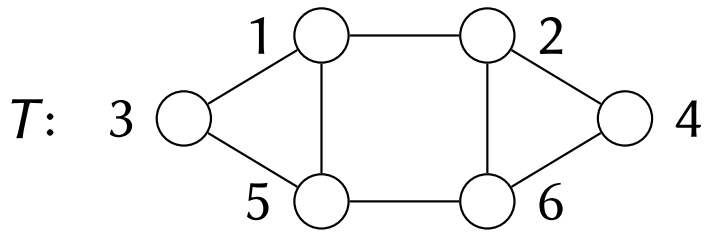
- Variables are pattern graph vertices.
- Values are target graph vertices.
- Constraints for adjacency and injectivity.

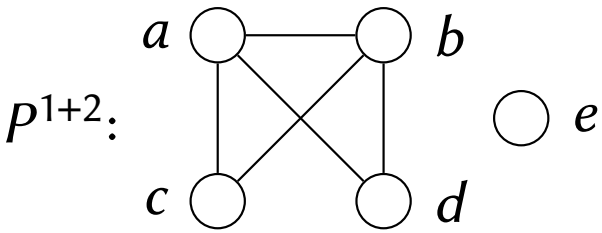
Supplemental Constraints

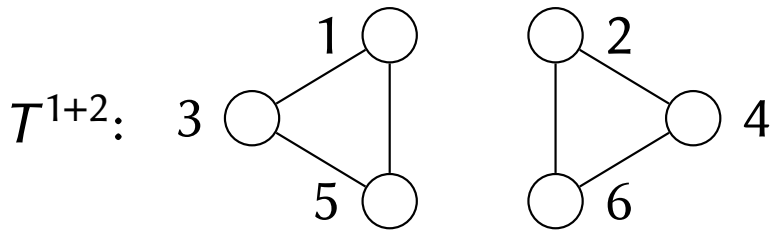
- Extra constraints using degrees, degree sequences, …
- Can reason about paths, not just adjacency.
  - Functors!

Supplemental Constraints









Search

- Branch on pattern vertices with *fewest possibilities* and *most constraints*.
- Aggressive restarts and biased randomisation.

Side Constraints

- Arbitrary additional "application" constraints.
- More constraints usually means better performance.

A Dedicated Constraint Solver

- "Modern" C++.
- Heavy use of bit-parallelism.
  - A full round of inference and propagation on a 40 vertex pattern, 200 vertex target in 200ns.
- Primarily run as an external process, unless your host programming language can deal with threads and memory management.

Proof Logging

- Conventional testing techniques don't work on solvers.
- We can output a formally verifiable proof log in pseudo-Boolean cutting planes format.
- Doesn't guarantee we're bug-free, but if we ever output an incorrect answer, it can be detected.

Problem Variants

- Bigraphs (with sharing).
  - Can only map to some target vertices if there are no "extra" unmapped edges.
- Matching with arbitrary side constraints.
  - Highly experimental high level modelling language support.
- Automatically detecting polynomial variants.

Symmetries and Other Ways of Counting Faster

- For traditional constraint programming, counting is enumeration.
- We can be exponentially faster in some situations.

Automatic Configuration and Benchmarking

- We need better benchmark instances!
- Even if they're really easy.
- Or if they are so hard you can't solve them at all.
- Or if they have side constraints.

http://www.dcs.gla.ac.uk/~ciaran
ciaran.mccreesh@glasgow.ac.uk

University of Glasgow