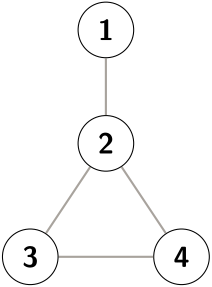
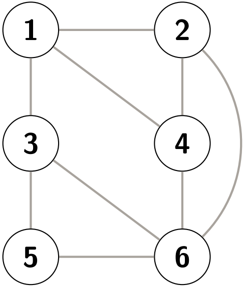
Finding Little Graphs Inside Big Graphs (in Parallel)

Ciaran McCreesh and Patrick Prosser

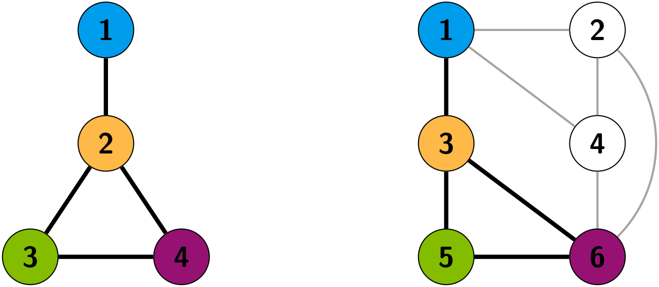
University of Glasgow

Subgraph Isomorphism





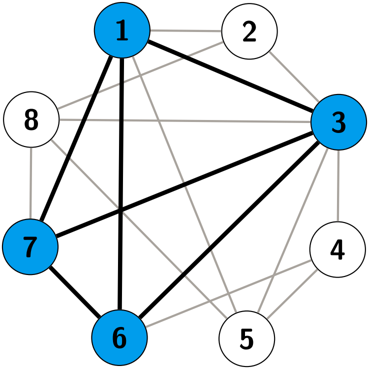
Subgraph Isomorphism



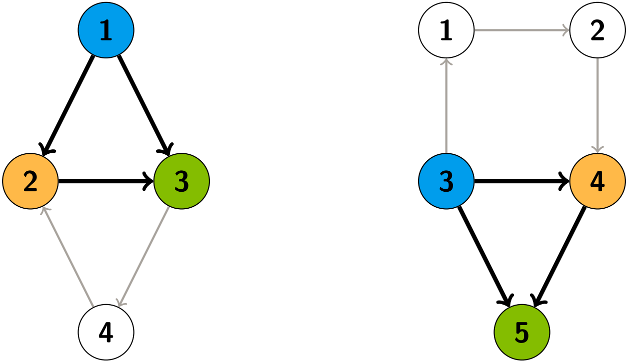
Subgraph Isomorphism

- Find an *injective* mapping from a *pattern* graph to a *target* graph.
- **Adjacent** vertices must be **mapped to adjacent** vertices.
- For the *induced* problem variant, non-adjacent vertices must be mapped to non-adjacent vertices.

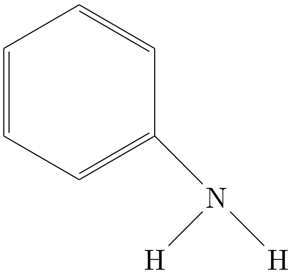
The Maximum Clique Problem

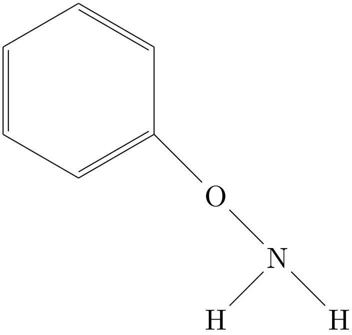


Maximum Common Subgraph



Maximum Common Connected Subgraph?





Who Cares?

- Bioinformatics
- Chemistry
- Drug design
- Computer vision
- Pattern recognition
- Financial fraud detection
- Model checking
- Fault detection
- Law enforcement
- Kidney exchange
- Social network analysis
- Compilers
- Diseased cows
- Computer algebra
- Circuit design
- Network design

Practical Algorithms

- Real-world inputs rarely have nice properties (low treewidth, particular degree spreads that are polynomial, etc).
- We can still solve some subgraph isomorphism problems with **thousand vertex patterns** and **ten thousand vertex targets** in a few seconds.
- Worst-case analysis is useless, and constant factors matter.

Constraint Models

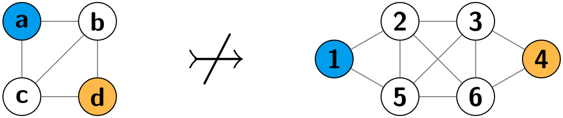
- We have some **variables**, each with a **domain**, and we want to give each variable a value from its domain.
  - Subgraph isomorphism: a variable for each pattern vertex, with domains being target vertices.
  - Clique: a boolean variable for each vertex.
- There are **constraints** between variables.
  - Subgraph isomorphism: all-different (injectivity), and adjacent pairs of vertices must be mapped to adjacent pairs of vertices.
  - Clique: for each pair of non-adjacent vertices, at least one of the two variables must be false.
- There is an **objective**.
  - Subgraph isomorphism: give each variable a value.
  - Maximum clique: set as many variables to true as possible.

Inference

- We want to **cross out values** from domains, until only one value is left in each.
- Subgraph isomorphism: high degree vertices cannot be mapped to low degree vertices.
- If an assignment becomes forced, we can **infer additional deletions**. This can have a cascade effect.

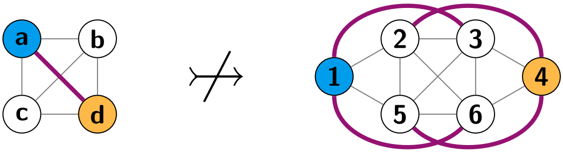
Implied Constraints for Subgraph Isomorphism

- Adjacent vertices must be mapped to adjacent vertices.
- Vertices that are distance 2 apart must be mapped to vertices that are within distance 2.
- Vertices that are distance k apart must be mapped to vertices that are within distance k.



Implied Constraints for Subgraph Isomorphism

- G^d is the graph with the same vertex set as G, and an edge between v and w if the distance between v and w in G is at most d.
- For any d, a subgraph isomorphism $i : P \rightarrowtail T$ is also a subgraph isomorphism $i^d : P^d \rightarrowtail T^d$.



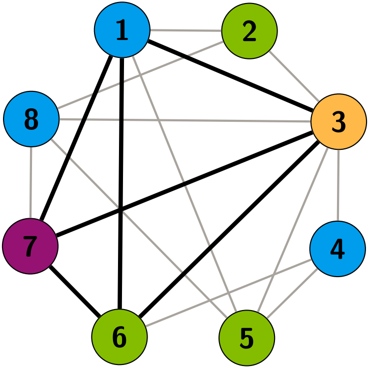
Implied Constraints for Subgraph Isomorphism

- We can do something stronger: rather than looking at distances, we can look at **(simple) paths**, and we can count how many there are.
- This is NP-hard in general, but only lengths 2 and 3 and counts of 2 and 3 are useful in practice.
- We construct these graph pairs once, at the top of search.
- We can also use these graph pairs for degree-based filtering.

Search

- Sometimes we have to **guess**: pick a variable x. Then for each value v_i in its domain in turn, see what happens if we force $x = v_i$.
- There are good heuristics telling us which variable to pick first.
- There are heuristics telling us which value to pick first, but this seems to be less reliable in general.

Branch and Bound

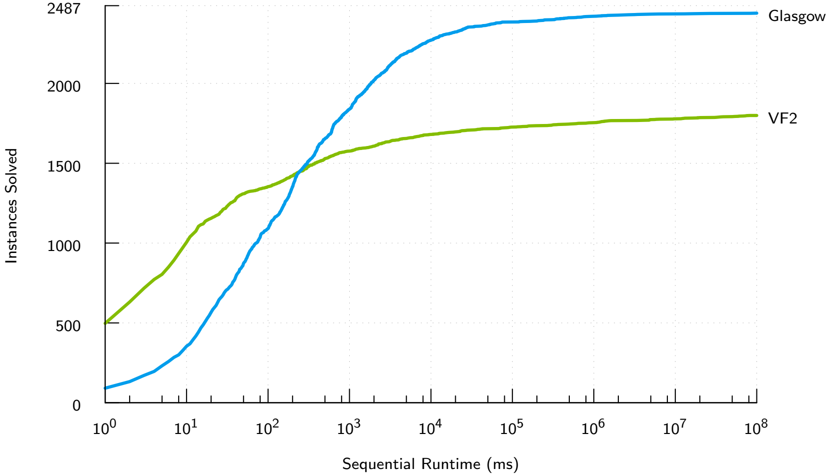


- For optimisation: keep track of the **best solution** we've found so far. If we can show we can't beat it, backtrack immediately.

Backjumping

- When backtracking, see if the current assignment actually removed any values which could have helped prevent the failure. If not, **jump back** another step.

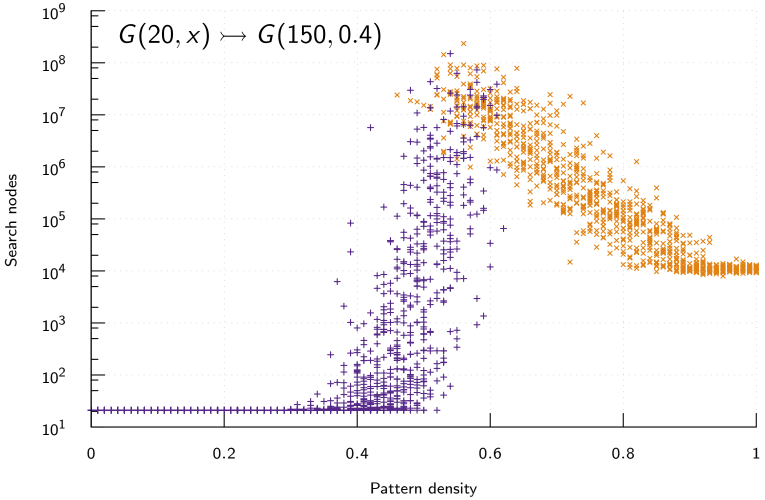
Is This Any Good?



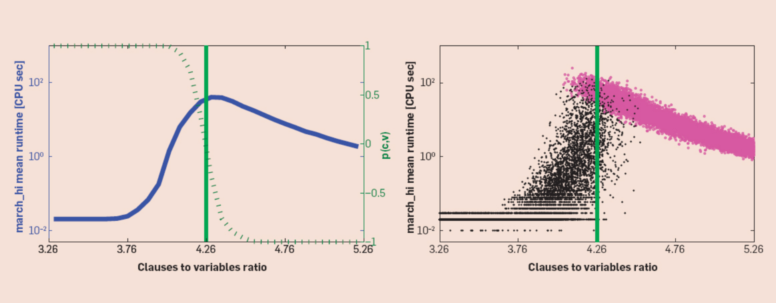
Generating Hard Subgraph Isomorphism Instances

- We can solve some problem instances with a thousand pattern vertices, and ten thousand target vertices. Can we solve *any* instance with these sizes?
- We like having **lots of instances**, to make sure we don't overfit algorithm parameters.
- How do we create random subgraph isomorphism instances?

Phase Transitions in Non-Induced Subgraph Isomorphism

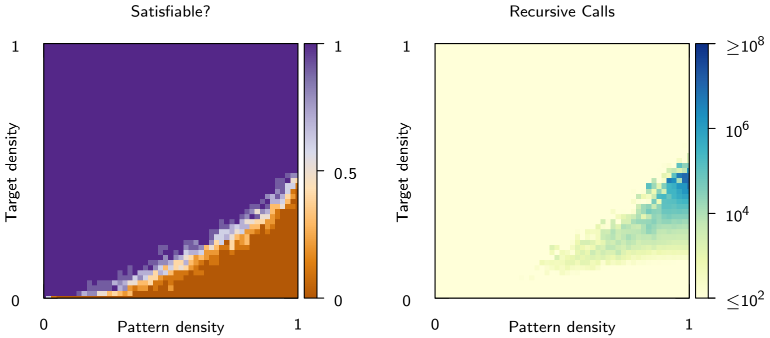


This Looks Familiar. . .



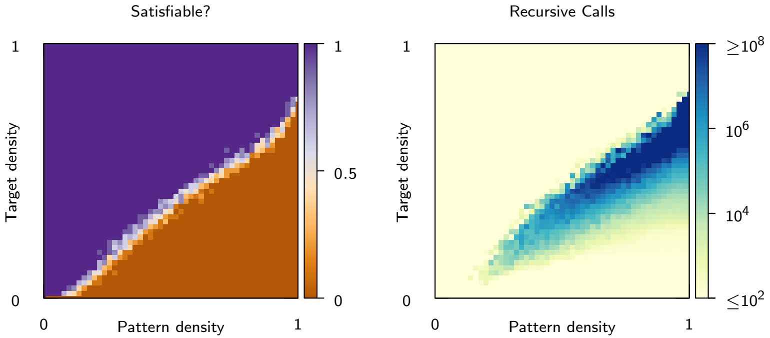
Understanding the Empirical Hardness of NP-Complete Problems.
Kevin Leyton-Brown, Holger H. Hoos, Frank Hutter, Lin Xu.
Communications of the ACM, Vol. 57 No. 5, Pages 98-107

Varying Both Densities?



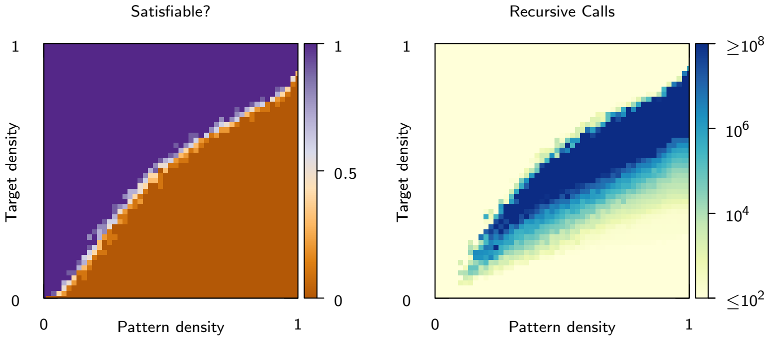
$$G(10, x) \rightarrowtail G(150, y)$$

Varying Both Densities?



$$G(20, x) \rightarrowtail G(150, y)$$

Varying Both Densities?



$$G(30, x) \rightarrowtail G(150, y)$$

Heuristics from Maximising Expectations

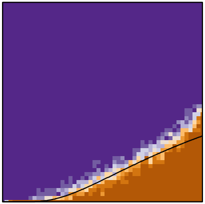
- With a few dubious assumptions regarding independence and integers, the expected number of solutions is

$$\langle Sol \rangle = t \cdot (t-1) \cdot \ldots \cdot (t-p+1) \cdot {d_t}^{d_p \cdot \binom{p}{2}}.$$

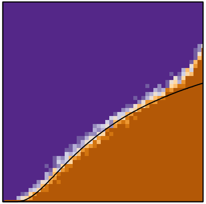
- If $\langle Sol \rangle \ll 1$, the instance is likely to be unsatisfiable.
- Unfortunately, if $\langle Sol \rangle \gg 1$, things are a bit trickier...

Heuristics from Maximising Expectations

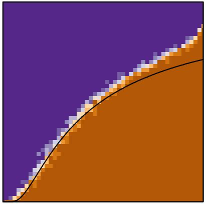
$G(10, x) \rightarrowtail G(150, y)$



$G(20, x) \rightarrowtail G(150, y)$



$G(30, x) \rightarrowtail G(150, y)$



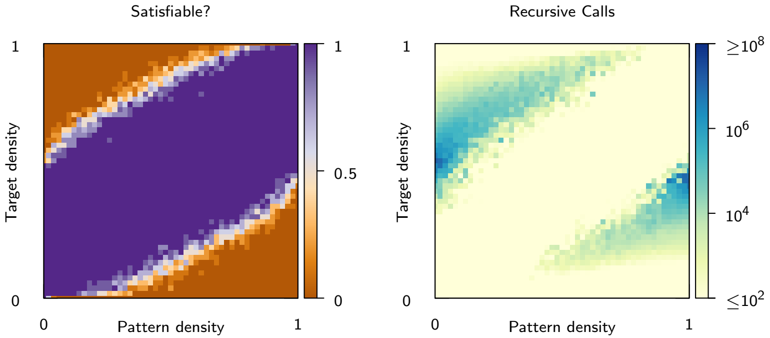
Heuristics from Maximising Expectations

- Suppose we wanted to maximise the expected number of solutions in a subproblem during search.

$$\langle Sol \rangle = \underbrace{t \cdot (t-1) \cdot \ldots \cdot (t-p+1)}_{\text{smallest domain}} \cdot \underbrace{d_t}_{\substack{\text{low}\\ \text{degree}}} {}^{\underbrace{d_p \cdot \binom{p}{2}}_{\substack{\text{high}\\ \text{degree}}}}$$

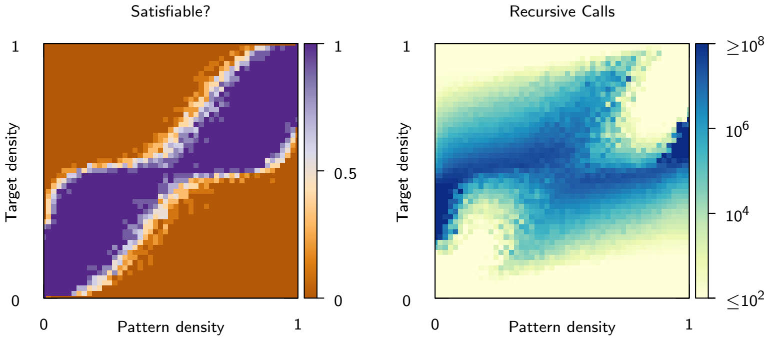
- These heuristics are used in practice (sort of), but were discovered through guessing and experiments!

Phase Transitions in Induced Subgraph Isomorphism



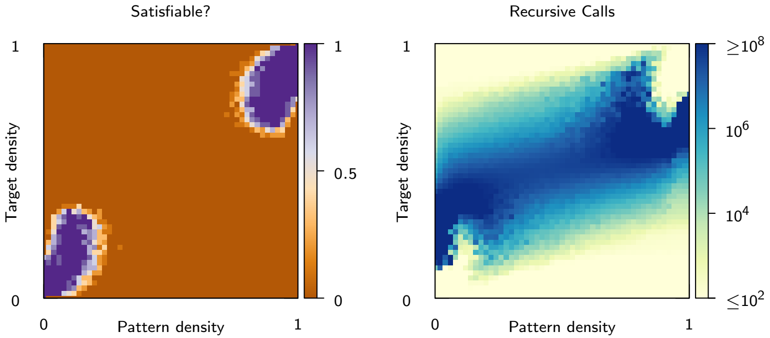
$$G(10, x) \hookrightarrow G(150, y)$$

Phase Transitions in Induced Subgraph Isomorphism



$$G(15, x) \hookrightarrow G(150, y)$$

Phase Transitions in Induced Subgraph Isomorphism



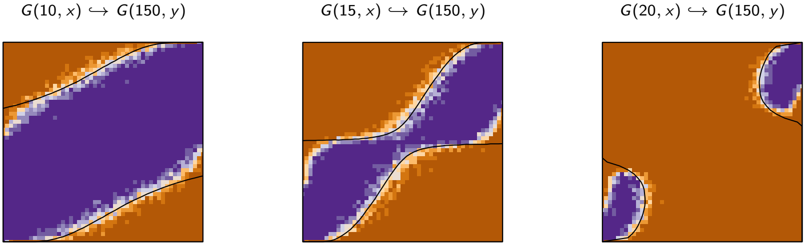
$$G(20, x) \hookrightarrow G(150, y)$$

Can We Predict This?

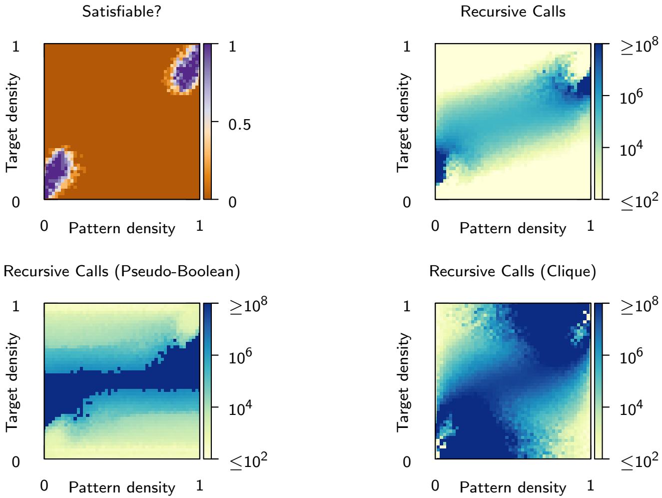
- With a few more dubious assumptions, the expected number of solutions is now

$$\langle Sol \rangle = t \cdot (t-1) \cdot \ldots \cdot (t-p+1) \cdot {d_t}^{d_p \cdot \binom{p}{2}} \cdot (1-d_t)^{(1-d_p) \cdot \binom{p}{2}}.$$

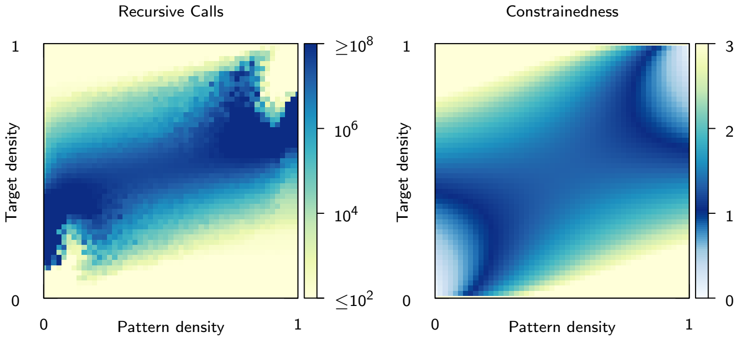
Can We Predict This?



Why is the Middle Region Hard?



Why is the Middle Region Hard?

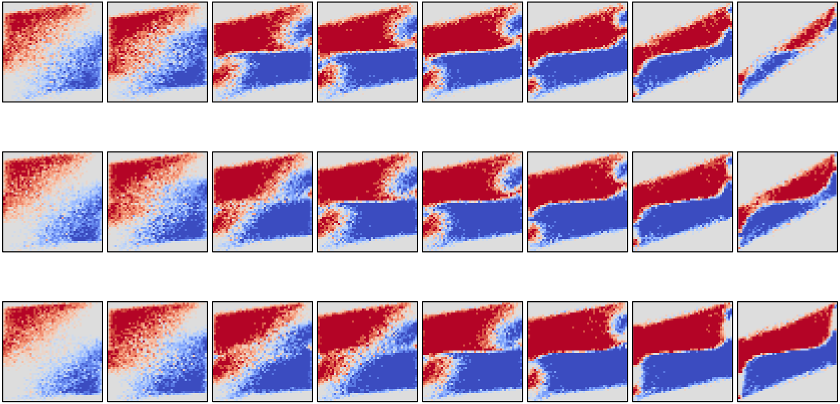


$$G(20, x) \hookrightarrow G(150, y)$$

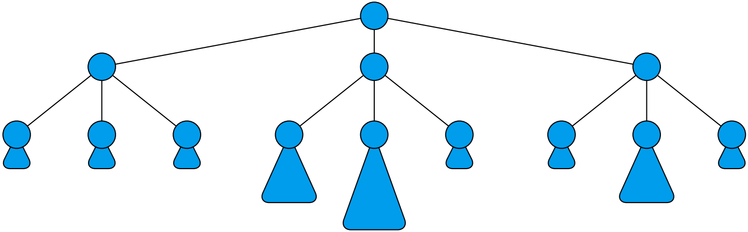
Induced Heuristics?

- For anything we say about degree, the opposite holds for the complement constraints.
- Degree-based value ordering heuristics don't seem to help. This is intuitive, but does this formula give us a heuristic after all?

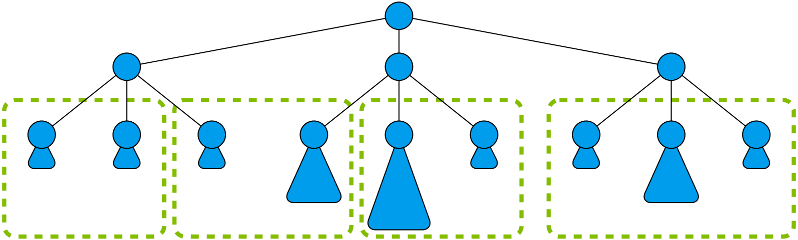
Induced Heuristics?



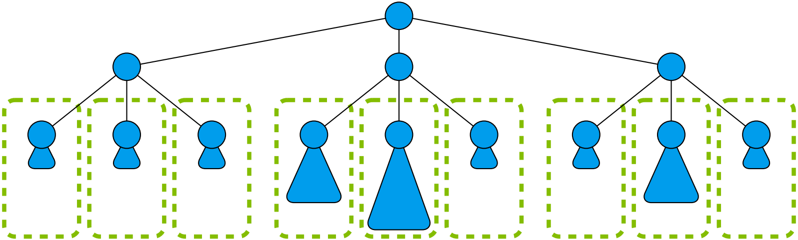
Thread-Parallel Tree Search



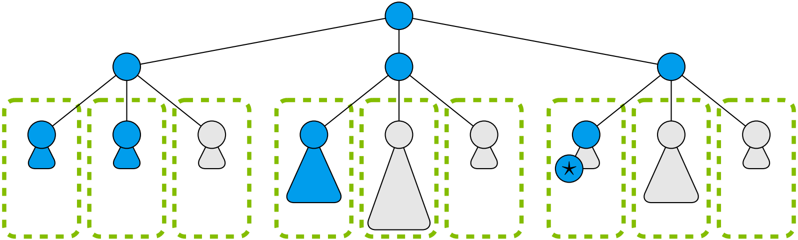
Thread-Parallel Tree Search



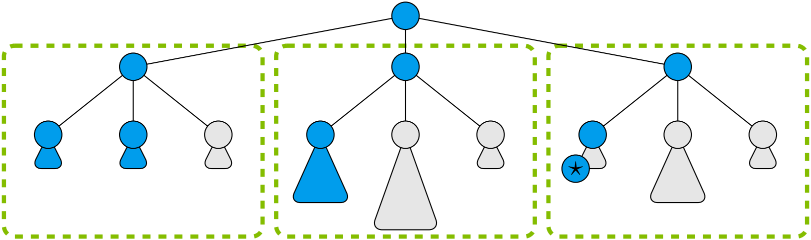
Thread-Parallel Tree Search



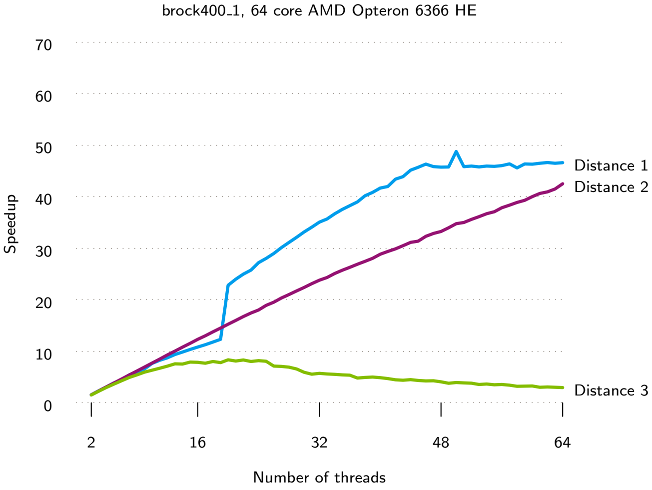
Parallel Search Order Matters



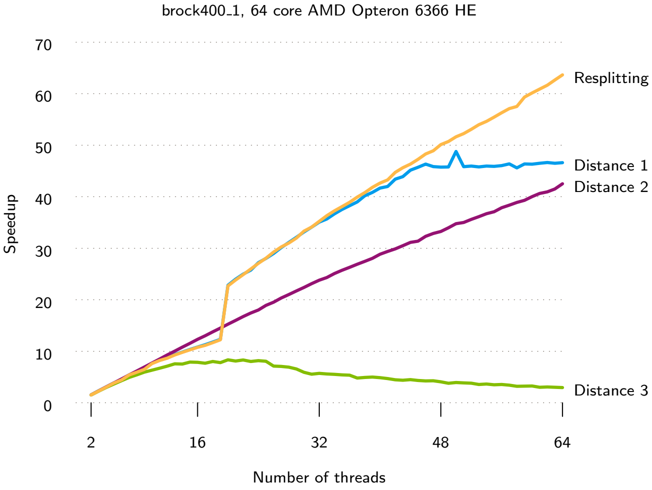
Parallel Search Order Matters



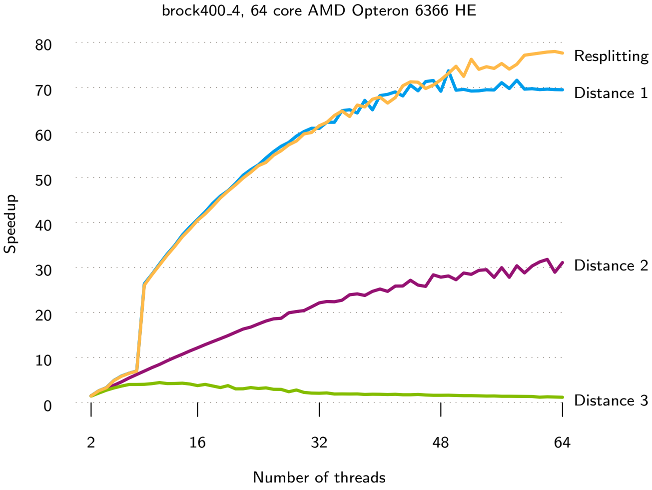
Parallel Search Order Matters



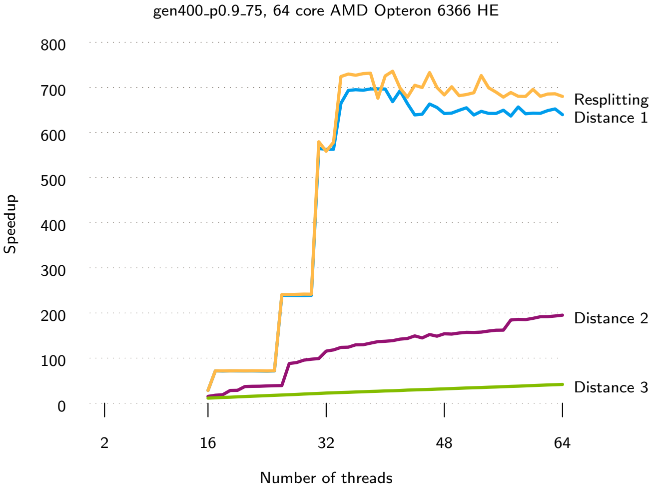
Parallel Search Order Matters



Parallel Search Order Matters



Parallel Search Order Matters



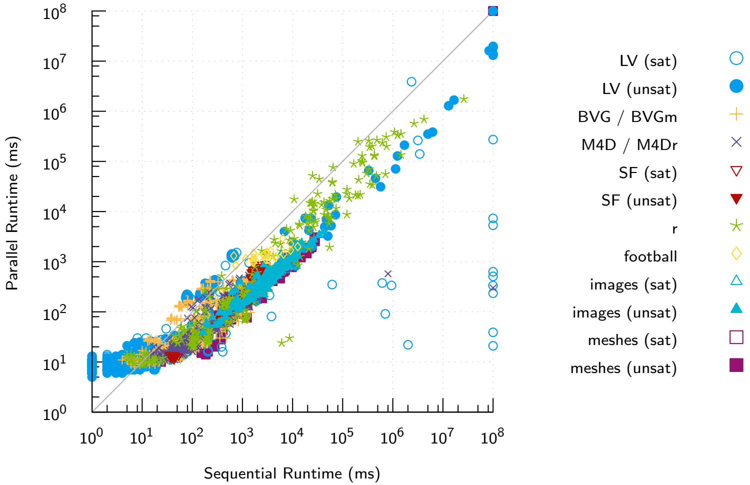
Parallel Search Order Matters

- Value-ordering heuristics tend to be **worst high up** the search tree.
- But depth-first searches commit completely to the first choice made. . .
- Discrepancy searches can avoid this problem by doing more work in total. Parallel search can give **similar benefits for free**.

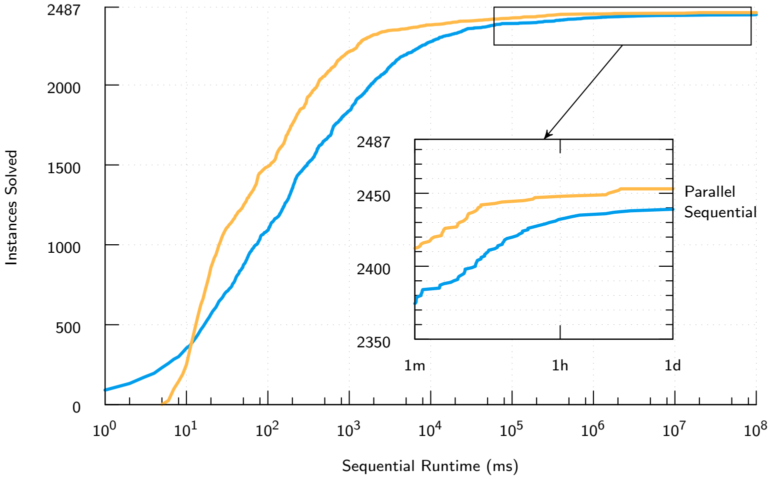
Safety and Reproducibility

- My “wish list”:
  1. Parallel search should **not be substantially slower** than sequential search.
  2. Adding more processors should **not make things substantially worse**.
  3. Running the same program twice on the same hardware should give **similar runtimes**.
- This is surprisingly tricky.
- On top of all that, we want to **prioritise work stealing** from where we’re most likely to be wrong, or possibly from where we’re most likely not to eliminate a subtree.

Parallel Search is Worth Doing



Parallel Search is Worth Doing



Describing and Implementing Parallel Search

- Implementing safe and reproducible parallel search by hand, even just for multi-core, is painful.
- Current high level approaches don't offer the properties we need.
- Is there a better way?

Symmetries

- Some graphs have known symmetries. Can we exploit this?
  - In some ways, maximum clique is just a completely symmetric version of maximum common subgraph.
- What about if we have to detect the symmetries ourselves dynamically?

Explaining Failures

- Backjumping works because when we fail, we **work out why**, and use that to backtrack further.
- But then we throw that information away...
- CNF encodings for graph problems tend to be annoyingly big, and lose structural information.

Graph Algorithms and Optimisation

- There are a lot of real-world optimisation problems involving **a graph problem** (subgraph isomorphism, subgraph covering, finding sequences of related subgraphs, clique finding, graph colouring, . . .), plus some **other constraints**.
- Can we make these problems easier to specify in a **high-level constraint modelling language** like ~~Essence'~~ MiniZinc?
- There is a continuum of what we could do with these models:
  - Compile to CP, MIP or SAT (but these models tend to be large, and lose structural and heuristic information).
  - A hybrid, multi-solver approach, "graph morphisms modulo theories" style (but we need better theories).
  - Compile to subgraph isomorphism (but even simple arithmetic constraints become disgusting under reduction).

http://dcs.gla.ac.uk/~ciaran
c.mccreesh.1@research.gla.ac.uk