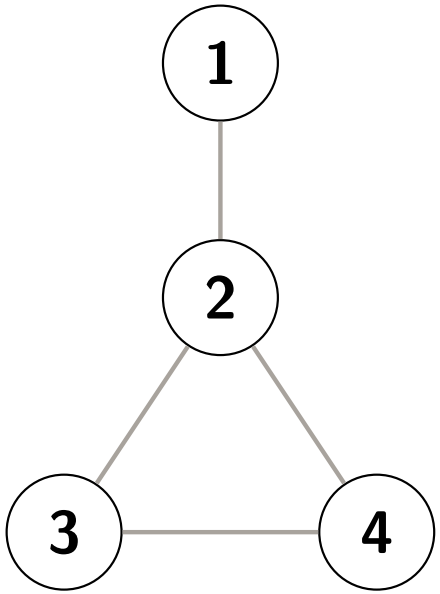
Finding Little Graphs Inside Big Graphs (In Parallel)

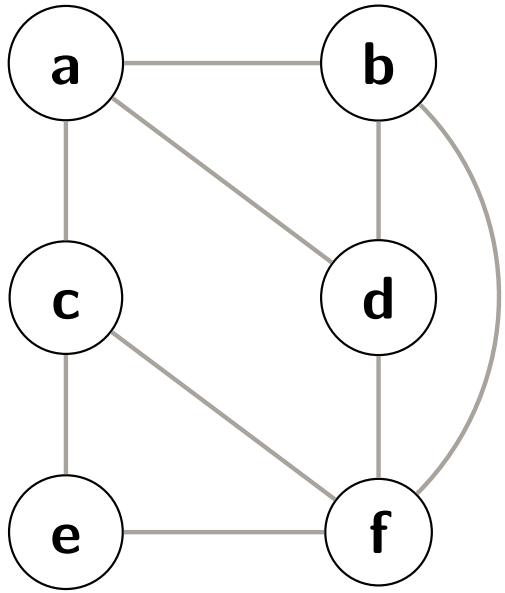
Ciaran McCreesh

with contributions from Ruth Hoffmann, Lars Kotthoff, Samba Ndojh Ndiaye, Patrick Prosser, Craig Reilly, Christine Solnon, and James Trimble

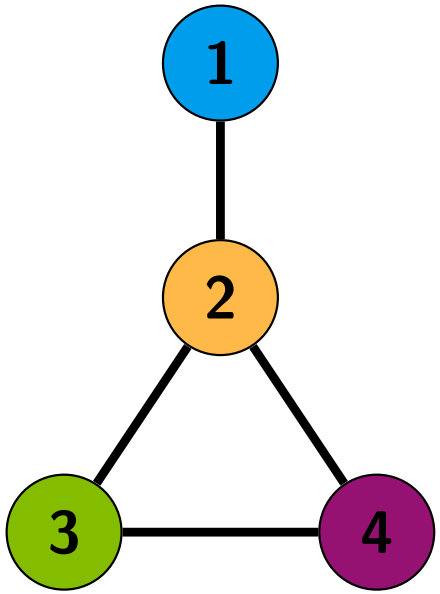
University of Glasgow

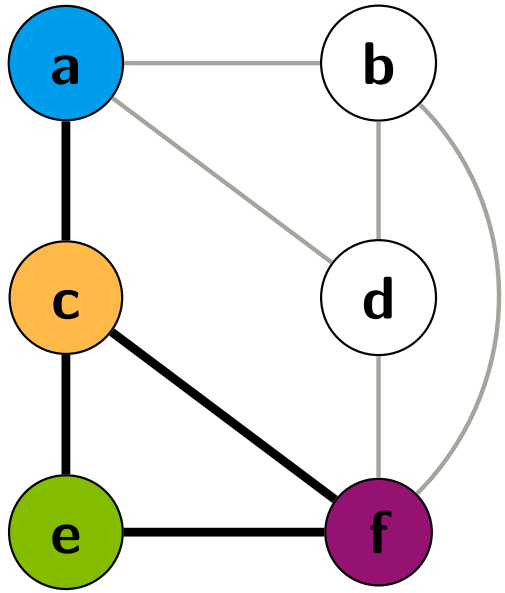
Finding Little Graphs Inside Big Graphs





Finding Little Graphs Inside Big Graphs

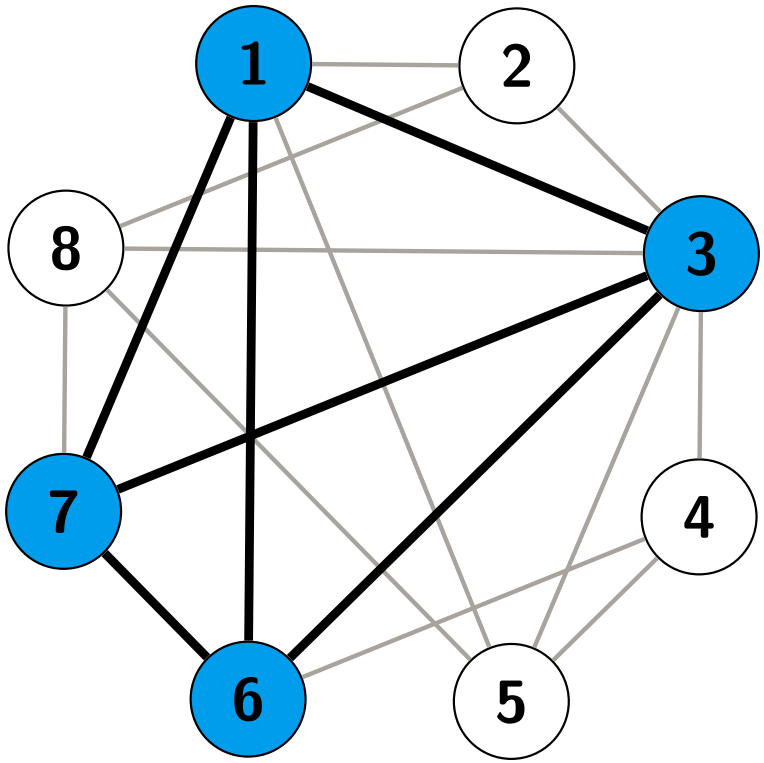




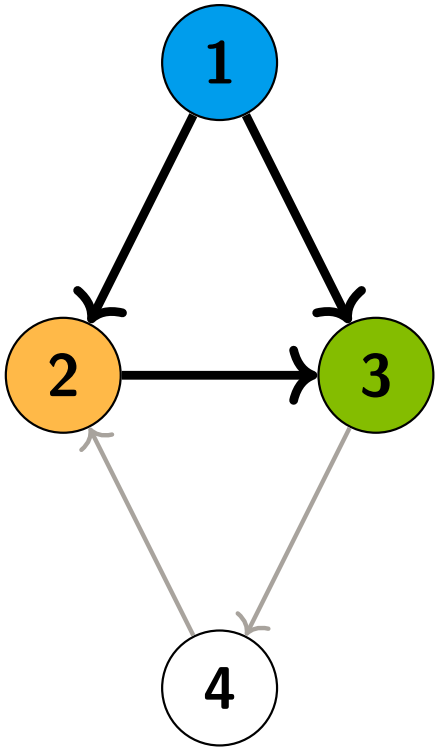
Subgraph Isomorphism

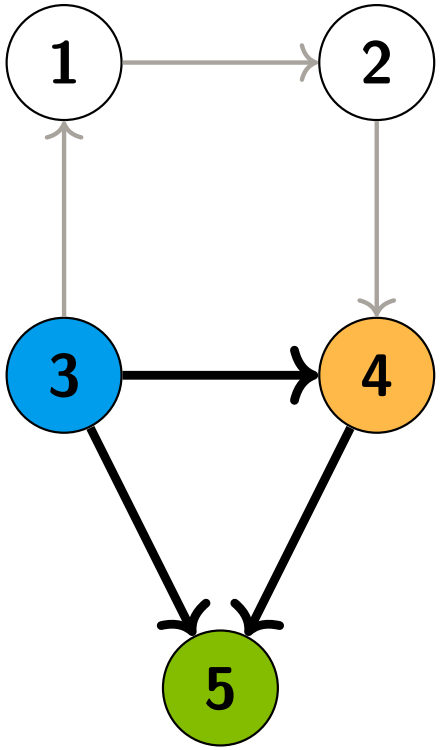
- Find an *injective* mapping from a *pattern* graph to a *target* graph.
- **Adjacent** vertices must be **mapped to adjacent** vertices.
- For the *induced* problem variant, non-adjacent vertices must be mapped to non-adjacent vertices.

Maximum Clique

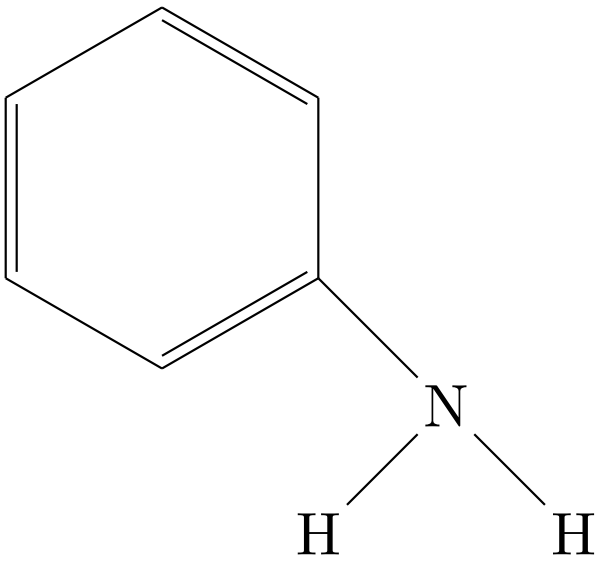


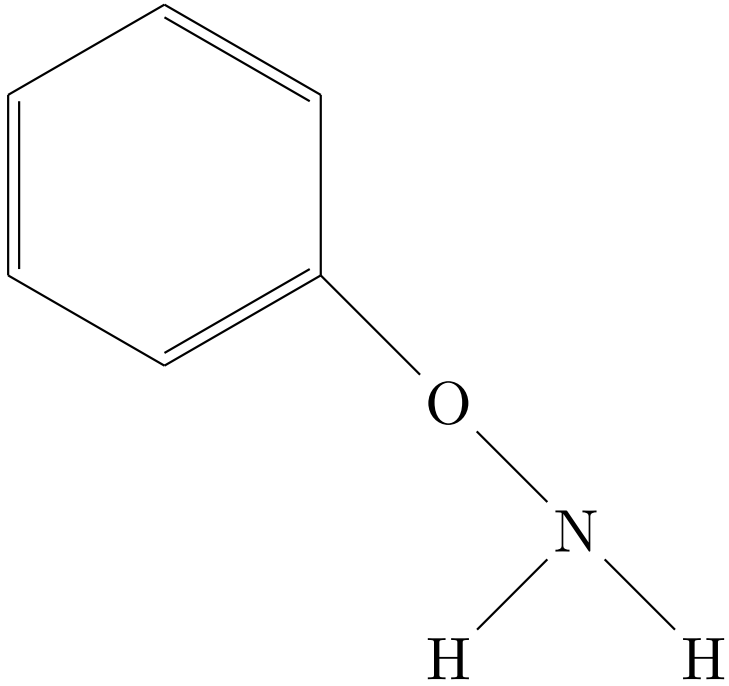
Maximum Common Subgraph





Maximum Common Connected Subgraph





Who Cares?

- Bioinformatics
- Chemistry
- Drug design
- Computer vision
- Pattern recognition
- Financial fraud detection
- Model checking
- Drone formation flying
- Fault detection
- Law enforcement
- Digital circuit design
- Kidney exchange
- Social network analysis
- Compilers
- Malware detection
- Computer algebra
- Network design
- Diseased cows

Why I Care

- A lovely area for empirical algorithmics.
  - Complicated enough for it to be interesting, but not too complicated that we can't understand it.
  - An active research area, but it's not overwhelming.
- Added bonus: parallel algorithms that aren't just about solving differential equations.
- What general lessons can we learn with a thousand core-years to play with?

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.

Backtracking Search

- Pick an unassigned variable (using a heuristic).
- For each value left in its domain (ordered using a heuristic):
  - Try giving the variable that value.
  - Infer some things.
  - If no variable is wiped out, recurse.

Branch and Bound

- Keep track of the best solution found so far (the **incumbent**).
- If a current partial solution can't unseat the incumbent, backtrack immediately.
  - We need a **bound** function.
  - For clique: number of vertices accepted, plus number of vertices not yet rejected.
  - Better: greedily colour the unrejected vertices.
  - We'll look at a clique algorithm in detail after the break.

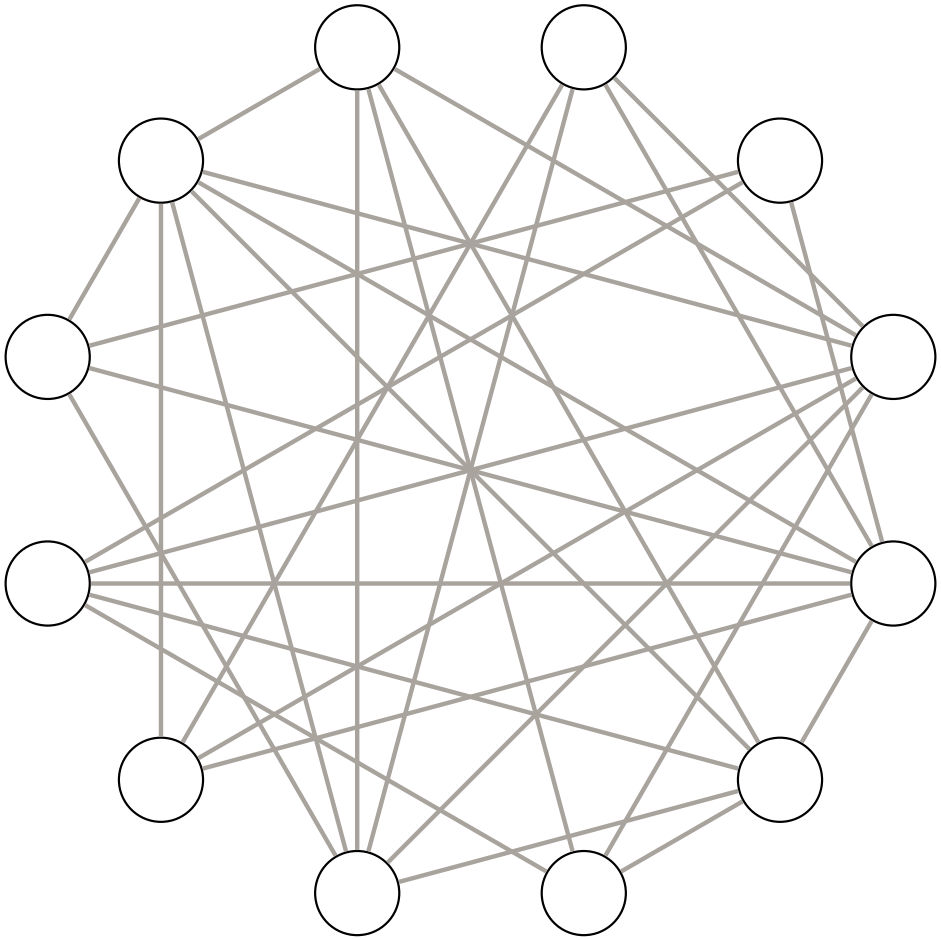
What Can We Solve?

- Maximum clique:
  - 250 vertices no matter what.
  - 1,000 to 10,000 vertices is usually OK.
  - 1,000,000 vertices in really sparse graphs.
- Subgraph isomorphism:
  - 1,000 / 10,000 vertices, if you believe the standard benchmark suites are fair.
  - 20 / 150 vertices, if you don't.
- Maximum common (connected) subgraph:
  - 35 / 35 vertices.
  - 100 / 100 vertices if there are enough labels.
  - 1,000 / 10,000 unlabelled if the solution is almost the entire smaller graph.

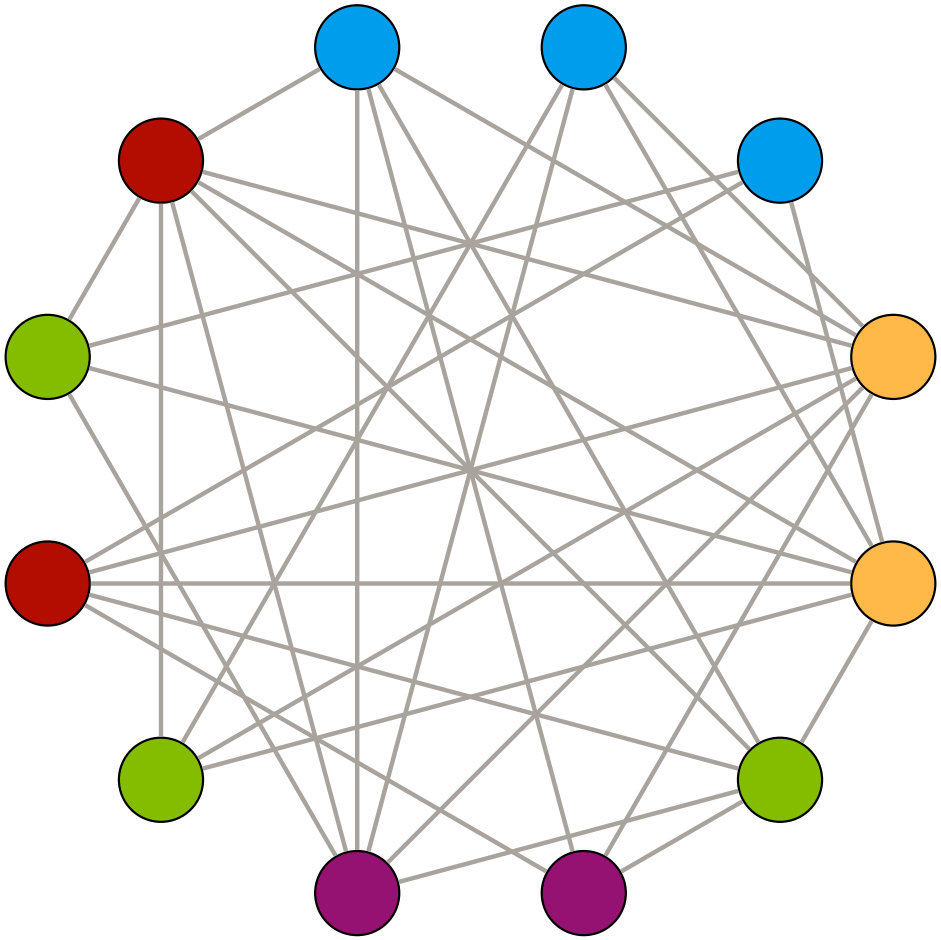
What Can We Solve?

- Maximum clique:
  - 250 vertices no matter what.
  - 1,000 to 10,000 vertices is usually OK.
  - 1,000,000 vertices in really sparse graphs.
- Subgraph isomorphism:
  - 1,000 / 10,000 vertices, if you believe the standard benchmark suites are fair.
  - 20 / 150 vertices, if you don't.
- Maximum common (connected) subgraph:
  - 40 / 40 vertices.
  - 100 / 100 vertices if there are enough labels.
  - 1,000 / 10,000 unlabelled if the solution is almost the entire smaller graph.

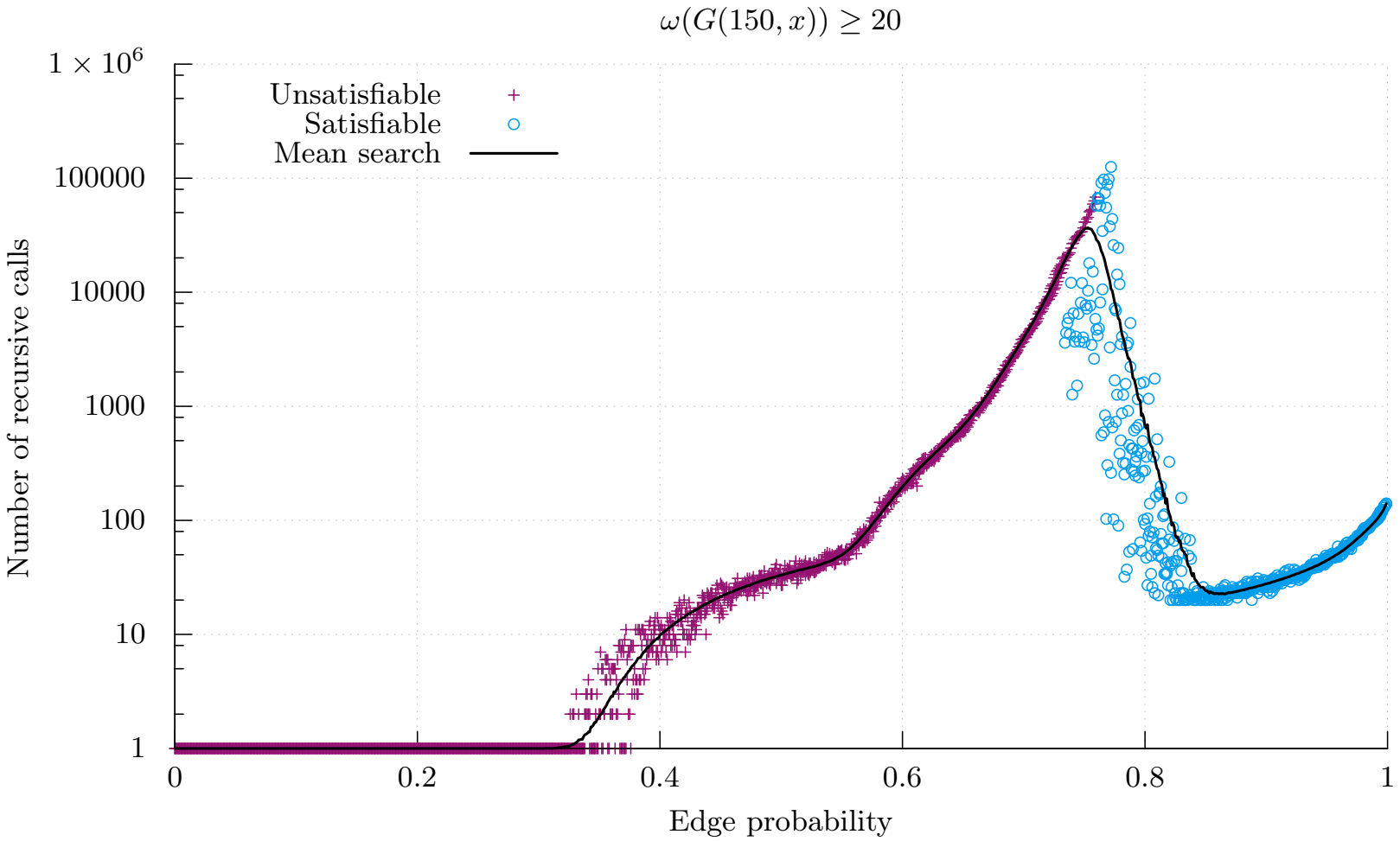
When is Clique Really Hard?



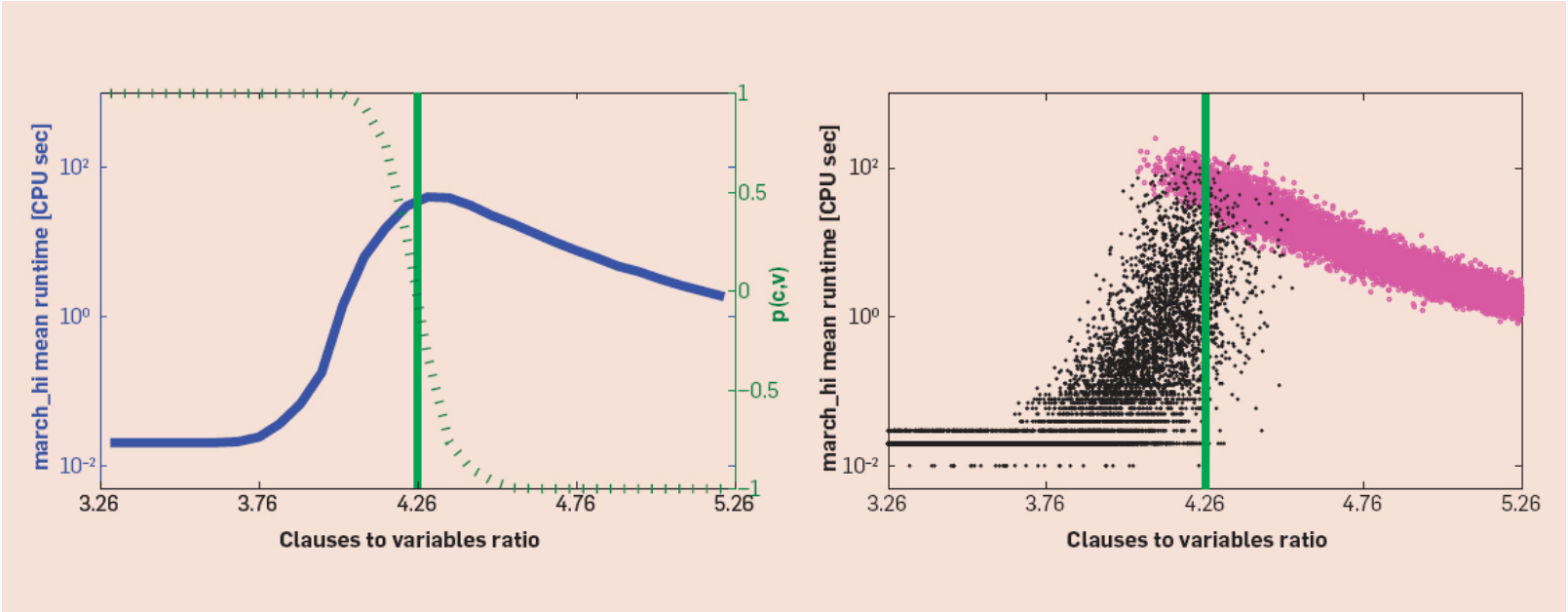
When is Clique Really Hard?



Phase Transitions

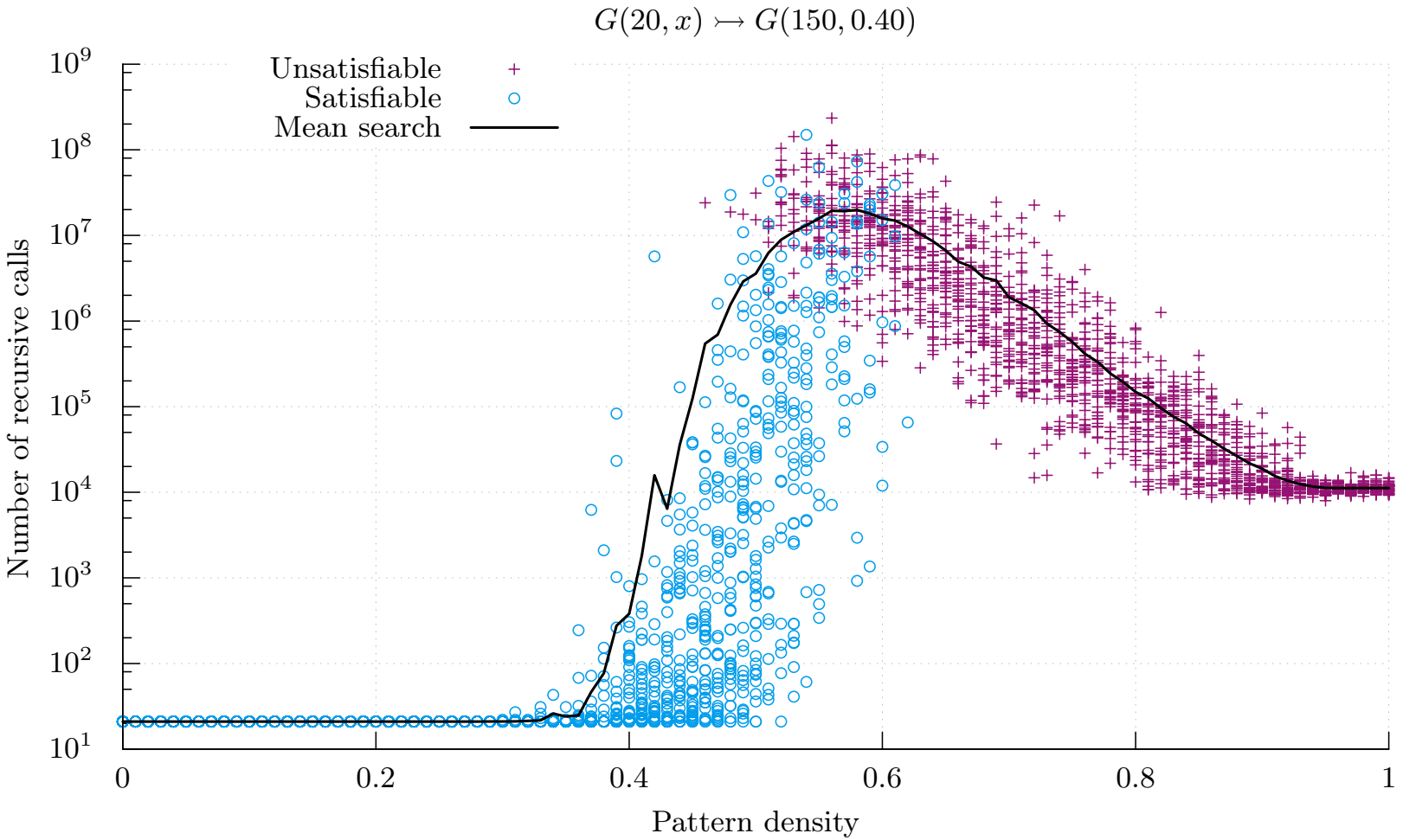


Phase Transitions

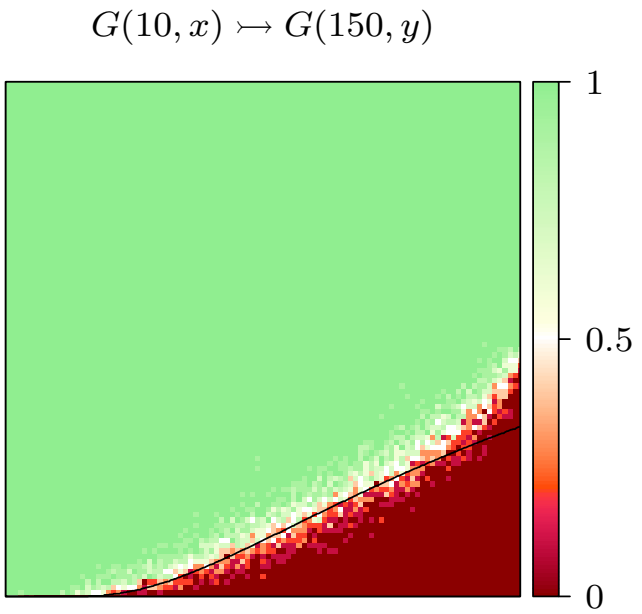


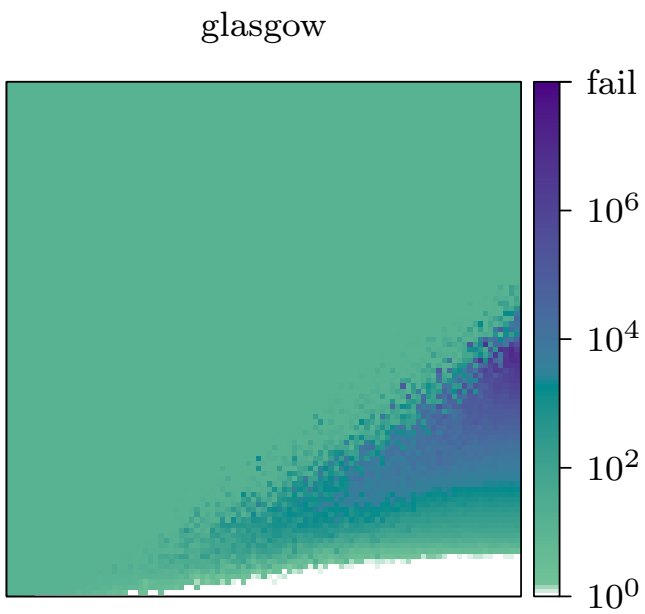
Understanding the Empirical Hardness of NP-Complete Problems.
Kevin Leyton-Brown, Holger H. Hoos, Frank Hutter, Lin Xu.
Communications of the ACM, Vol. 57 No. 5, Pages 98–107.

What About Non-Induced Subgraph Isomorphism?

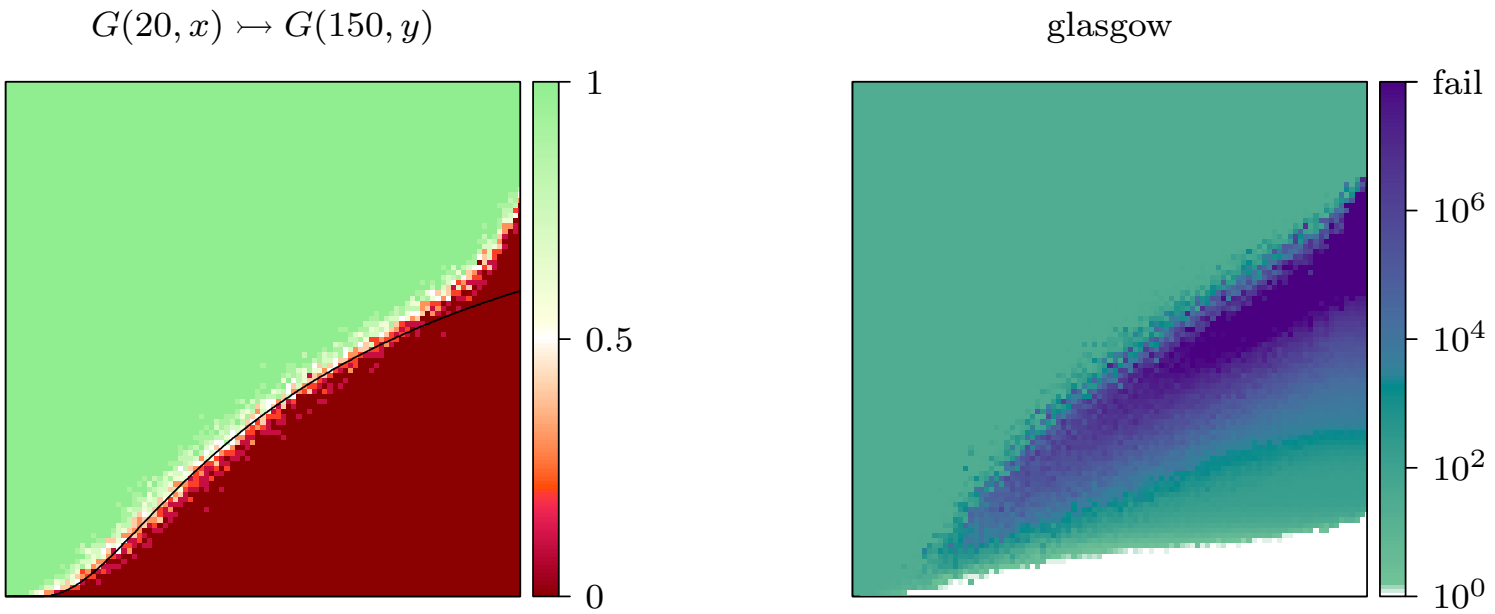


What About Non-Induced Subgraph Isomorphism?

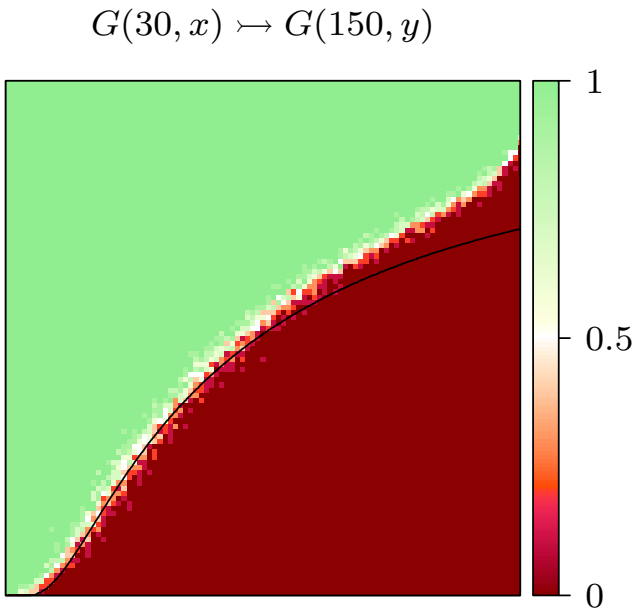


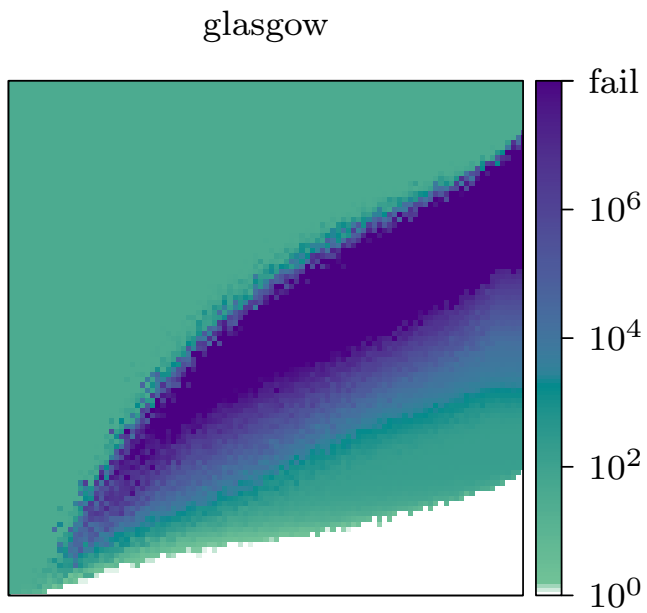


What About Non-Induced Subgraph Isomorphism?

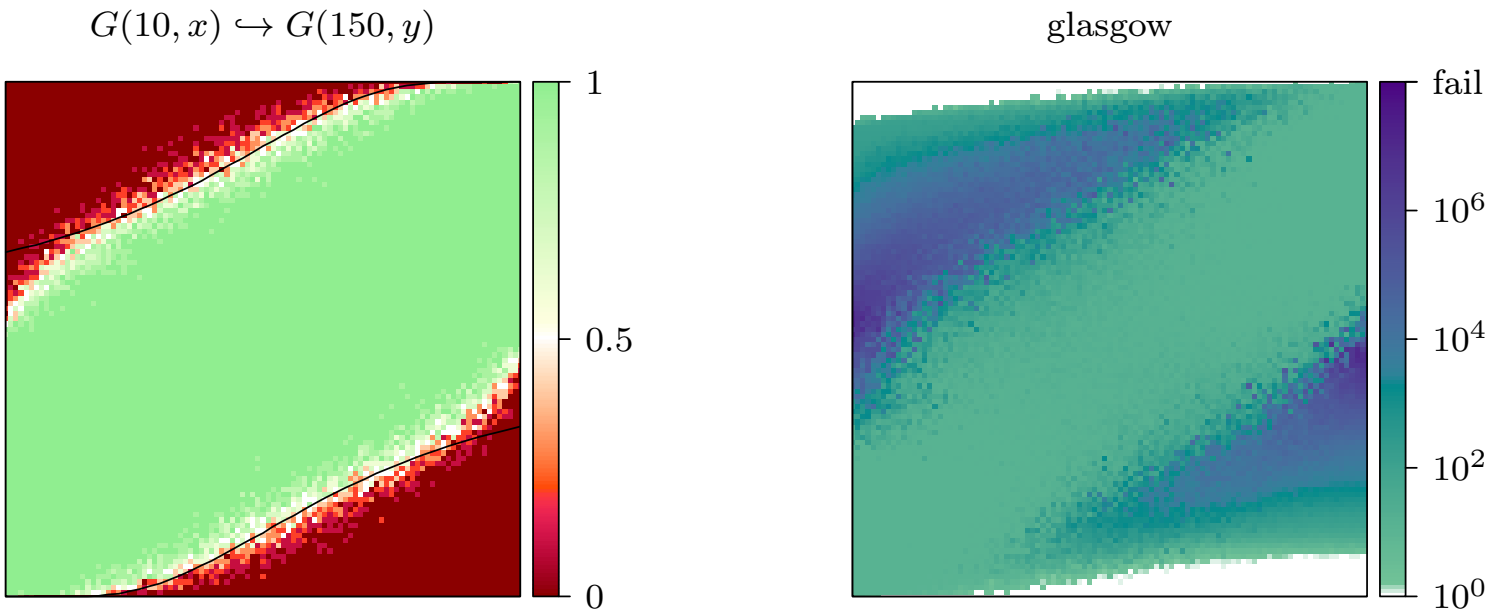


What About Non-Induced Subgraph Isomorphism?

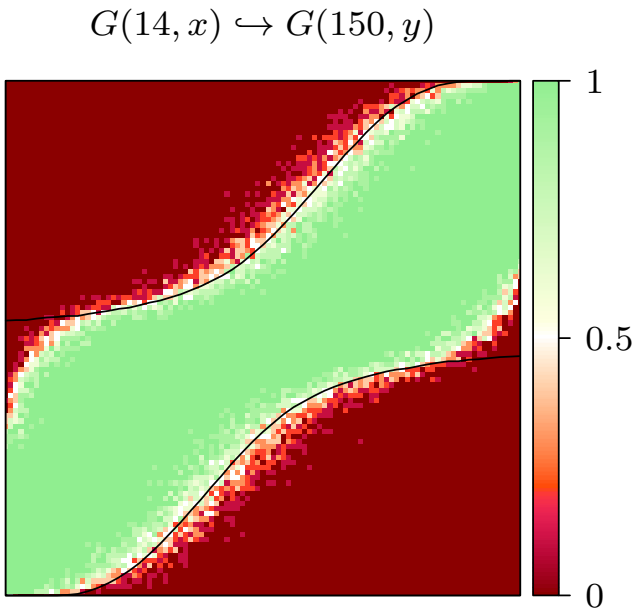


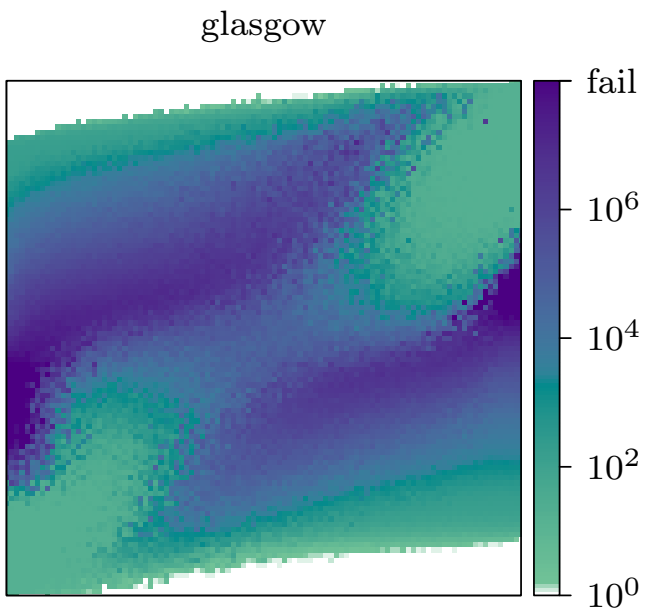


What About Induced Subgraph Isomorphism?

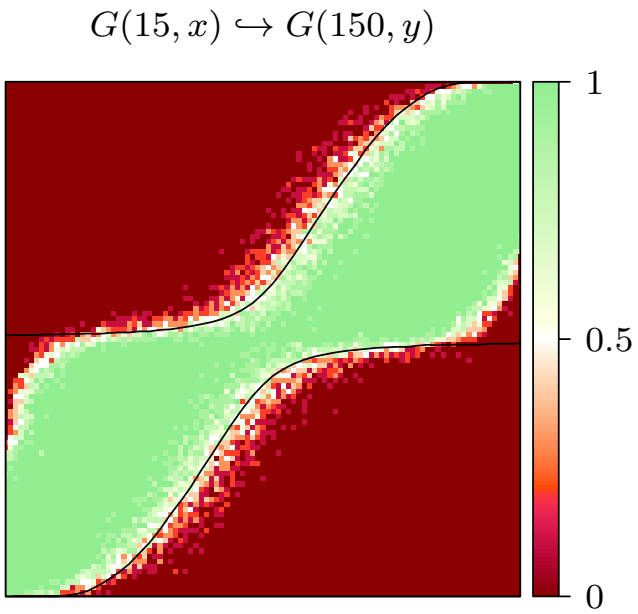


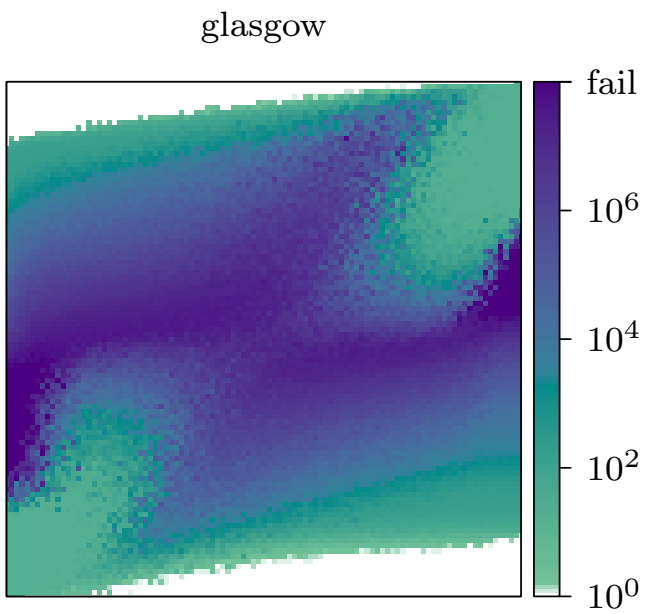
What About Induced Subgraph Isomorphism?



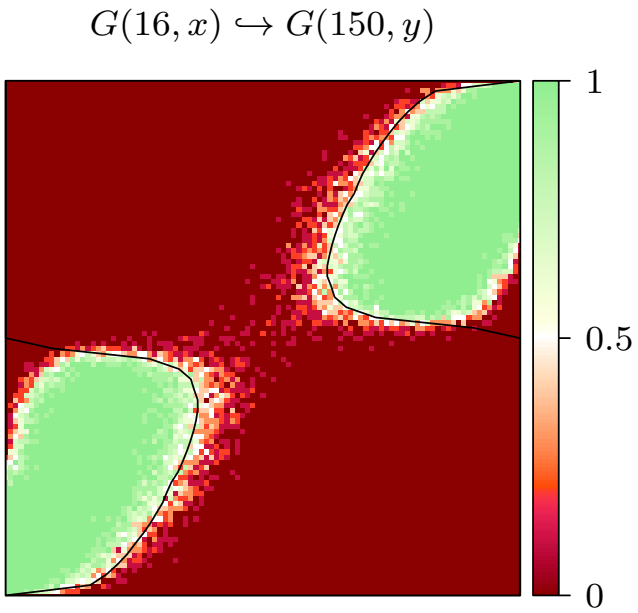


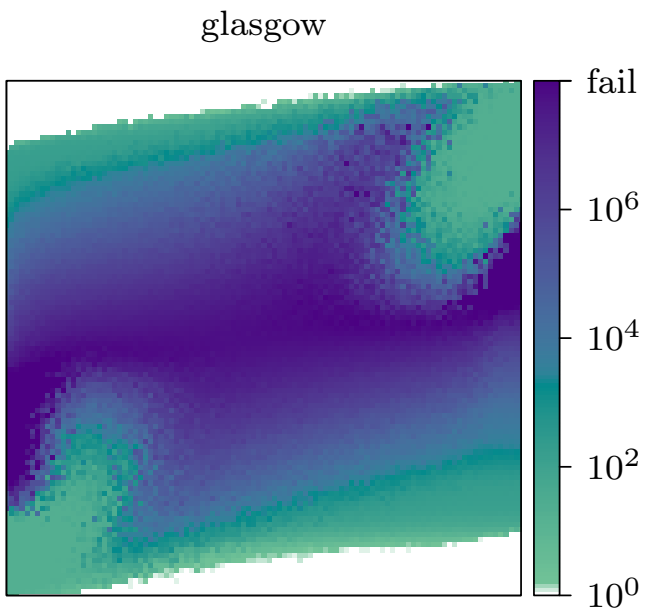
What About Induced Subgraph Isomorphism?



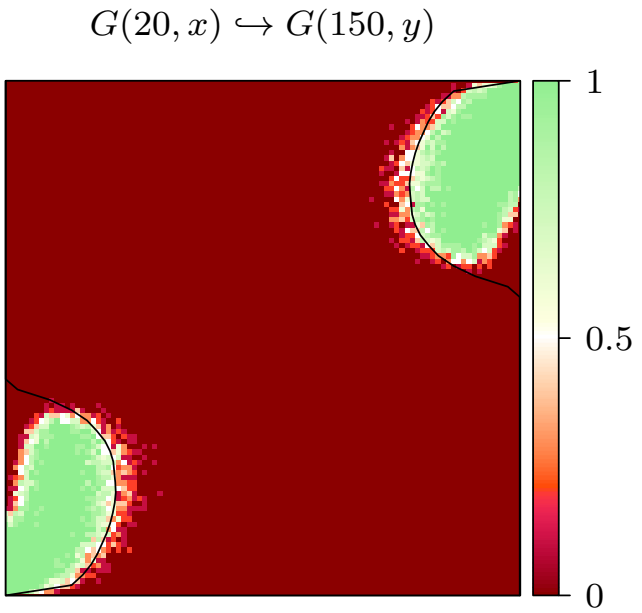


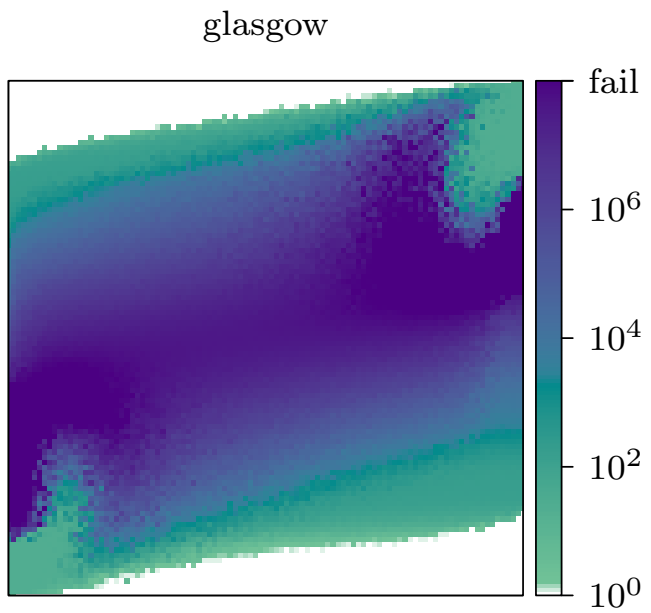
What About Induced Subgraph Isomorphism?



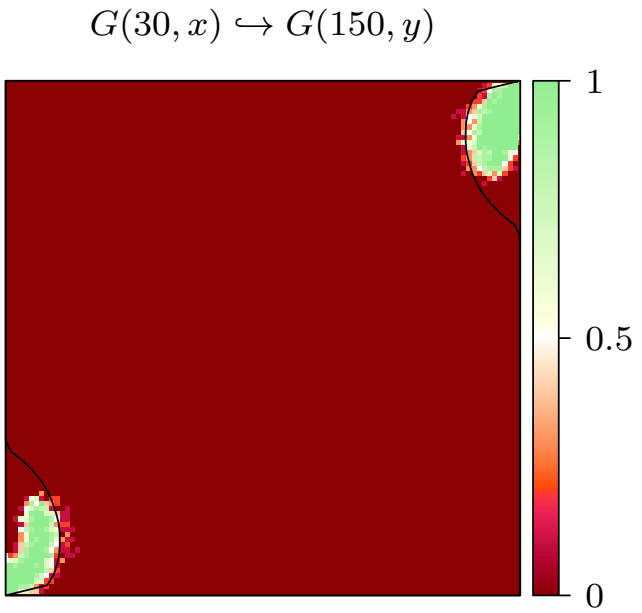


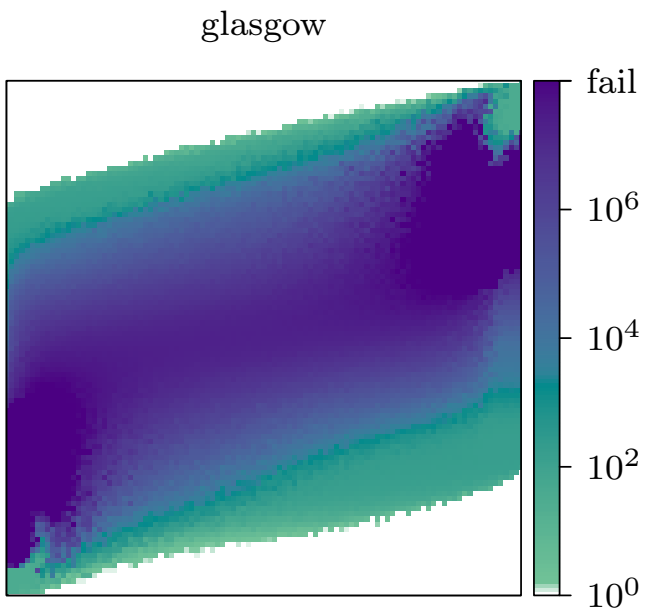
What About Induced Subgraph Isomorphism?





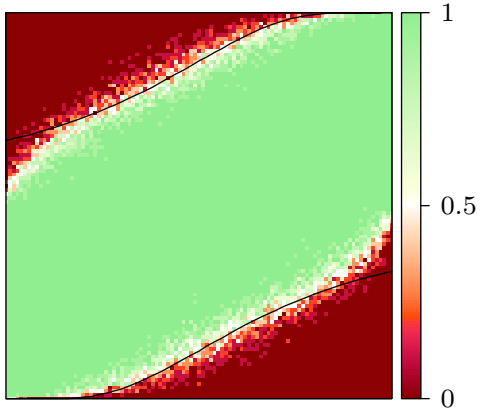
What About Induced Subgraph Isomorphism?



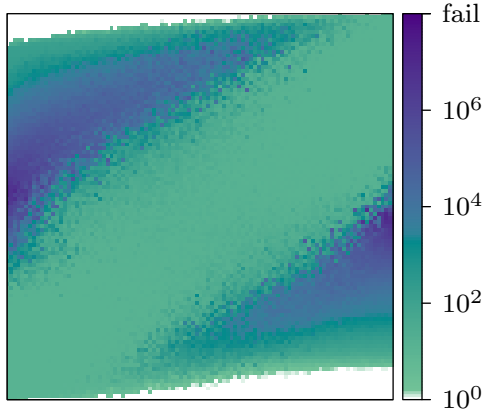


Is Difficulty Algorithm-Independent?

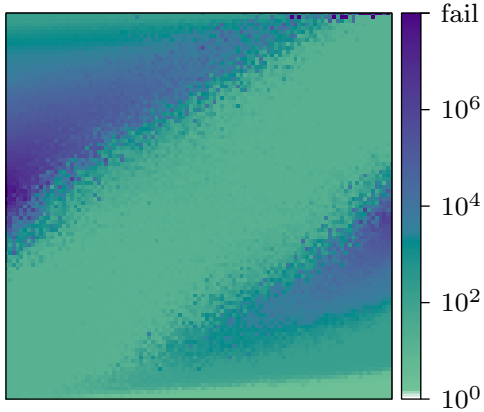
$G(10, x) \hookrightarrow G(150, y)$



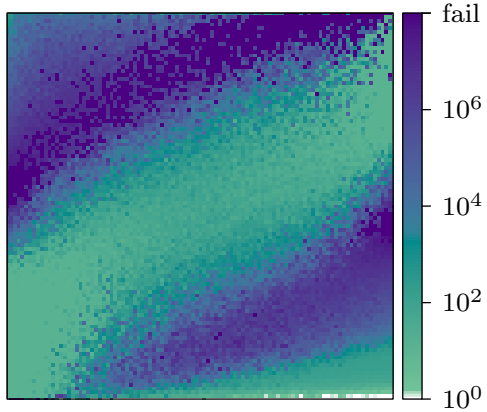
glasgow



lad

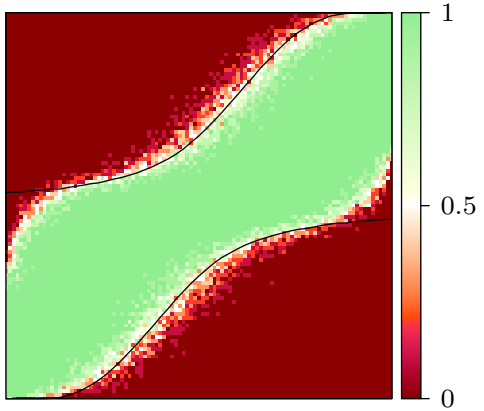


vf2

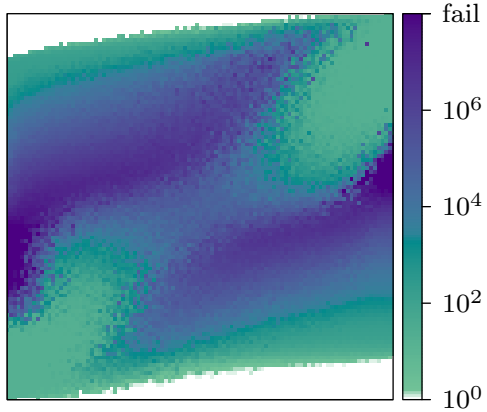


Is Difficulty Algorithm-Independent?

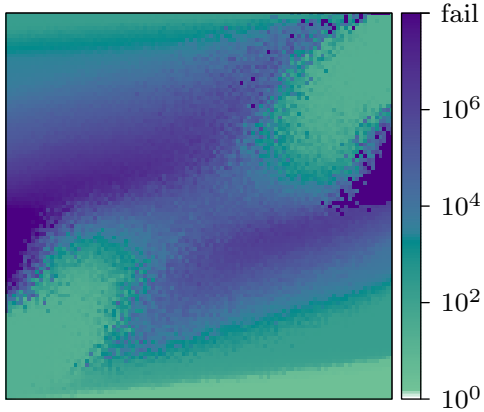
$G(14, x) \hookrightarrow G(150, y)$



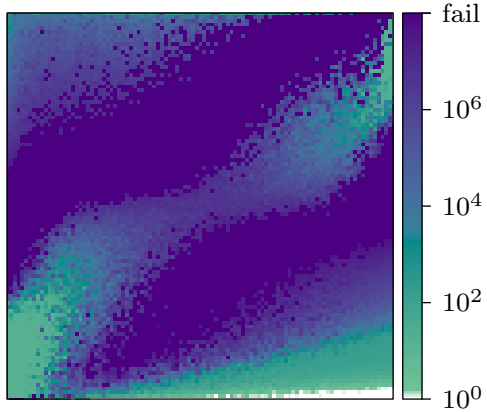
glasgow



lad

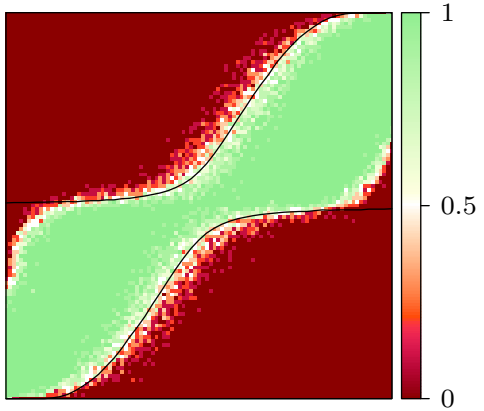


vf2

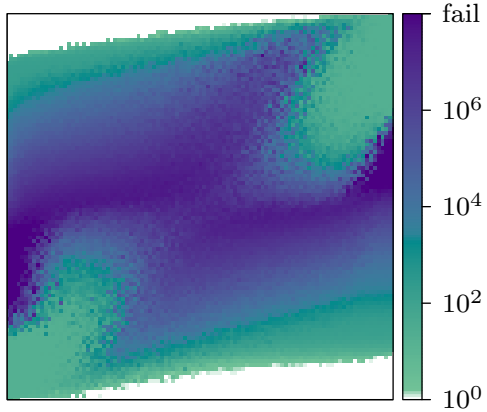


Is Difficulty Algorithm-Independent?

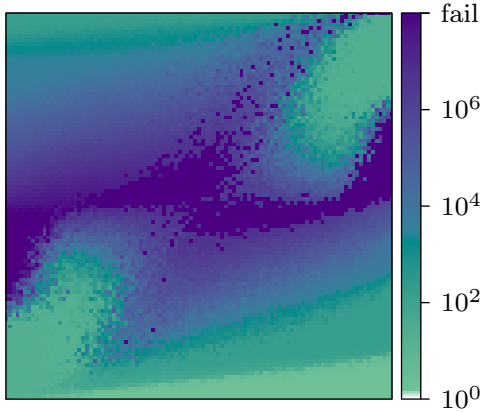
$G(15, x) \hookrightarrow G(150, y)$



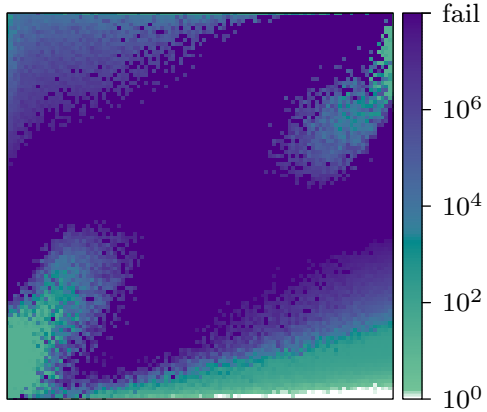
glasgow



lad

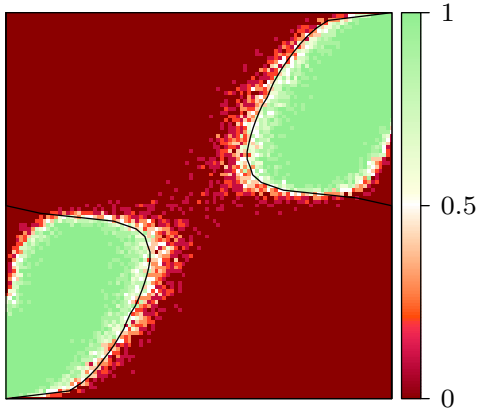


vf2

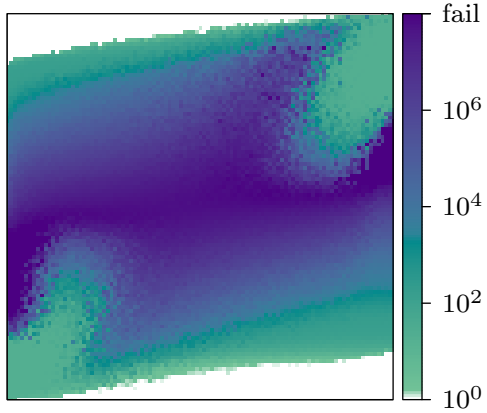


Is Difficulty Algorithm-Independent?

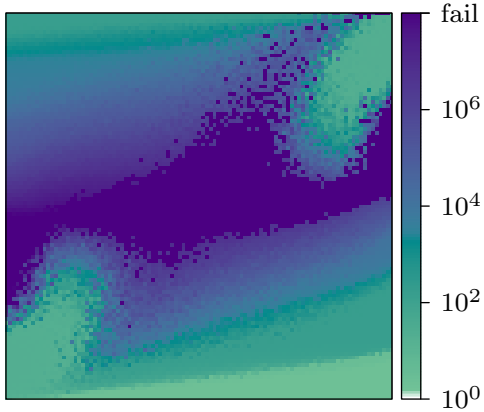
$G(16, x) \hookrightarrow G(150, y)$



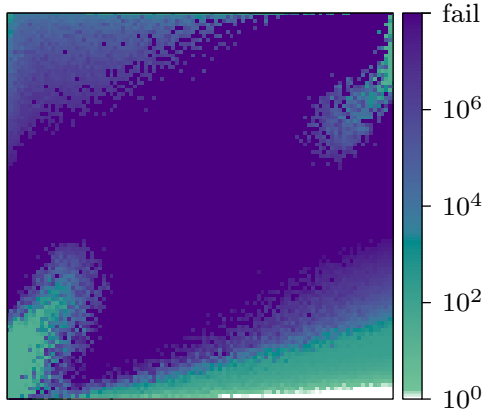
glasgow



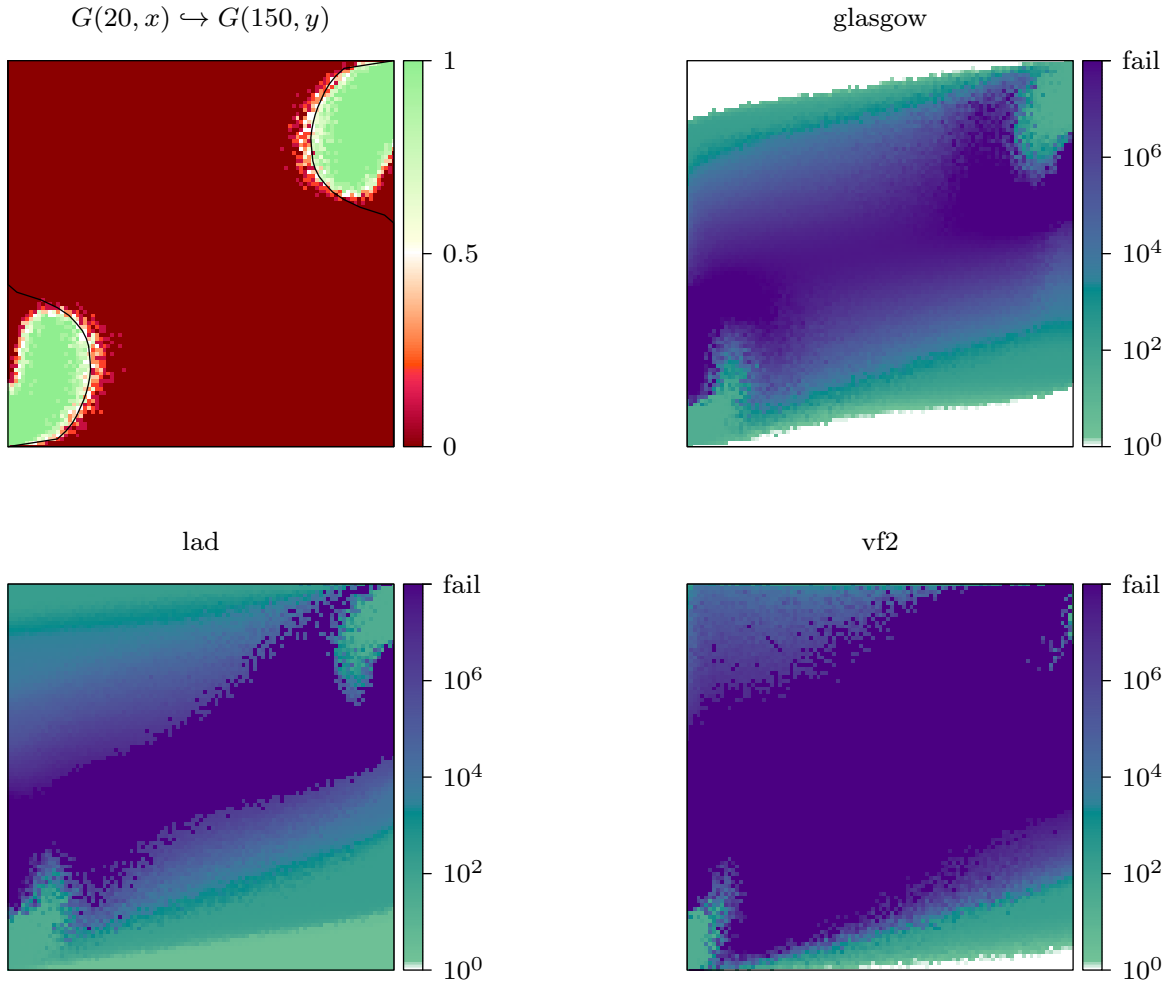
lad



vf2

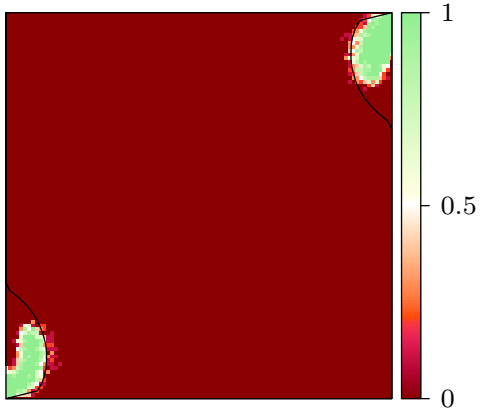


Is Difficulty Algorithm-Independent?

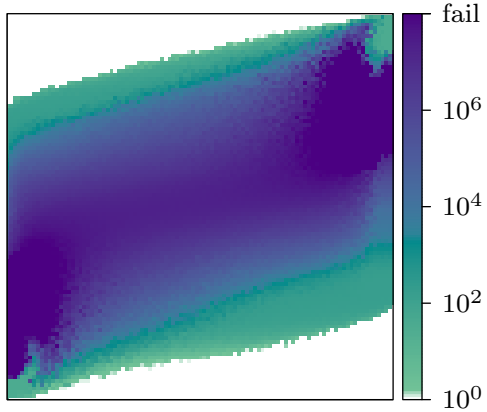


Is Difficulty Algorithm-Independent?

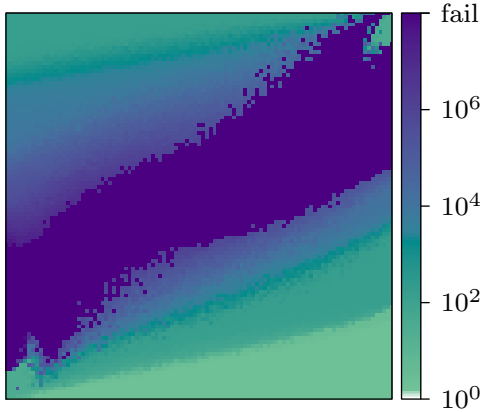
$G(30, x) \hookrightarrow G(150, y)$



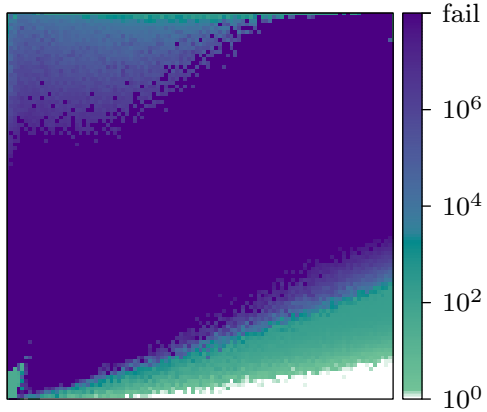
glasgow



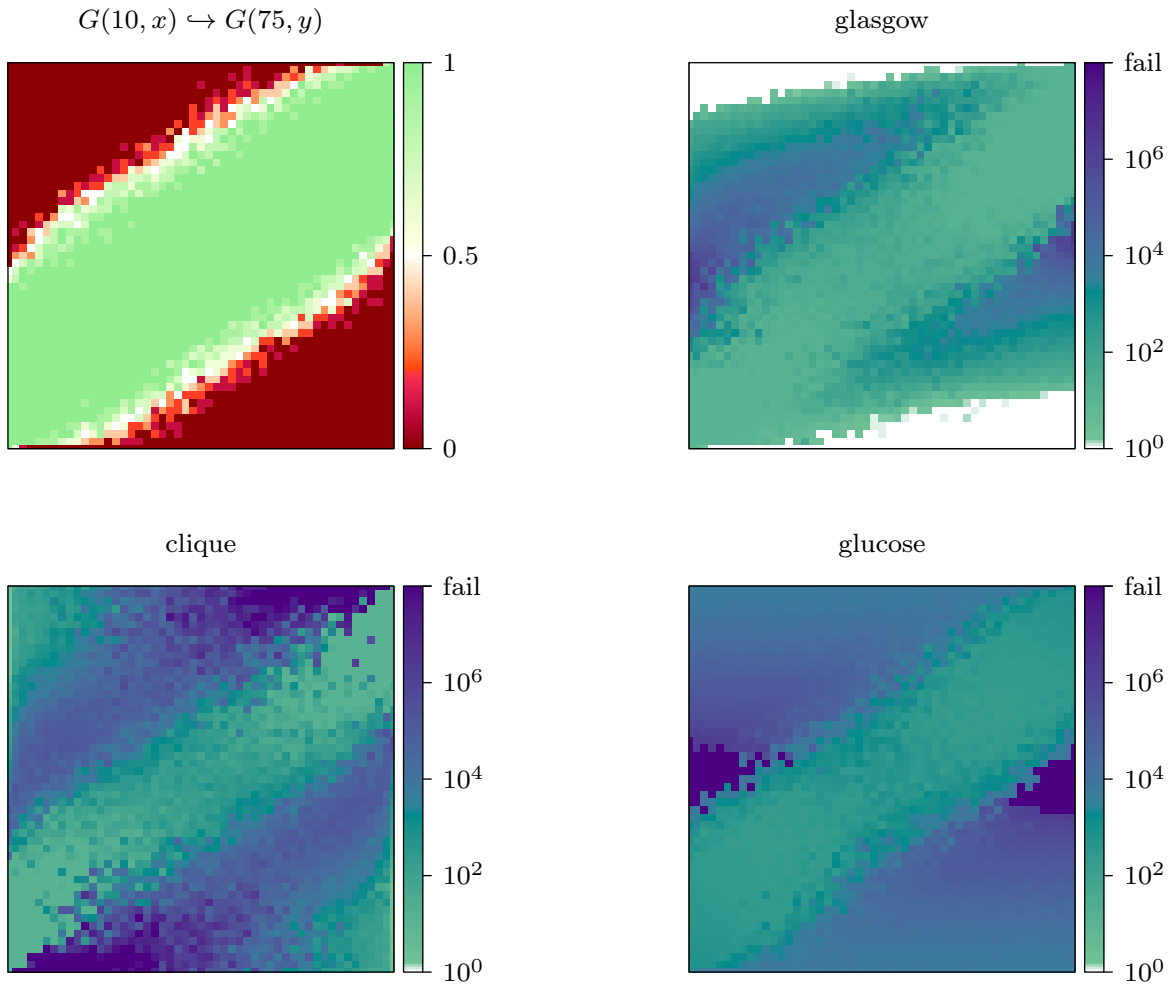
lad



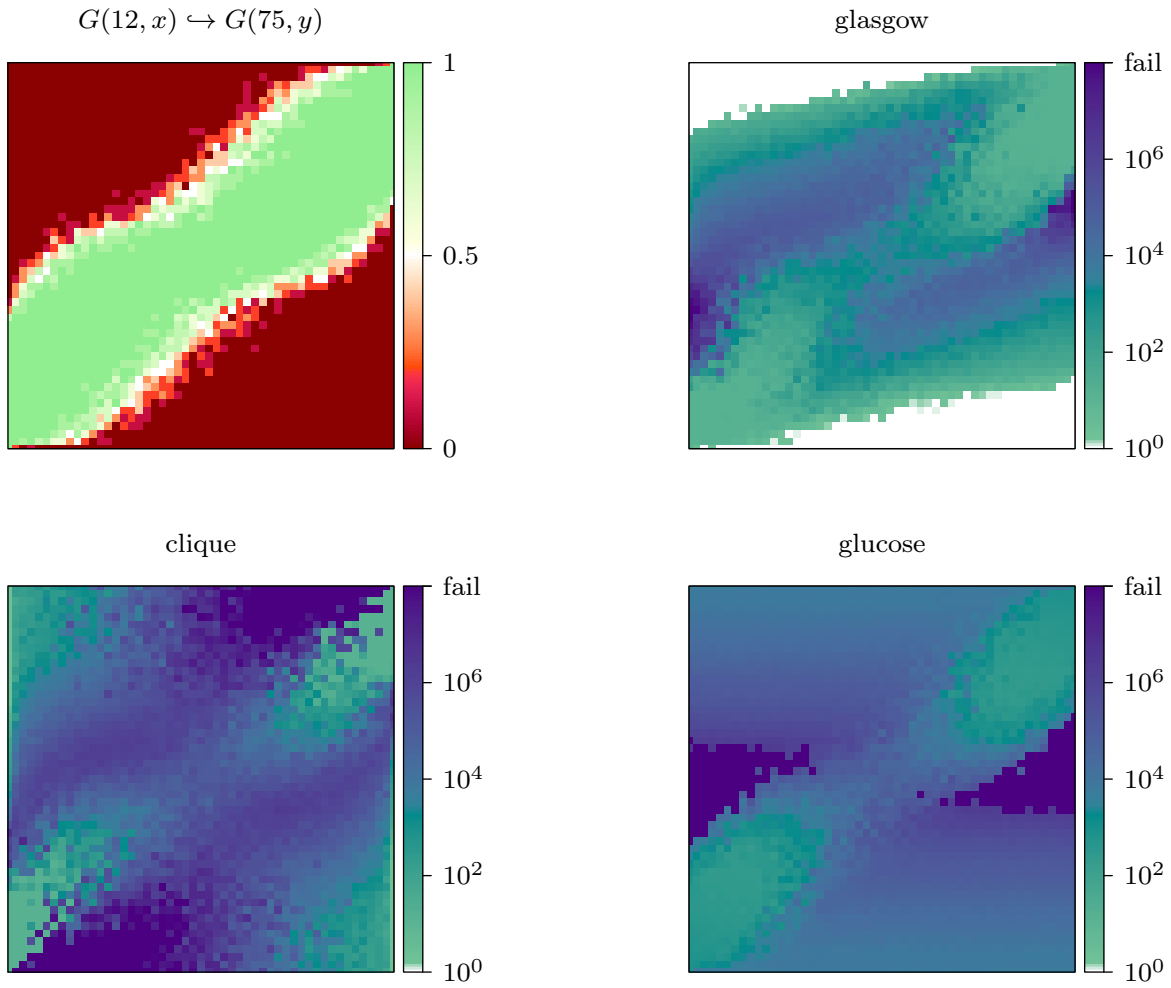
vf2



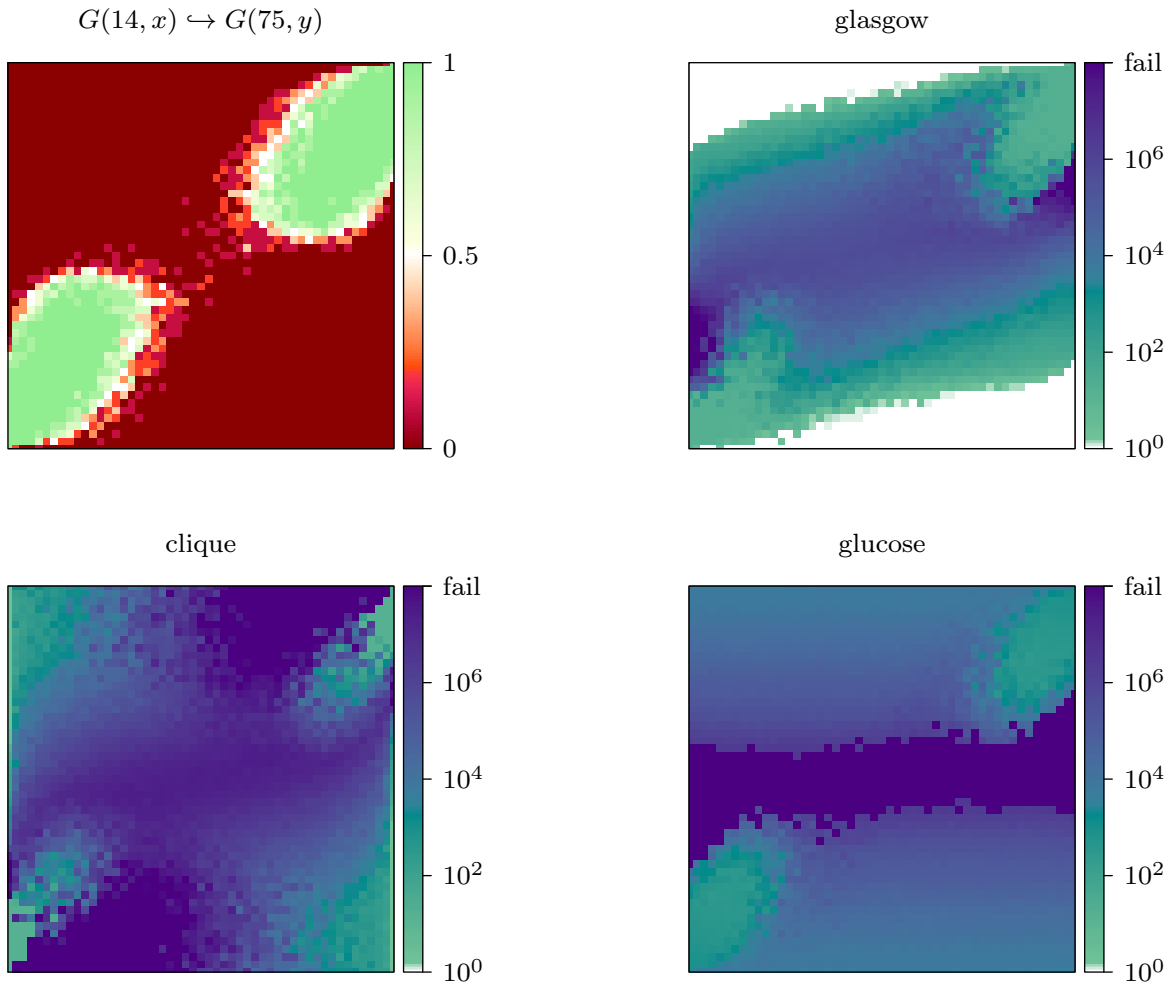
Is Difficulty Encoding-Independent?



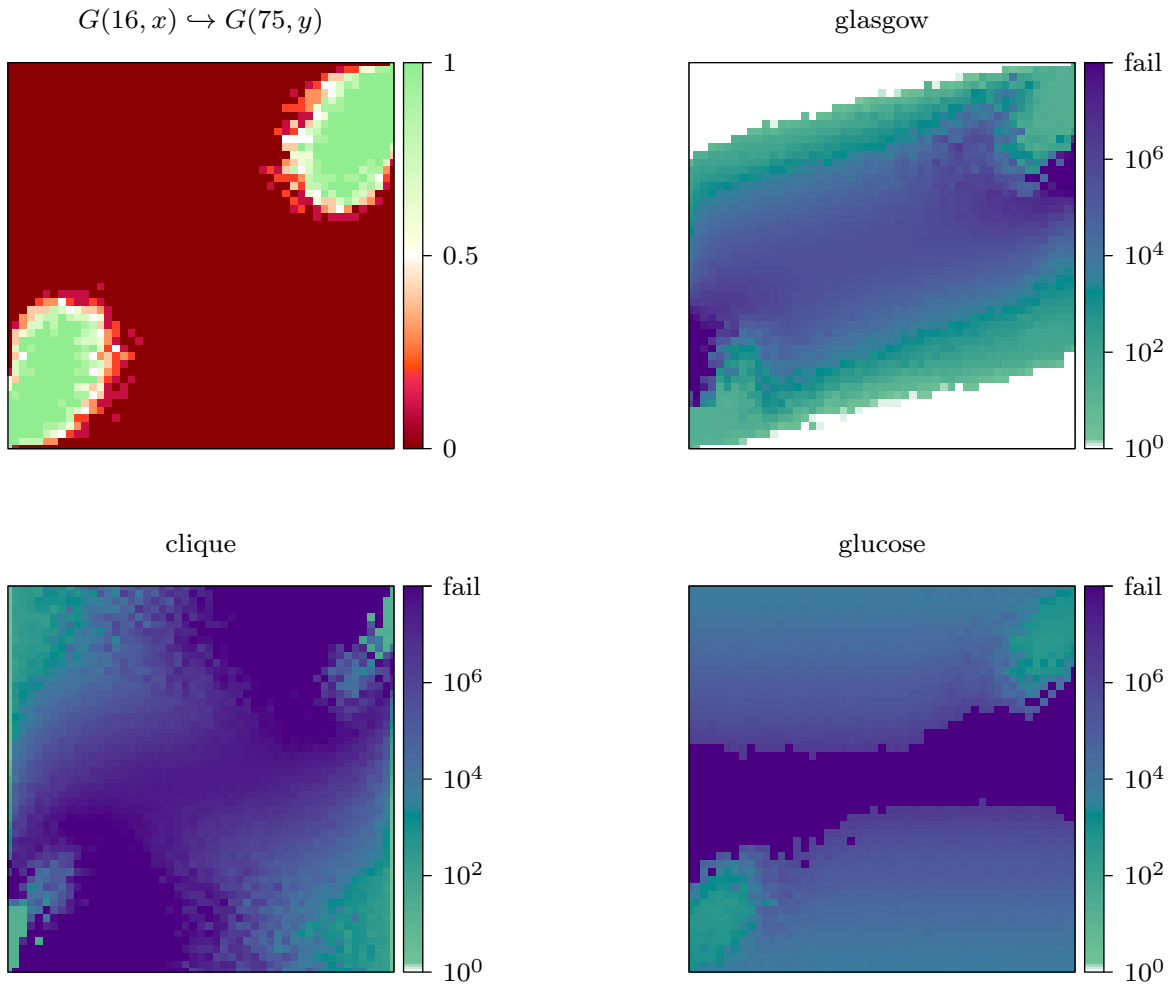
Is Difficulty Encoding-Independent?



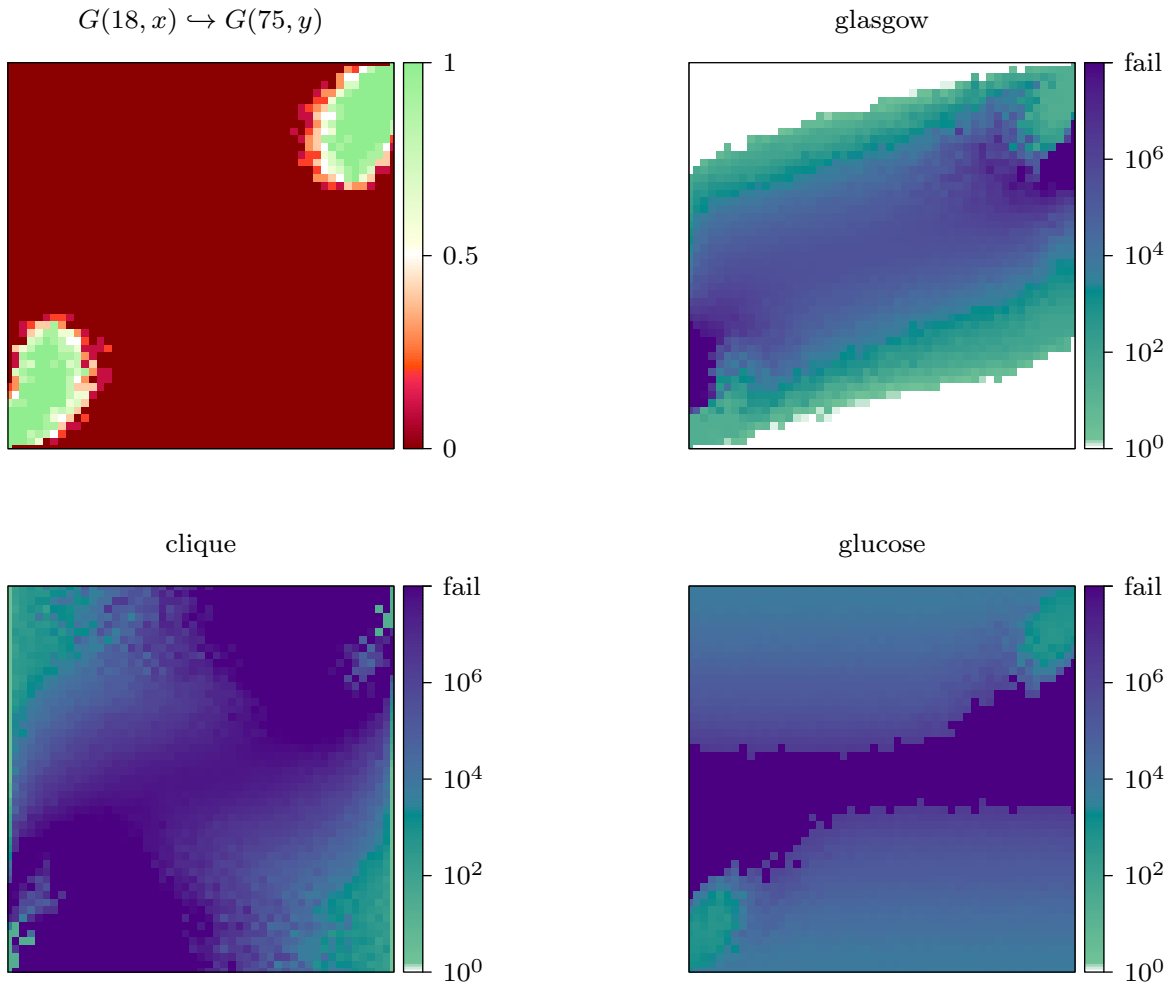
Is Difficulty Encoding-Independent?



Is Difficulty Encoding-Independent?

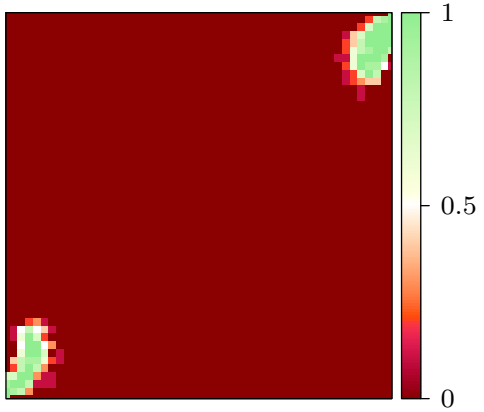


Is Difficulty Encoding-Independent?

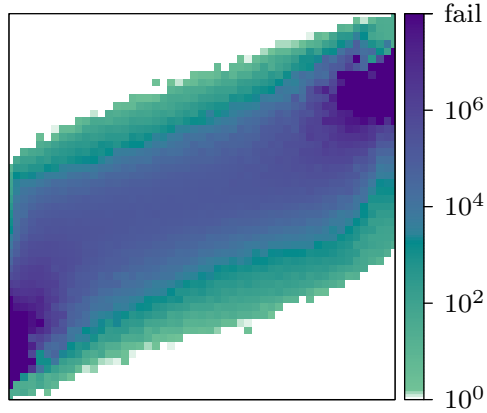


Is Difficulty Encoding-Independent?

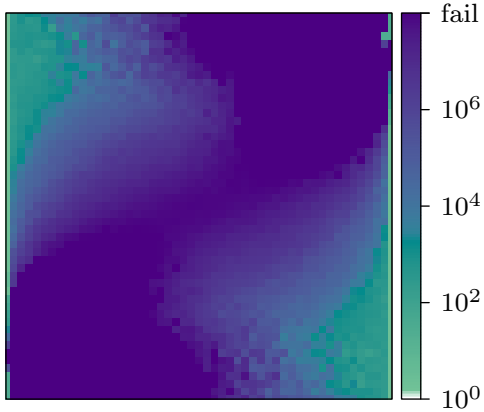
$G(25, x) \hookrightarrow G(75, y)$



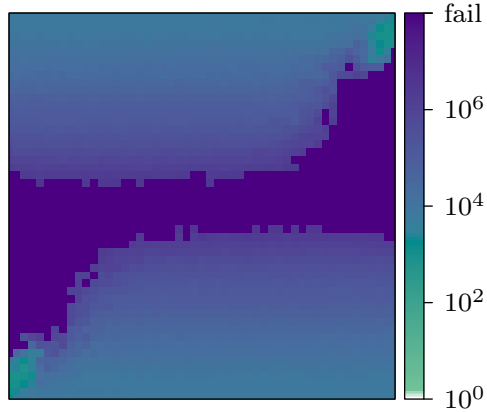
glasgow



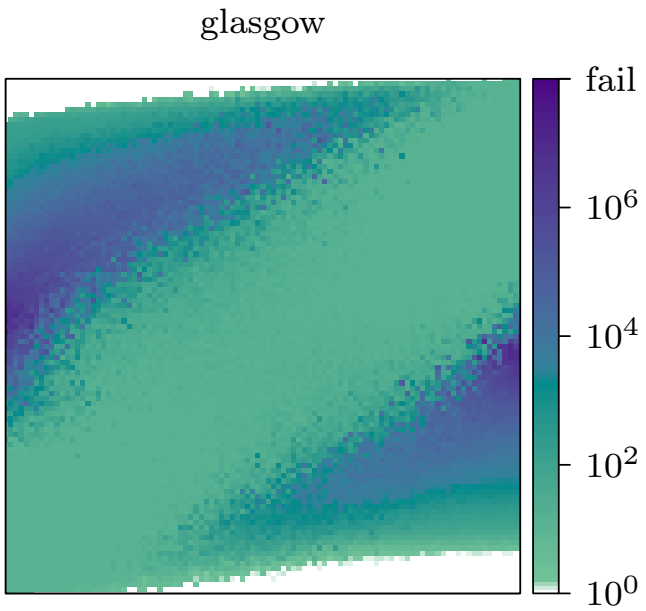
clique

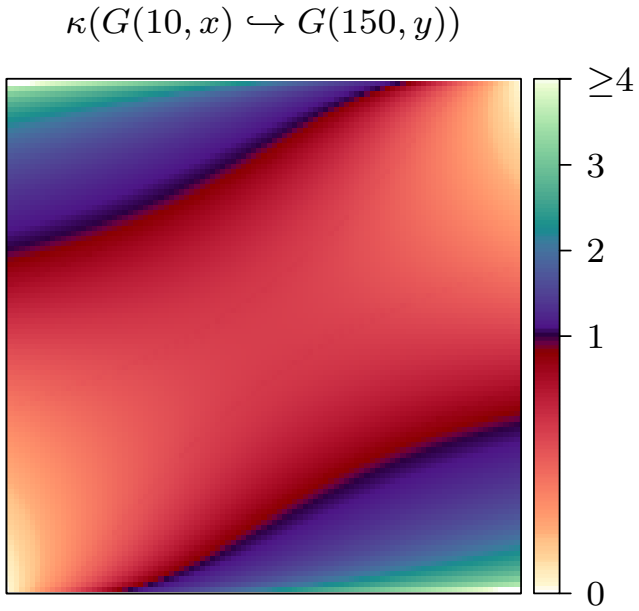


glucose



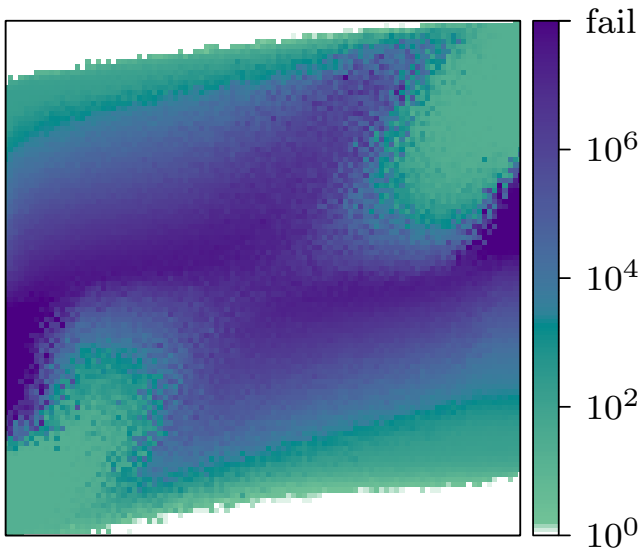
Constrainedness



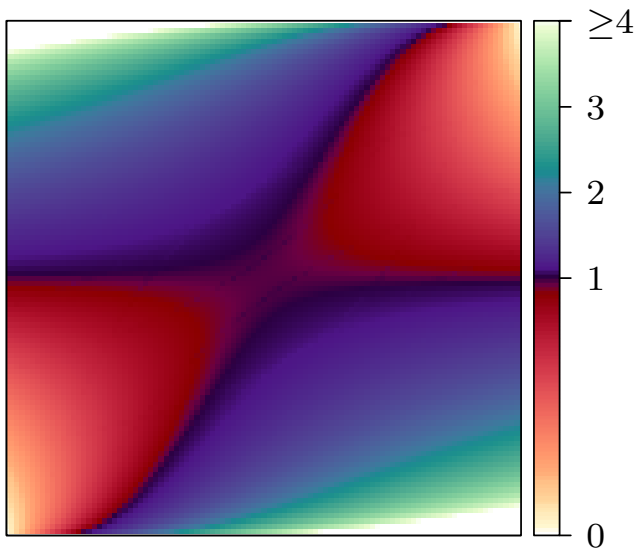


Constrainedness

glasgow

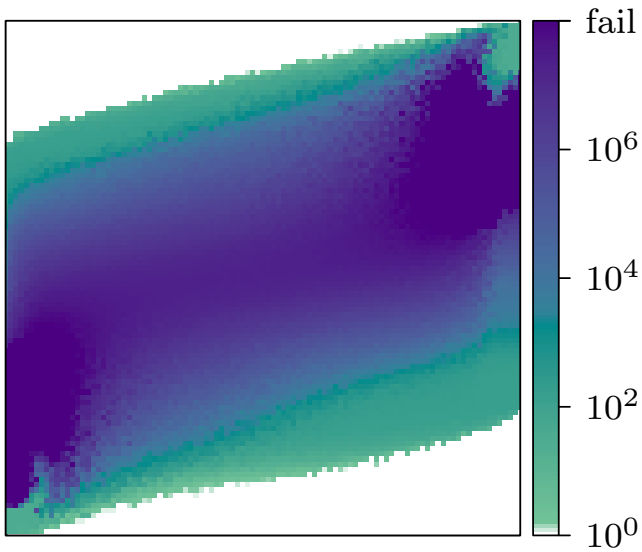


$\kappa(G(15, x) \hookrightarrow G(150, y))$

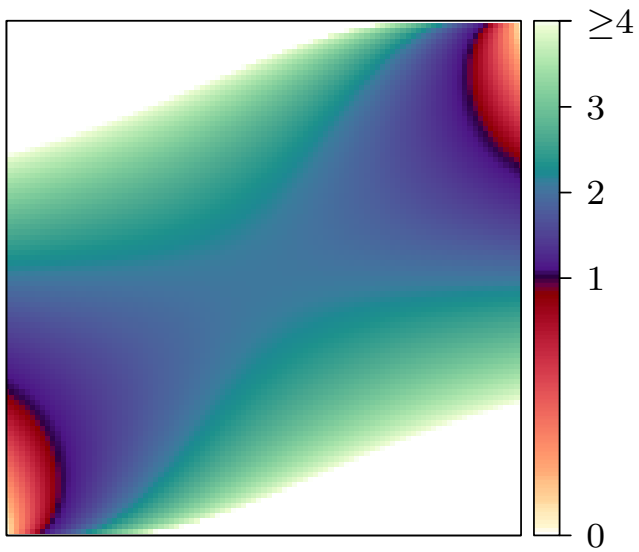


Constrainedness

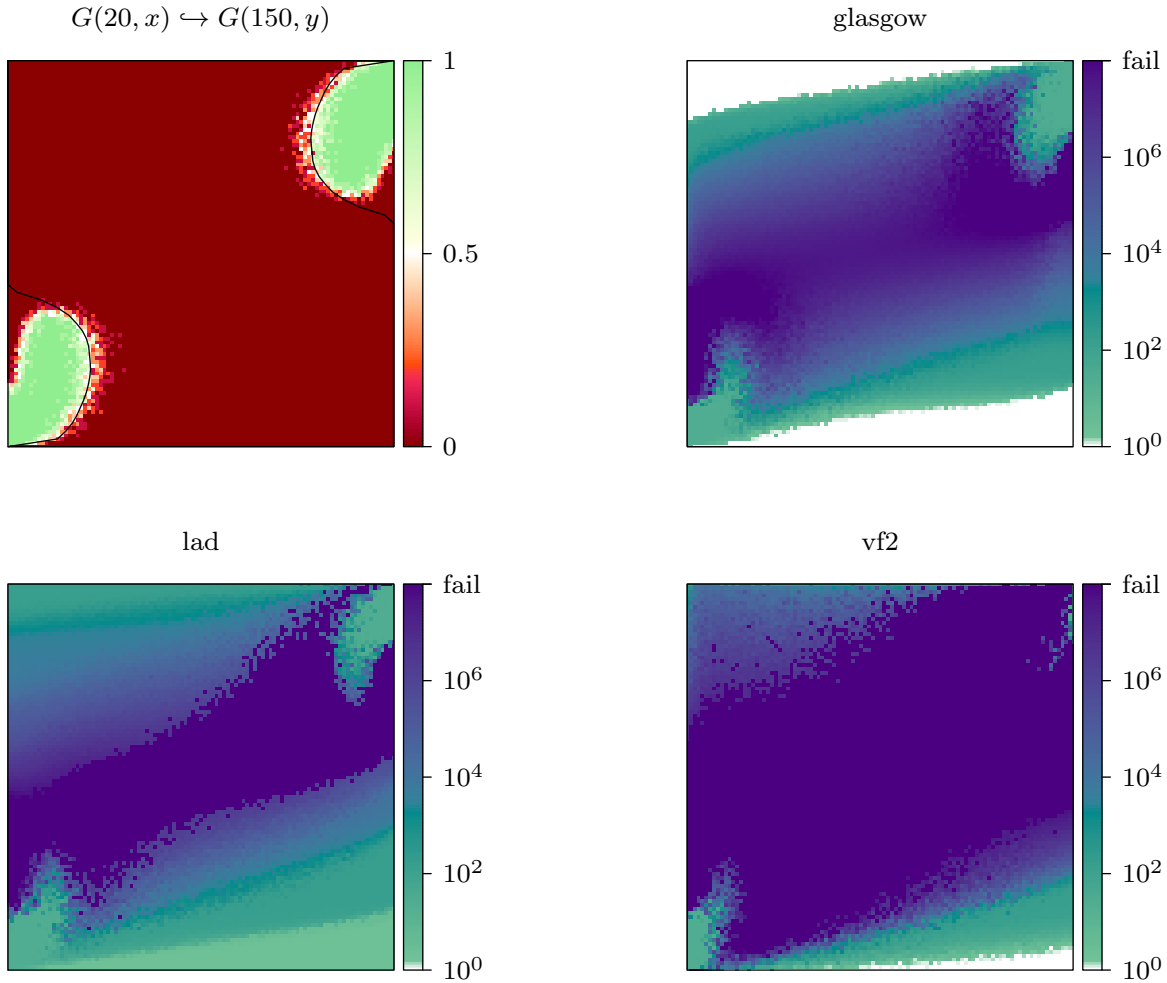
glasgow



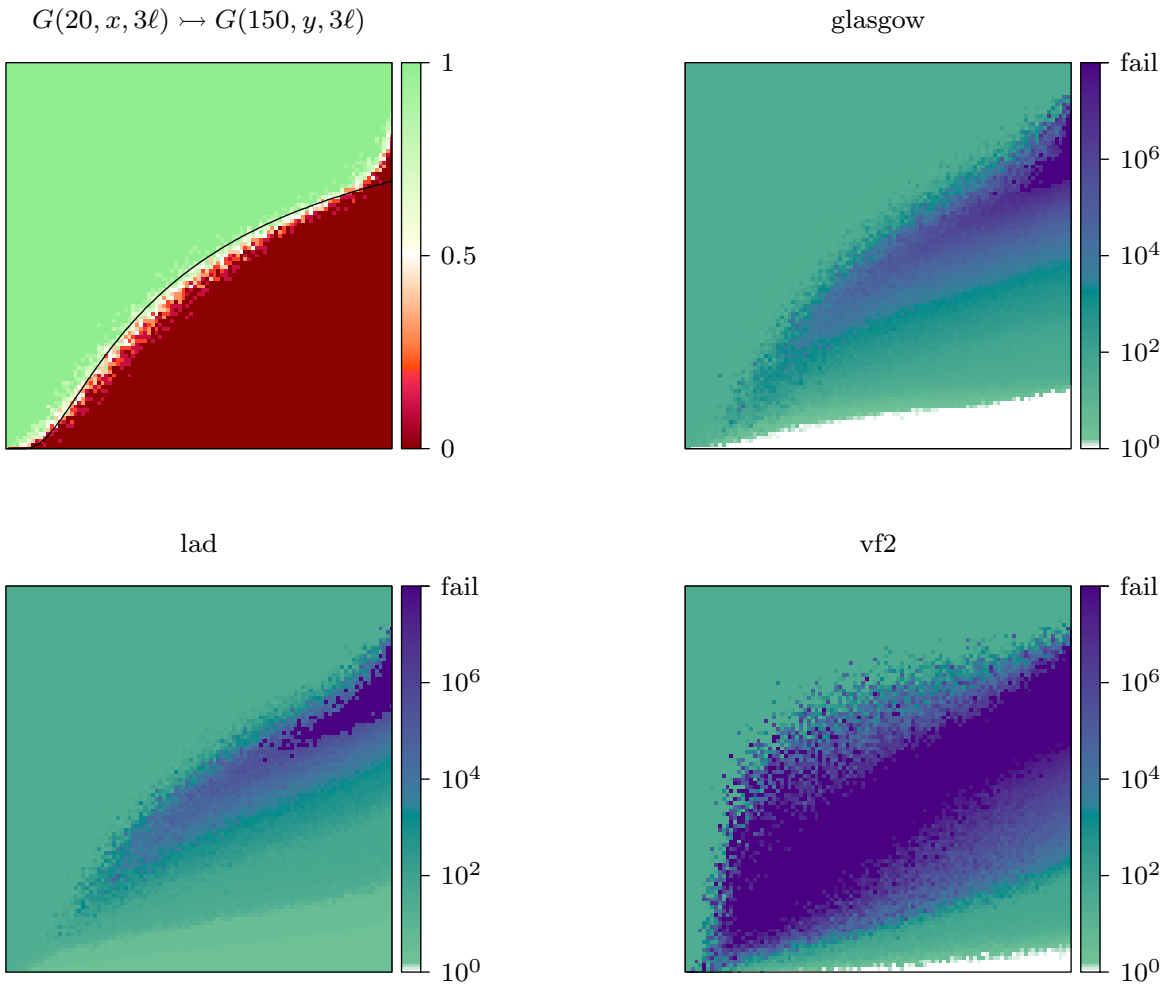
$\kappa(G(30, x) \hookrightarrow G(150, y))$



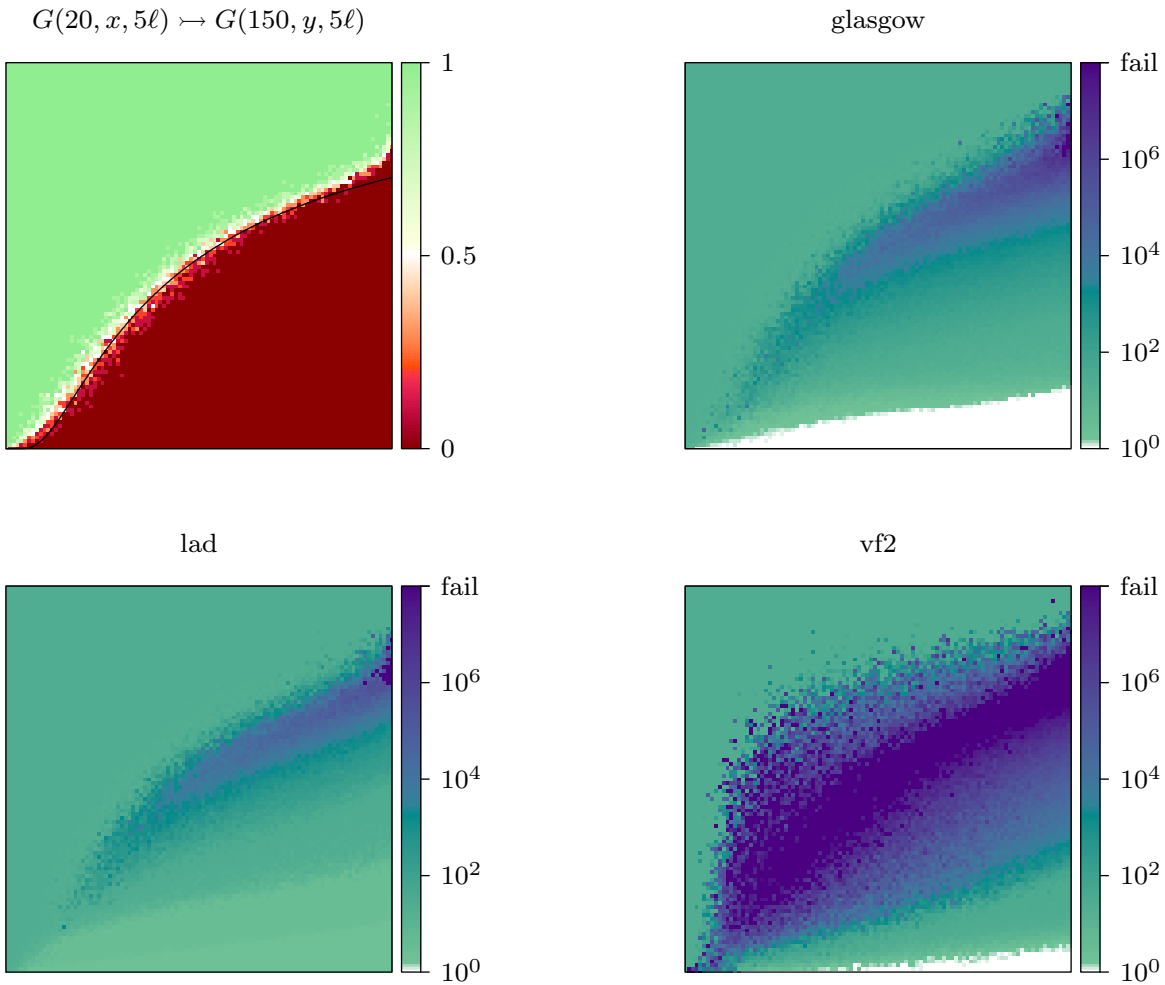
Why I Don't Like VF2



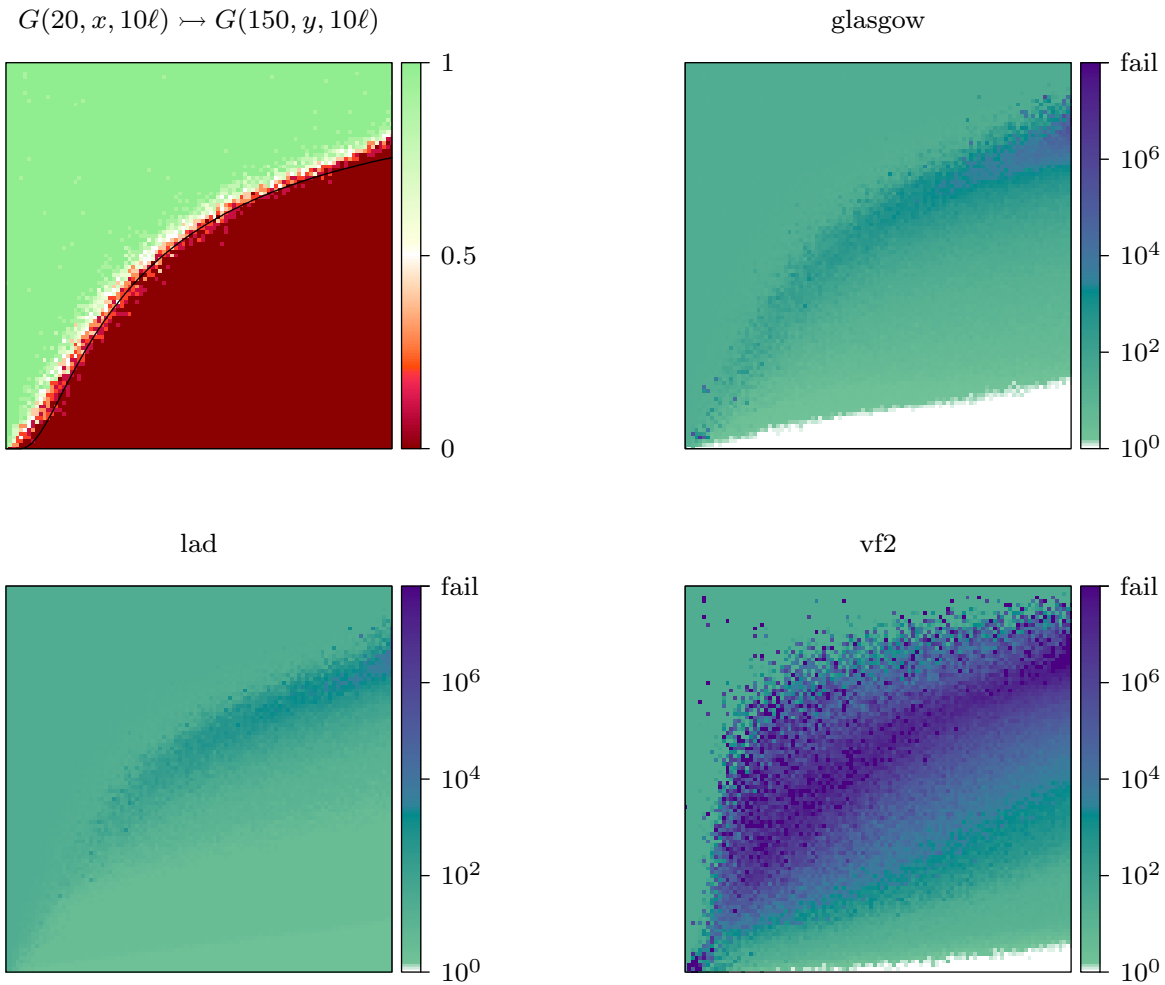
Why I Don't Like VF2



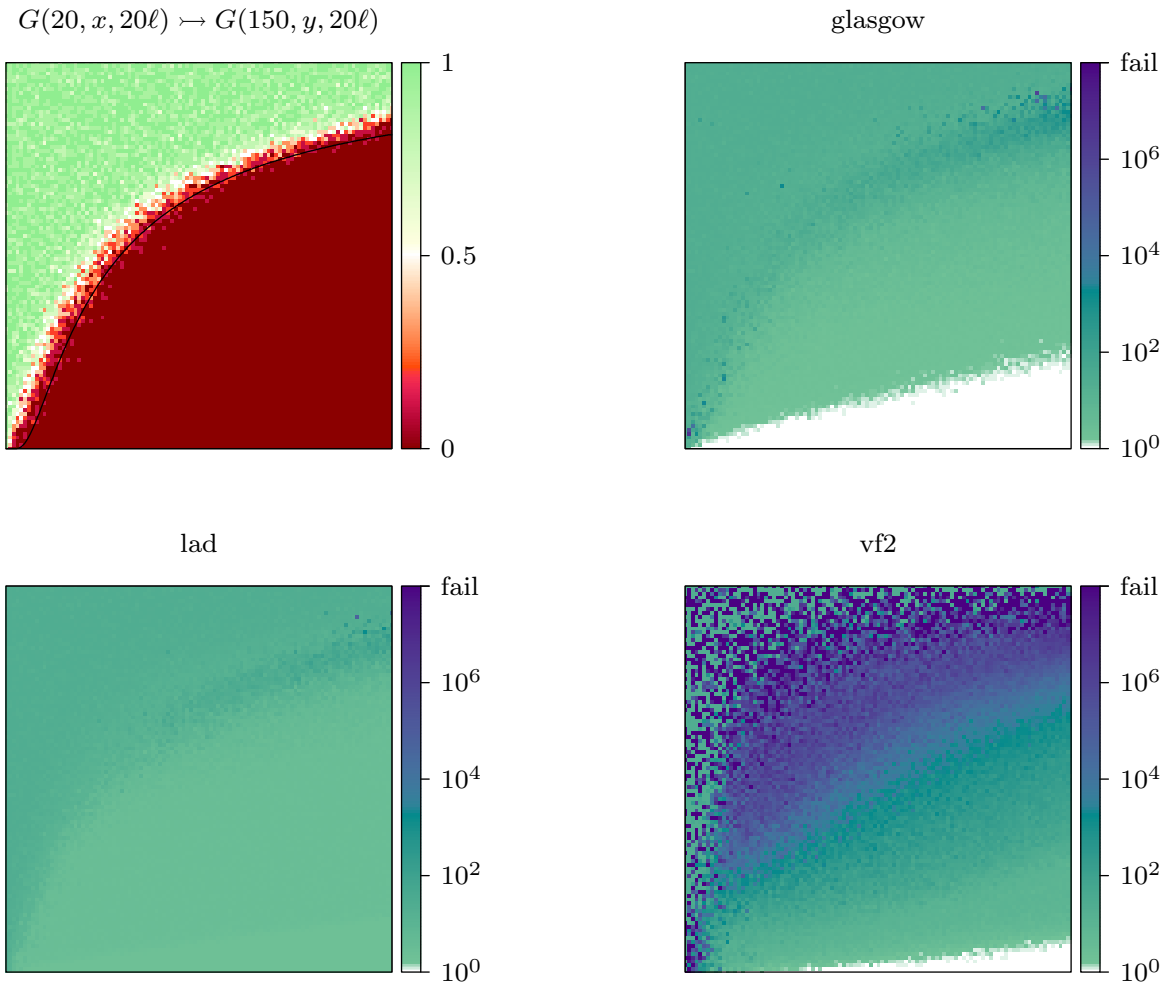
Why I Don't Like VF2



Why I Don't Like VF2



Why I Don't Like VF2



Why I Don't Like VF2

"There are (not so many) instances for which creation of a clear design is prohibitively slow in the current implementation that evaluates subgraph isomorphism with the VF2 algorithm by Cordella, Foggia, Sansone, and Vento (2001) as implemented in **igraph**. Recent experiences with a few of these showed that the LAD algorithm (Solnon 2010) was very fast in ruling out impossible matches, where VF2 took a long time."

R Package FrF2 for Creating and Analyzing Fractional Factorial 2-Level Designs. Ulrike Grömping. Journal of Statistical Software, Vol. 56 No. 1, Pages 1–56.

Scurrilous Remarks Regarding Graph Databases

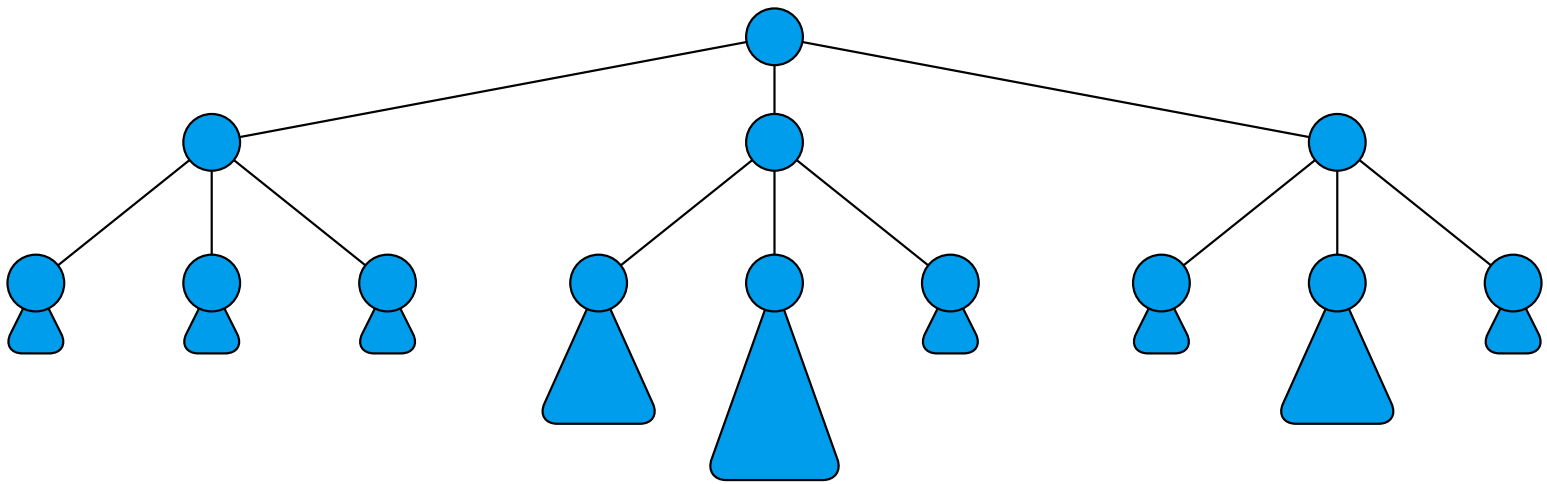
$$T_{search} + |C_q| \cdot (T_{io} + T_{iso\_test})$$

"The value of T_{iso_test} does not change much for a given query"

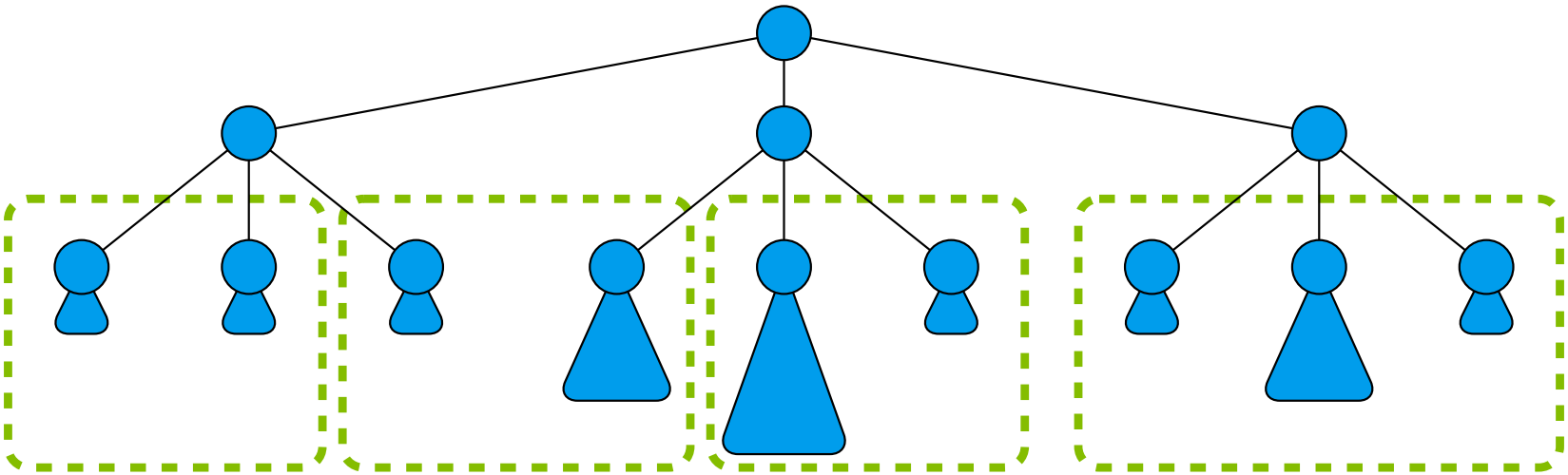
"Sequential scan is very costly because one has to not only access the whole graph database but also check subgraph isomorphism. It is known that subgraph isomorphism is an NP-complete problem. Clearly, it is necessary to build graph indices in order to help processing graph queries."

Graph indexing based on discriminative frequent structure analysis. Xifeng Yan, Philip S. Yu, Jiawei Han. ACM Trans. Database Syst. Vol. 30 No. 4, Pages 960–993.

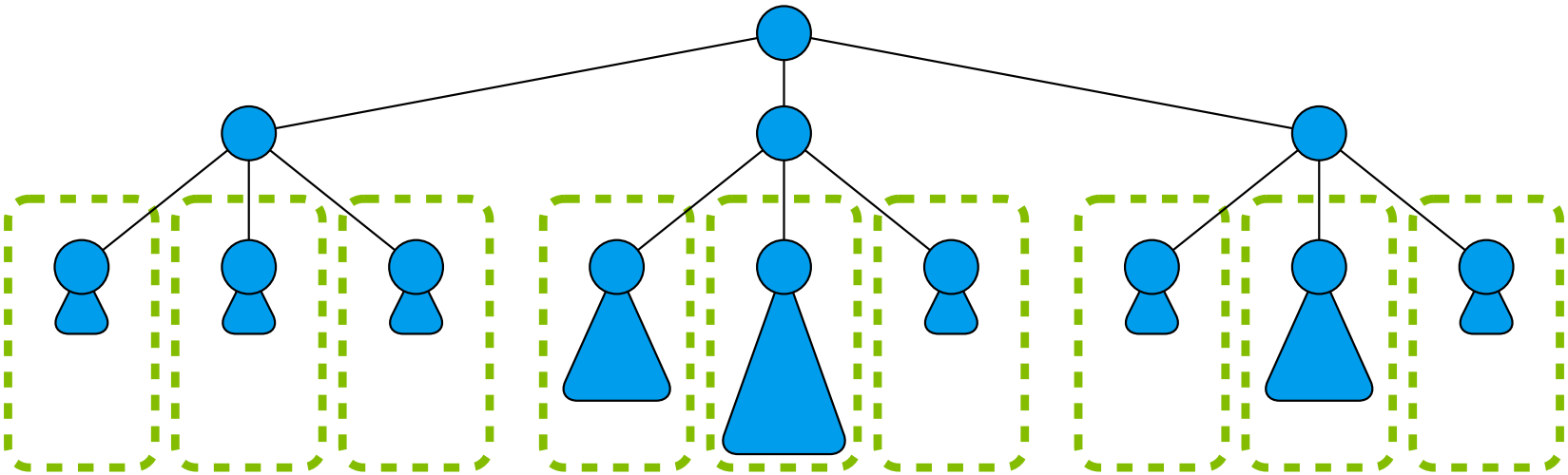
Parallel Search



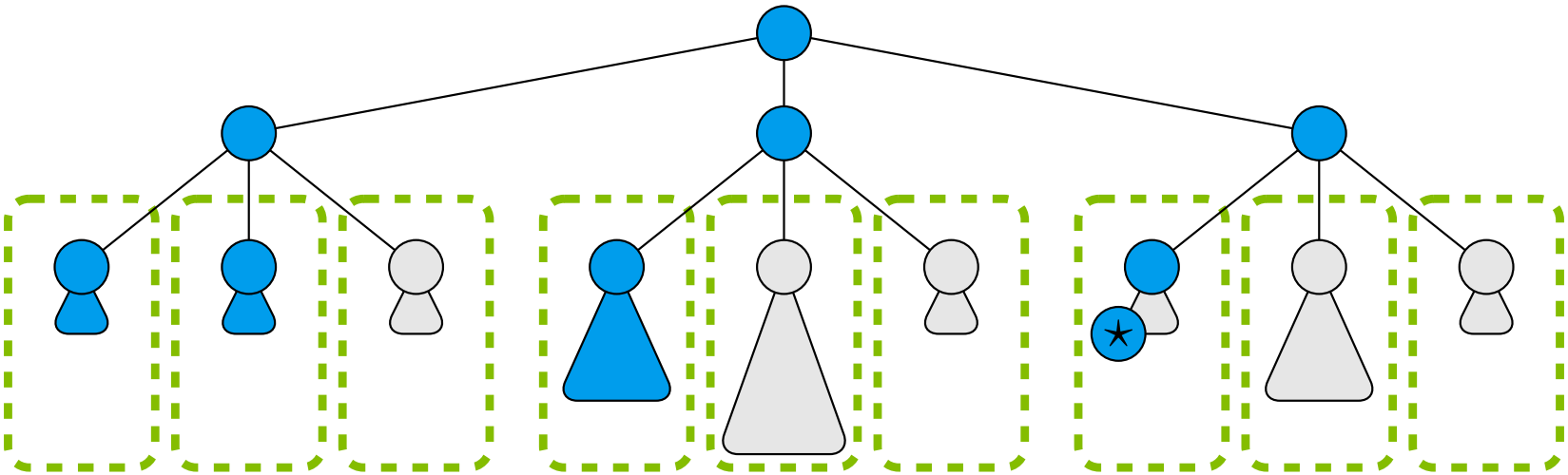
Parallel Search



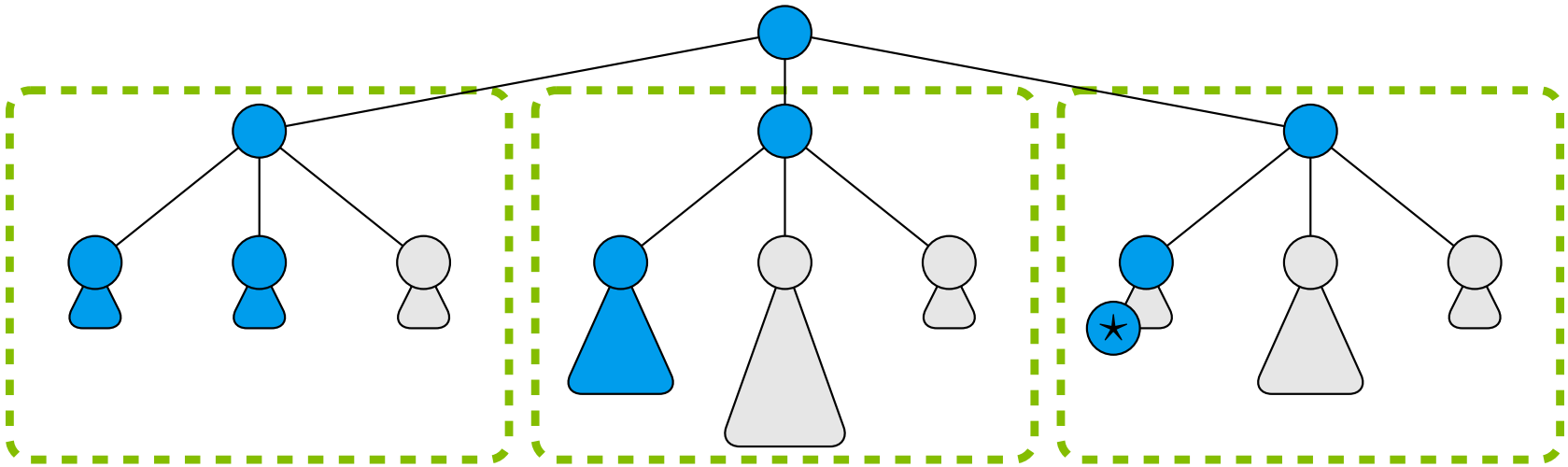
Parallel Search



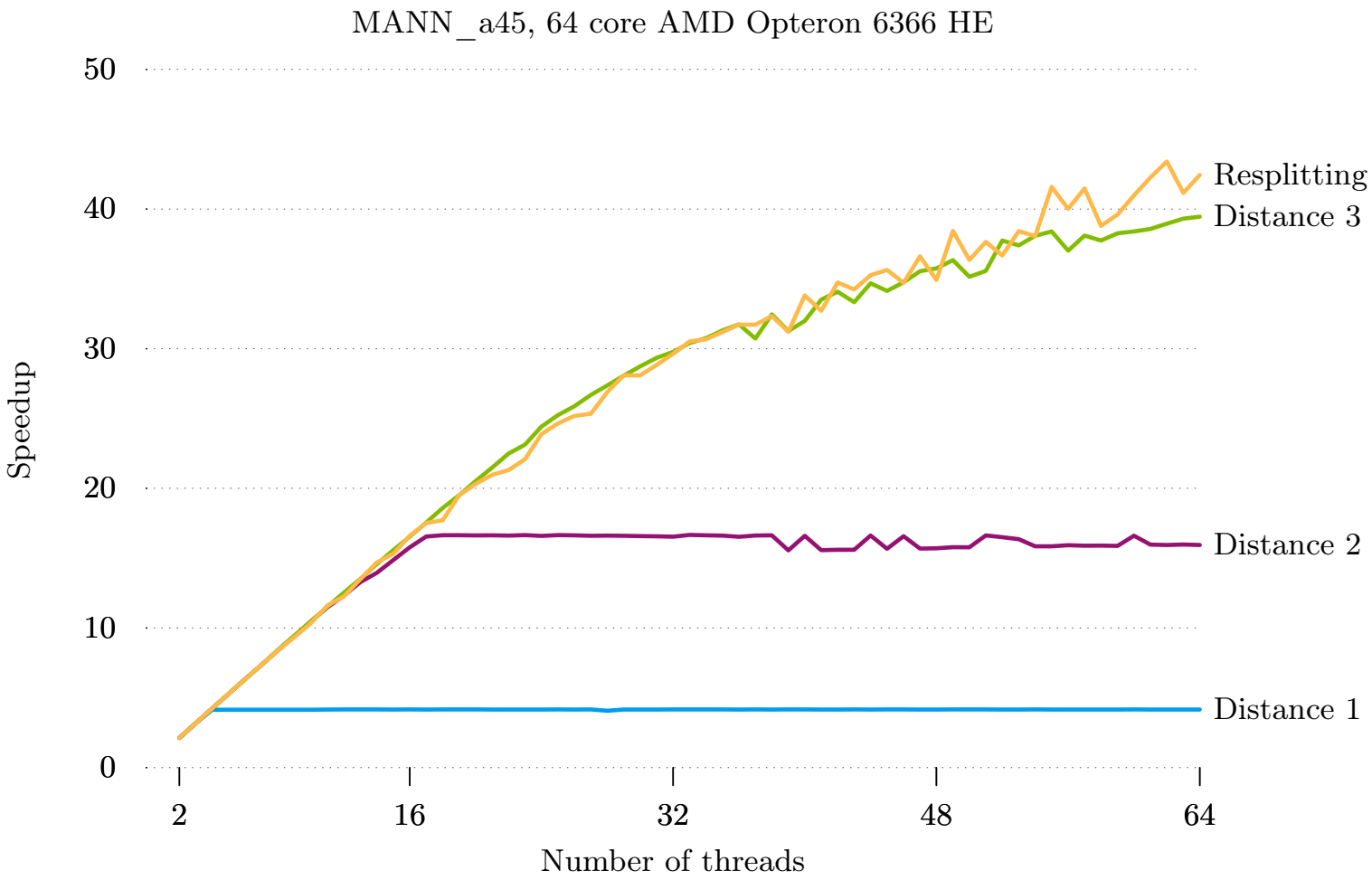
Parallel Search Order Matters



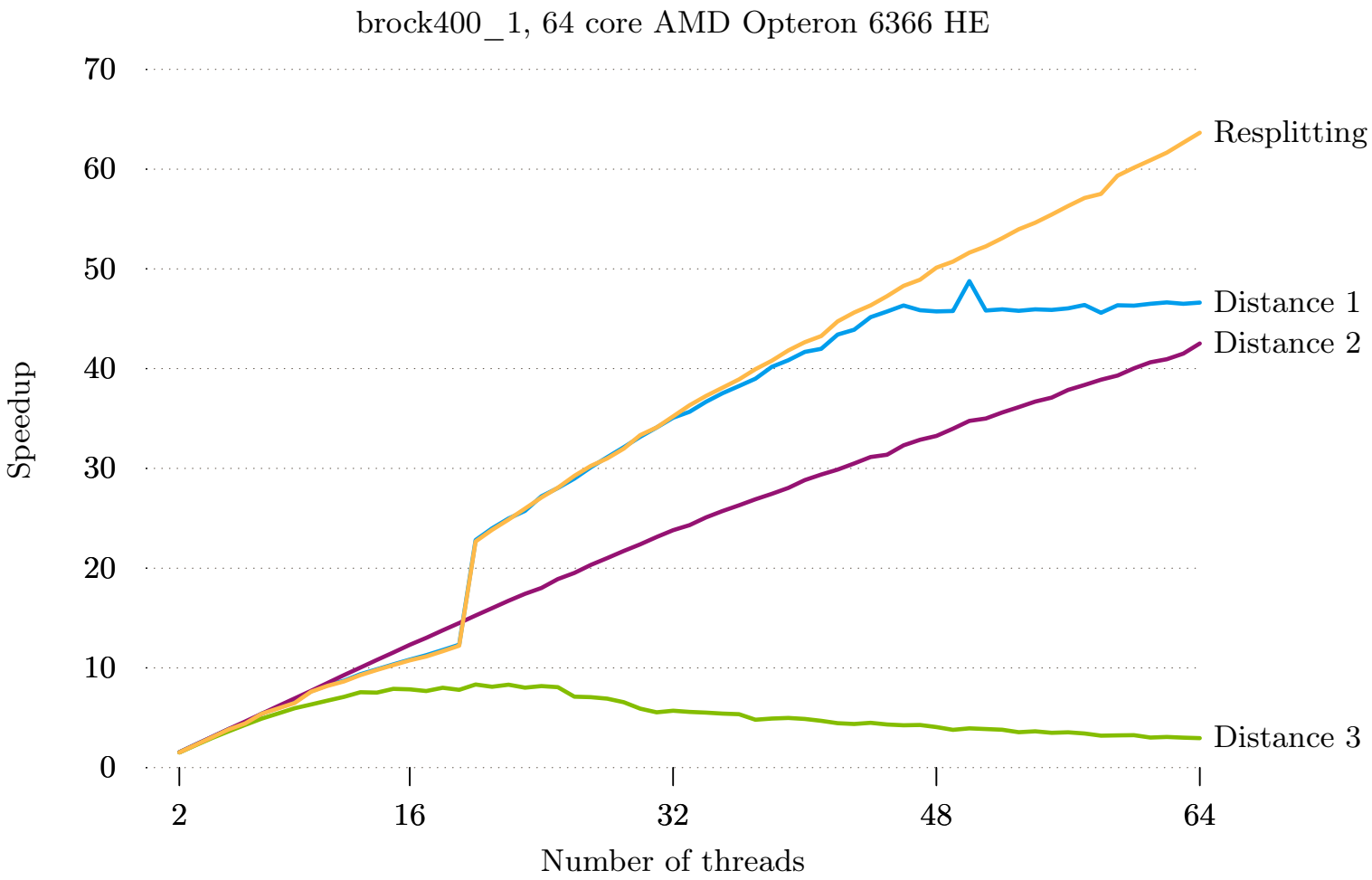
Parallel Search Order Matters



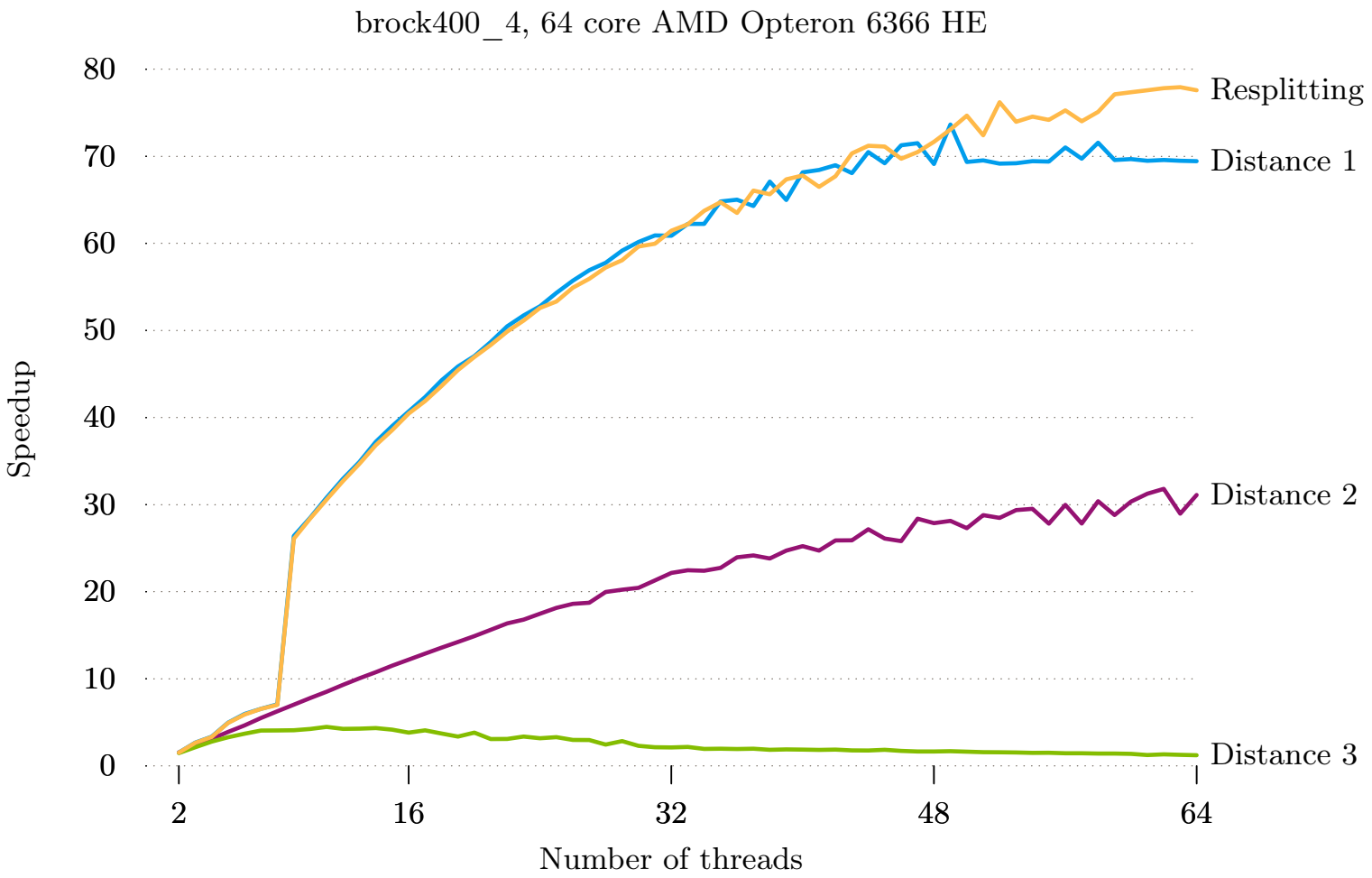
Parallel Search Order Matters



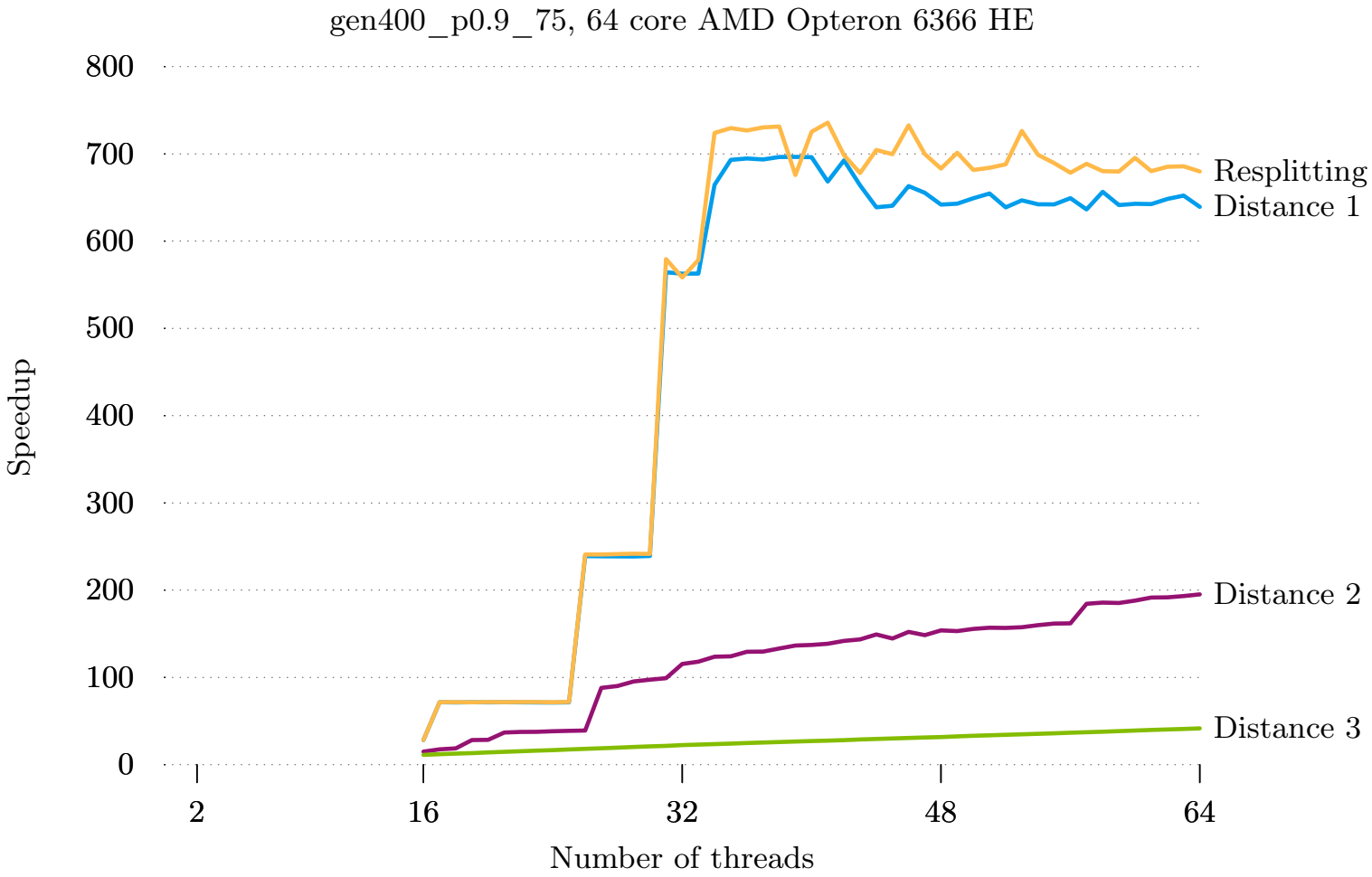
Parallel Search Order Matters



Parallel Search Order Matters



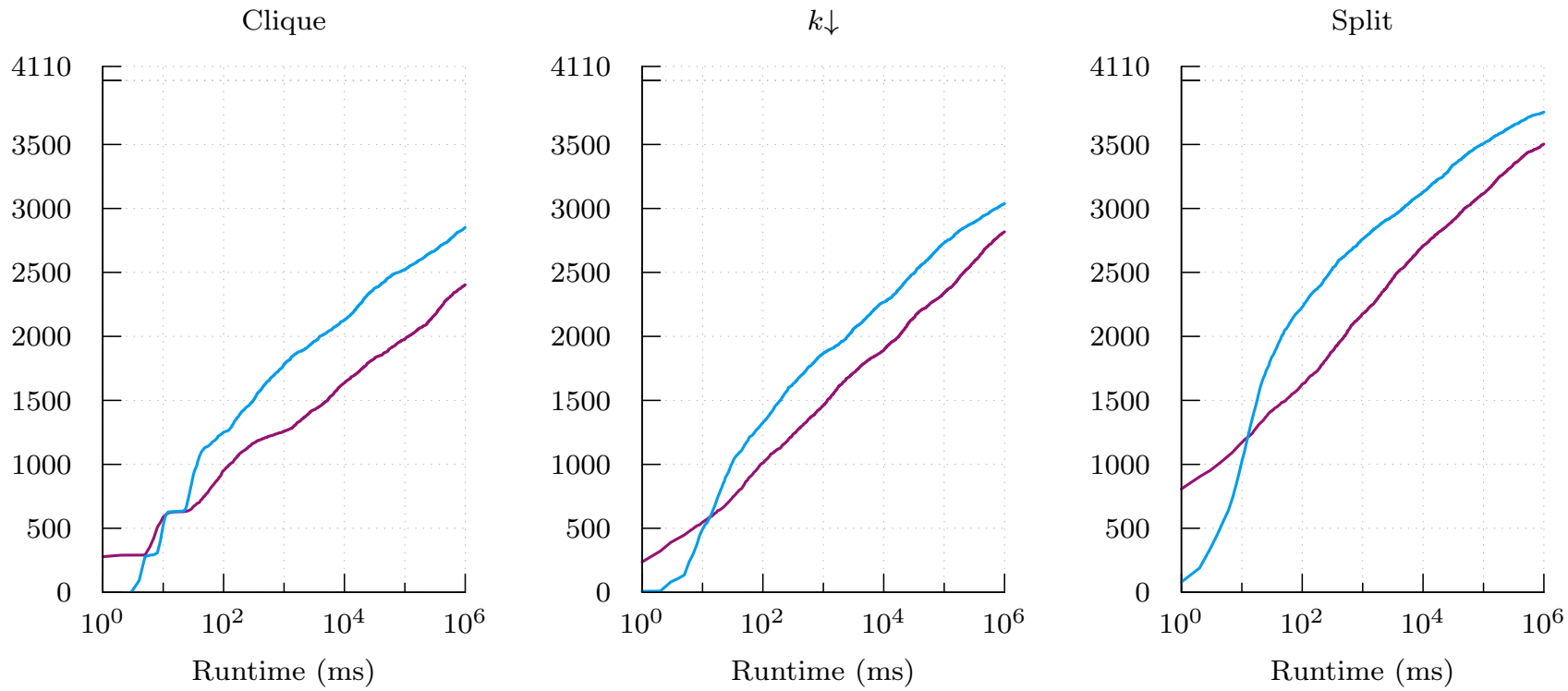
Parallel Search Order Matters



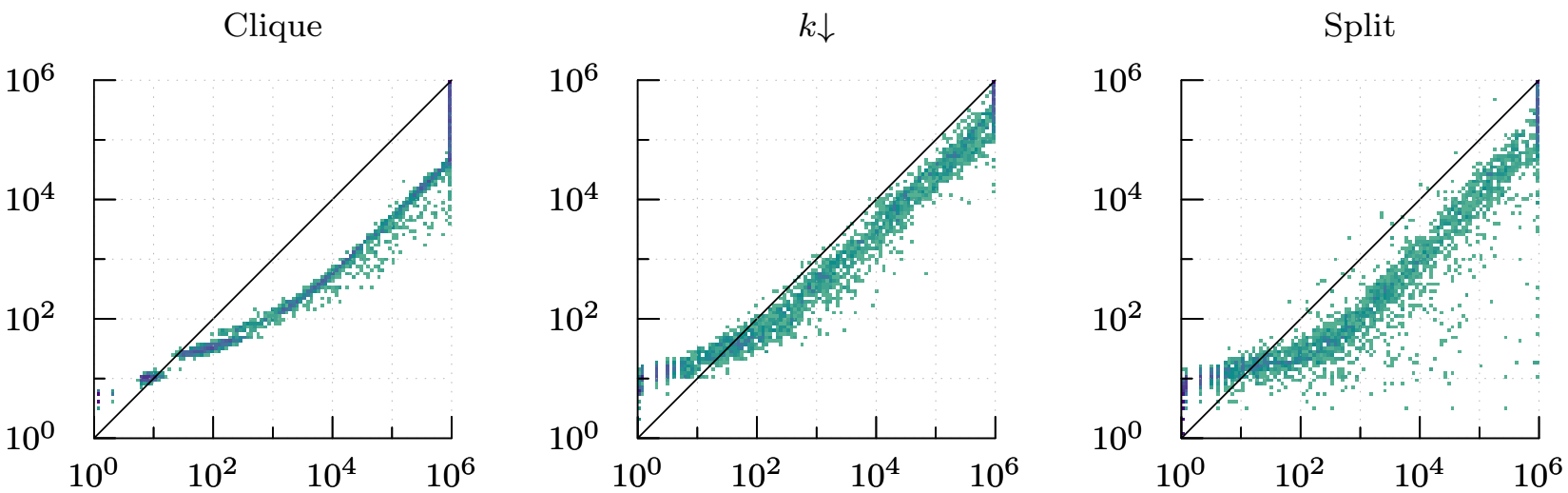
Desirable Properties from Parallelism

- **Beneficial**: the parallel algorithm should obviously be better, in aggregate.
- **Risk-Free**: no exponential slowdowns from introducing parallelism.
- **Reproducible**: the same algorithm, instance, and hardware, run twice, will take roughly the same amount of time.
- **Safely Scalable**: adding cores cannot introduce an exponential slowdown.

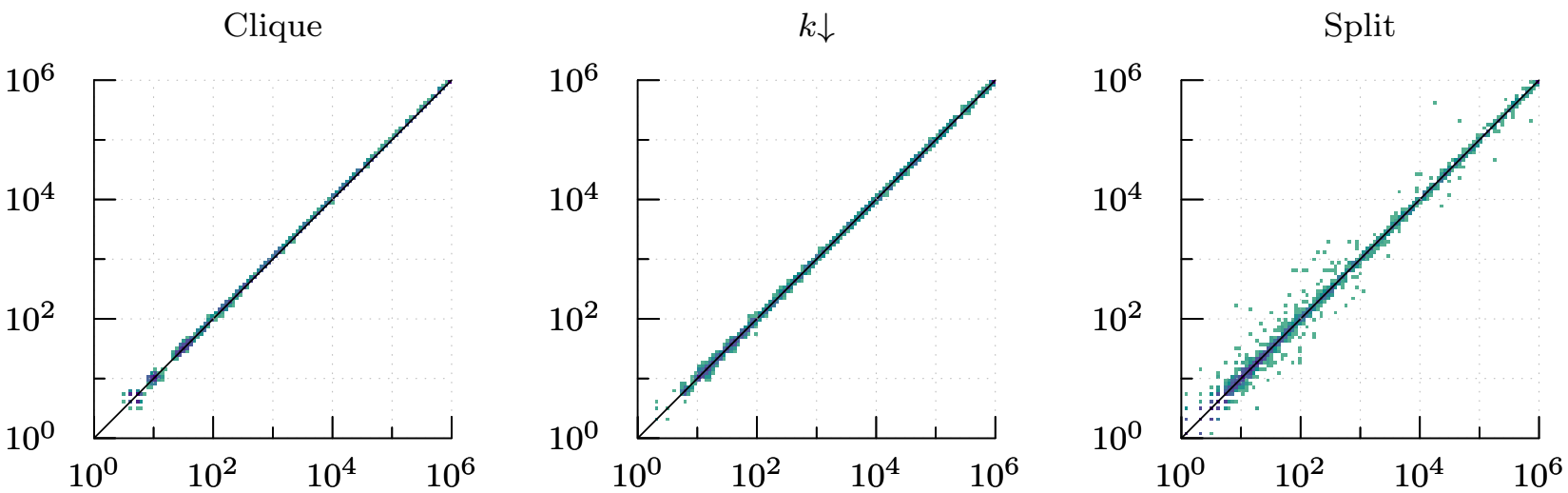
Parallel Search (Done Right) is Beneficial



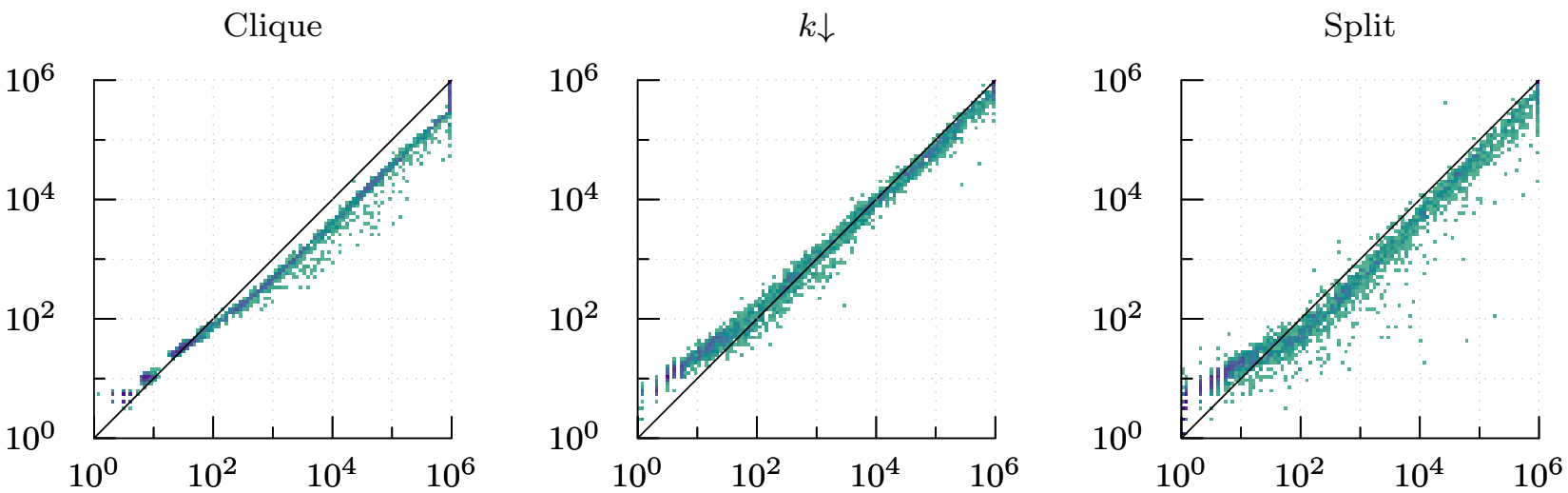
Parallel Search (Done Right) is Risk-Free



Parallel Search (Done Right) is Reproducible



Parallel Search (Done Right) is Safely Scalable



Next...

- Learning, and subgraphs modulo theories.
  - 1UIP produces really bad conflict explanations.
  - 2WL is too slow.
  - VSIDS aren't as good as tailored heuristics.
  - Polarity matters.
- How do we engineer a subgraphs library that isn't awful?
- High level modelling for graphs.

Collaborators Wanted!

- Symmetries.
- A better understanding of random graphs and hardness.
  - Wanted: collaborator who understands linearity of expectation, the second moment method, finite size scaling, and things that don't come from normal distributions.
- Making (distributed) parallelism less unpleasant to implement.

http://www.dcs.gla.ac.uk/~ciaran

ciaran.mccreesh@glasgow.ac.uk