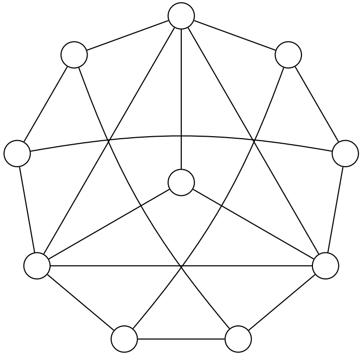
Computational Experiments, Sample Size Fifteen Billion

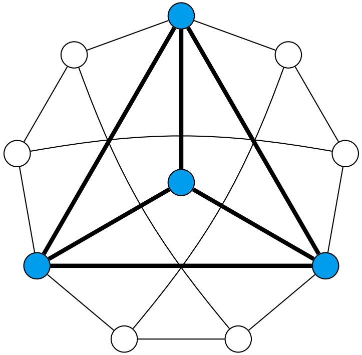
Ciaran McCreesh

University of Glasgow

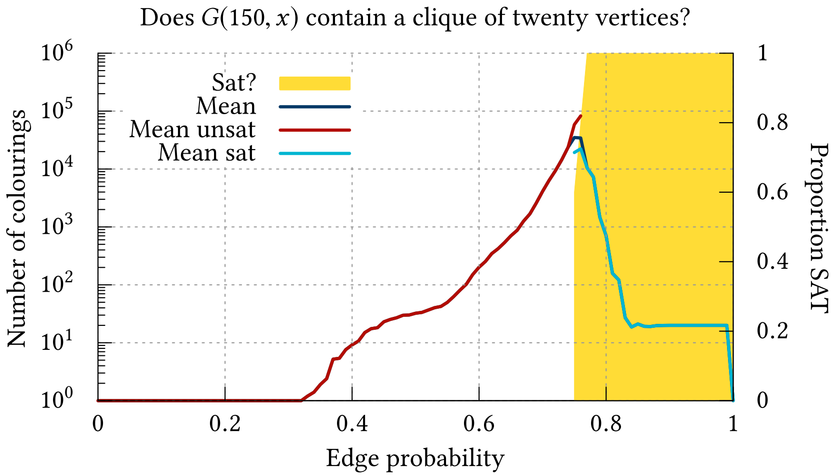
Cliques



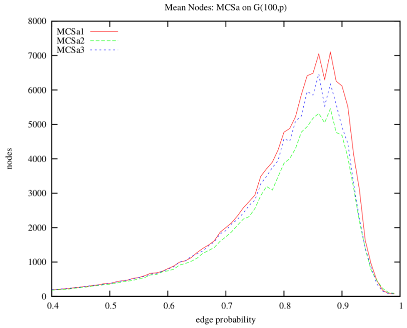
Cliques

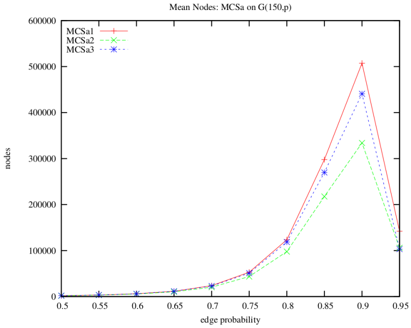


Phase Transitions



Optimisation, an Incomplete Picture





Patrick Prosser: Exact Algorithms for Maximum Clique: A Computational Study. Algorithms 5(4): 545-587 (2012). Both plots have 100 samples per density step. The left-hand plot seems to go up in density steps of 0.01, and the right-hand plot, 0.05.

Which is the Hardest Density?

- Which density is hardest, for the optimisation problem?
- Does this change depending upon the number of vertices? The algorithm used? The random graph model selected?
- Is this the same as the hardest density for the decision problem, if we can also pick the decision number? And if so, which decision number do we pick?

Back of the Envelope Feasibility Estimates

- Steps of 0.05 are obviously not enough, and 100 samples looks to be far too low. Also, let's go from density 0 to density 1, just in case something interesting happens.
- We have a faster implementation, so $G(150, x)$ rarely takes more than a second for any value of x, but $G(200, x)$ can take minutes.
- Increase density from 0 to 1 in steps of 0.001? This is around one pixel per step.
- Mean runtimes seem to settle down at around 10,000 samples. We probably want 100,000 samples to be safe.

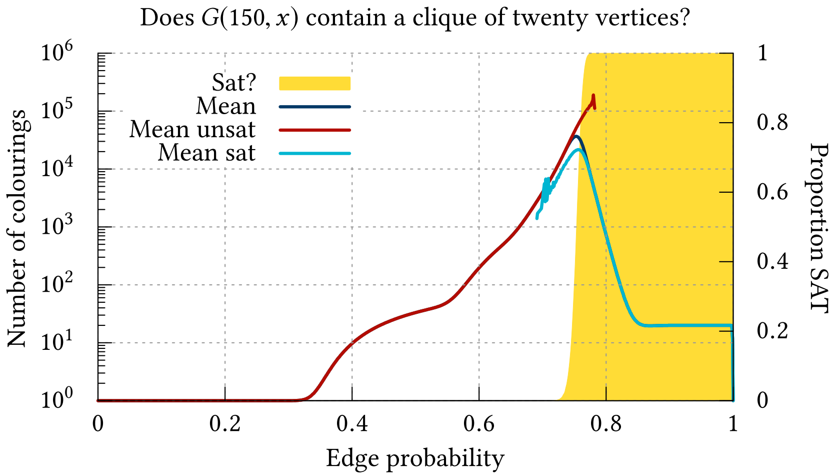
How Much CPU Power?

- One laptop core: maximum clique in $G(150, x)$, density going from 0 to 1 in steps of 0.01, one sample per point. Takes around four seconds.
- Ten times more density steps, one hundred thousand times the sample size: four million seconds, or 46 days.
- Then, 150 separate decision problem curves. Potentially 18 years (but probably less?).
- Conveniently, this is around 150,000 core hours.

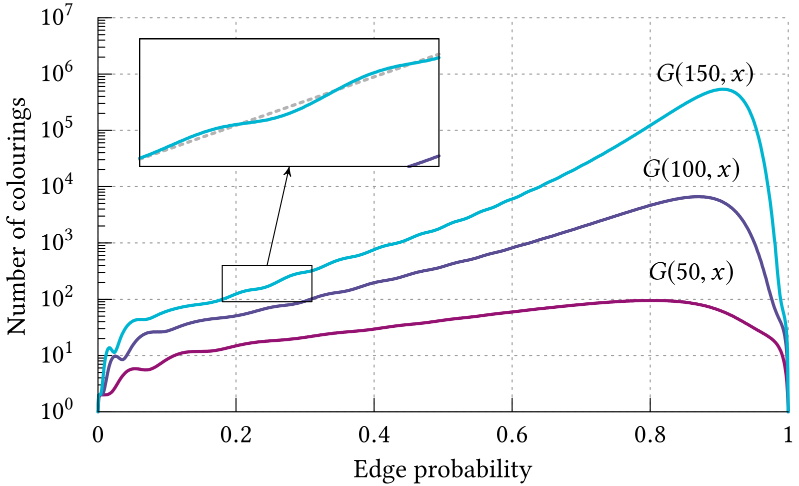
What About Storage?

- Source code, binaries, support libraries, etc: likely under 1GByte.
- 100,000,000 instances, 100KBytes per instance gives 10TBytes of space.
- 1KByte output per instance, for decision and optimisation, is another 15TBytes.
- Inconveniently, this is *probably* at least a couple of orders of magnitude too large.

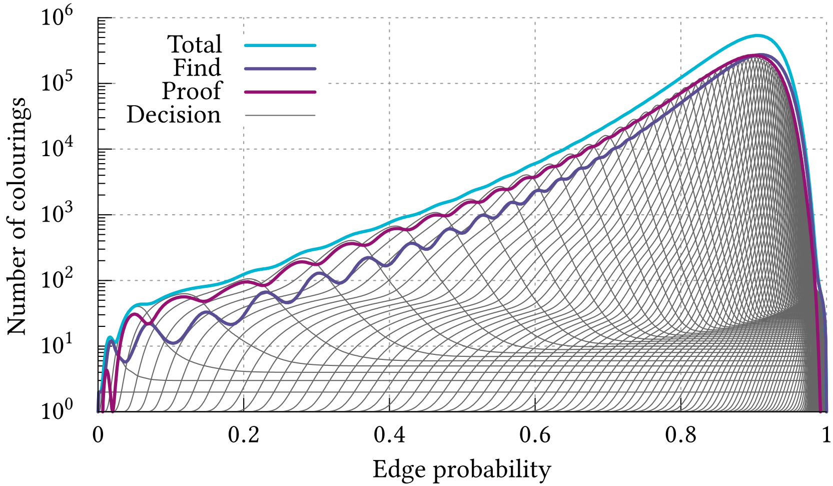
Phase Transitions, Refined



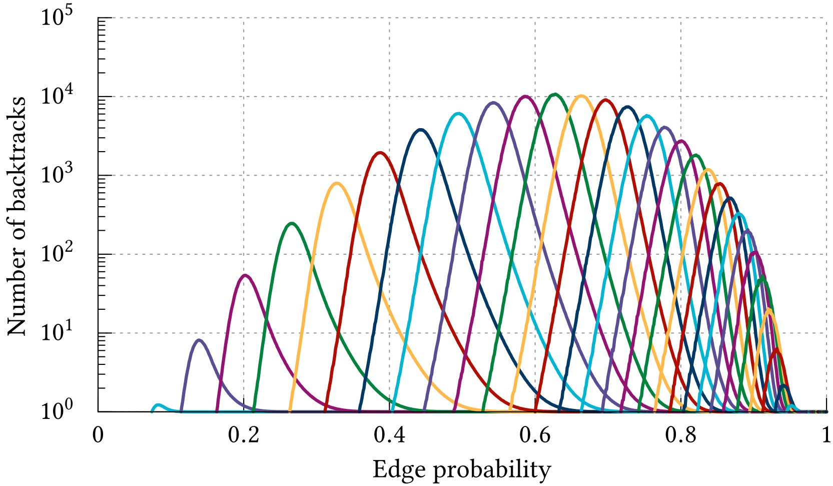
Optimisation, Refined



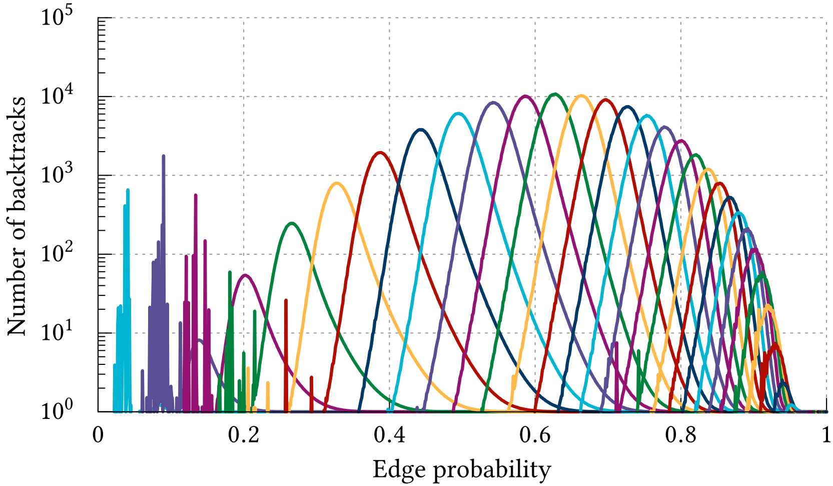
Decision and Optimisation



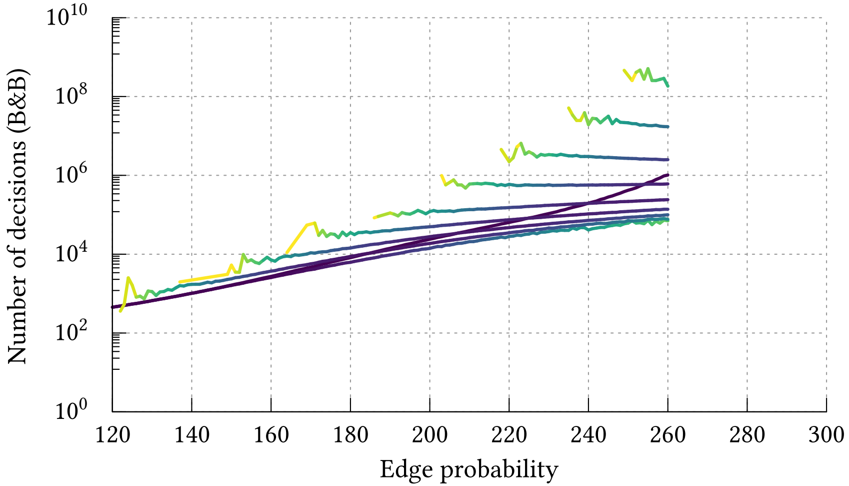
Colouring?


Number of backtracks
10^5
10^4
10^3
10^2
10^1
10^0
0
0.2
0.4
0.6
0.8
1
Edge probability

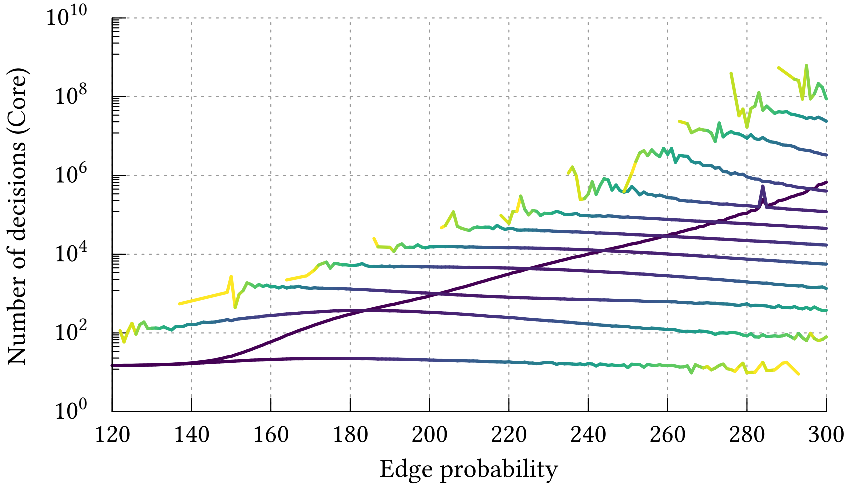
Colouring?


10^5
10^4
10^3
10^2
10^1
10^0
Number of backtracks
0
0.2
0.4
0.6
0.8
1
Edge probability

MaxSAT?


10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
10^{0}
Number of decisions (B&B)
120
140
160
180
200
220
240
260
280
300
Edge probability

MaxSAT?



10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
10^{0}
Number of decisions (Core)
120
140
160
180
200
220
240
260
280
300
Edge probability

Other Experiments Not Appearing In This Talk

- What exactly is going on with the wiggles?
- What does this have to do with search order heuristics?
- How can we use this knowledge to design better algorithms?
- Is there a tradeoff between anytime behaviour and proofs of optimality?
- Have we been doing branch and bound algorithms wrong the whole time?
- There are types of problem (e.g. matching) where something different happens.

So What?

- Very large sample size experiments yield interesting results!
- This is potentially much more interesting than just using an increase in CPU power to look at larger and larger graphs.
- Why not give it a go for your own problems?

Your Own Laptop or Desktop

- Your processor's clock speed will change dynamically depending upon temperature, potentially by a factor of two or three.
- You can damage some laptops by running them constantly at full CPU load for hours...
- Memory corruption is surprisingly common.

The School’s Unix Machines

- Varying hardware, software, ownership, and access policies…

The FATA Nodes

- Six identical Linux servers: dual Xeon E5-2697A v4 CPUs, 32 cores, 512GBytes RAM.
    - Officially, two general use, four for ModOptGraph.
    - Carefully calibrated for reproducible runtimes.
    - If you cannot make use of 32 cores simultaneously, your desktop machine is faster.
- Calendar booking system: email me for details.
- Currently oversubscribed…
    - So we'd prefer it if you only booked the time you needed.
    - Interactive jobs are possible, but preferably only when the machines aren't heavily booked.
- Approximate cost: £22k for a pair. (Easily included in an EPSRC standard grant! Also just fits together with a PhD studentship in an LKAS application.)

The University’s Tier Three HPC Cluster

- It exists…
- Mixed hardware, mostly very out of date, not suitable for measurements.
- IT Services run occasional training courses.
- Currently oversubscribed…

EPSRC National Resources

- EPSRC Tier Two HPC: Cirrus, at EPCC in Edinburgh.
- 280 machines, each with dual 18 core Xeon E5-2695 v4 processors and 256GBytes RAM. (Similar to the FATA nodes.)
    - Although you can't often use all 280 machines at once... Usually you'll not get more than twenty.
    - Access is non-interactive only, and controlled by a job launcher.
- Some shared disk space, but it's not very fast and you're unlikely to be allowed TBytes of space.

Instant Access

- Instant access: 20,000 core hours for feasibility studies and to help fill in a proper technical assessment form.
- One small form to fill in. Expect to hear back within a few days.

Scottish Academic Access

- Up to 1,000,000 core hours over one year.
- Limit one per researcher.
- Postdocs can definitely apply as lead investigator, and the rules don't explicitly forbid PhD students as lead either.
  - Can go on your CV. 1,000,000 core hours is a grant with nominal value £9,000.
- A short application plus a technical assessment is required.
- Expect to hear back within a week.

As Part of an EPSRC Grant

- No limit on requested hours, although more than 10,000,000 core hours is probably unlikely to make it past the technical assessment.
- A couple of extra boxes on Je-S.
- Some text in the Justification of Resources.
- Requires an approved technical assessment before submission (usually takes less than a week to hear back).

EPSRC RAP Open Access

- Three calls from EPSRC per year. Relatively high funding rate.
- No limit on requested resources, although most funded grants have been no more than 1,000,000 core hours.
- Not a full Je-S application, just a case for support and a technical assessment. Covers resource access only, no money, staff or technical support time is allowed. No internal costing needed.
- Calls explicitly allow:
  - Projects that do not warrant a full grant application.
  - Collaborative projects with international or industry partners.
  - Joint applications from students (as Co-Is) with proven HPC experience and their PIs.
  - Projects that link consecutive standard grant applications or that aid the preparation of a grant or fellowship application.

The Technical Assessment

- Demonstrate that you know how to use HPC equipment, that your code can run successfully when parallelised at large scale, that your resource estimates are sane, and that you know how to get your data in and out.
- Some of the scalability questions are unsuitable for software development and empirical algorithmics, where we're not looking for strong linear scalability, or where we're implementing new software. There are staff at EPCC who understand this, and it isn't a problem.
- Talk to me, Blair Archibald, or Phil Trinder if you want to see successful completed versions, or if you need to list an experienced name for technical support.

You Have to be Parallel

- Your desktop machine is faster than HPC equipment.
- HPC has a lot of parallel throughput, not sequential speed.
- Your software must support this!
- One way of doing this is through running lots of instances in parallel.

Job Launchers

- When using Cirrus, you **must** use the job launcher.
- Jobs are run entirely non-interactively on a fresh environment on an unknown machine, and must save any output to disk.
- Jobs define their own timeouts (with limit of 96 hours), and are killed after that amount of time.
- The job queue is designed only for a few thousand jobs, and job startup time can be in the tens of seconds, so one instance per job is unlikely to be feasible.
- You "pay" per-machine, so whatever you run in that job has to be able to make use of 36 cores. So, if your solver isn't parallel, you need a second level of job management.
- In other words, the job launcher is an additional level of inconvenience, not a solution.

Parallelism Using `parallel`

- The `parallel` utility just runs a long sequence of commands in parallel, using a specified number of jobs, with automatic load balancing.

Parallelism Using `make`

- The `make` tool is usually used for automated compilation, but is flexible and supports parallel job execution with dependencies.
- A file gives a series of rules for creating files out of other files. Any file that does not exist is created by running an external program, recursively.
- So, you can define a target for a “blah.out” file for each instance, and have a rule for creating a “.out” file using your solver.

Home-Grown

- At the millions of instances scale, storing result files from individual runs is impractical.
- For the wiggly lines plot, I ended up writing software from scratch to handle instance generation, solver running, load balancing, and result parsing and aggregation.
- I'm not sure whether it's worth making this software more generally useful.

Failures and Reproducibility

- At this scale, you **will** encounter hardware failures.
- Cirrus will refund you your hours for the job that failed, but you have to be able to re-run it yourself.
- Being able to rerun failed results is necessary.
- Also, what about outliers and crashes? What if one instance in one hundred thousand does something weird? Can you recreate that instance and rerun it?
- You **must** have control over random seeds for instance generation and for solvers.
- What if your solver uses, e.g. a timer-based restart or presolving policy?

Load Balancing

- If one in a million instances takes a day to run, and the rest take under a minute, are you going to be able to keep all 36 cores you're paying for busy?
- What if you did a back of the envelope calculation and set a timeout of one hour?

Instances, and Reading from Standard Input

- If your solver reads instances from files, how are you going to manage these files? You probably can't store a billion instance files on disk indefinitely.
- What if the job queue likes you, and too many of your jobs get run simultaneously?
- You can write an instance generator, have your solver read instances from standard input, and use the shell to pipe in input.
    - Except certain commercial solvers require a particular file extension for their input files.
    - If you write your own multi-threaded job launcher, implementing this correctly on Linux is a nightmare, and is impossible on some BSD variants.

Limited I/O

- I/O is very slow on Tier Two HPC.
- Going through and parsing your results files after the fact can take longer than the original runs.
- Did you know that there's a maximum number of files that you can have in a directory, and that that number isn't as big as you want it to be?
- There's also a maximum you can have per filesystem, and the default for that number isn't as big as you want it to be. (On the FATA nodes we don't use the default. Most other places do.)
- Solvers that write log files cause all kinds of problems.

Measuring Time

- Did you know your clock can run backwards, or jump around arbitrarily?
- Modern C and C++ do support monotonic steady clocks, if you explicitly ask for them, but most solvers don't do this.
    - Particularly harmful: several early empirical algorithmics papers suggested measuring CPU time (e.g. `getrusage`), not wallclock time.
- No way of measuring time accurately in many other languages, including Java.

Measuring Search Nodes

- If you can find a good solver-independent measure of complexity, use that.
- But it can hide a great deal!
- Also, beware integer overflow…
    - Almost certainly an issue when calculating averages.
    - Floating point doesn't fix this!

Using Your Favourite Programming Language

- Proven track record for HPC: C, C++, Fortran.
- Struggles due to memory management problems: Java.
  - Added bonus: running 36 sequential Java processes on one high-RAM machine will kill all of the VMs once garbage collection kicks in.
- Basically no support for compute-intensive parallelism: Python.
  - *Maybe* usable if your time is spent doing large matrix calculations, etc, where the work is done in a C library.
- You're faster rewriting it as single-threaded C code: Haskell, Erlang.

Dealing with Buggy and Otherwise Annoying Solvers

- Many research quality solvers will just about produce the right answers most of the time for the instances tested in the paper, and nothing else. Unless there is a community with rigorous solver competitions (e.g. SAT), expect to have to deal with bugs.
  - What do you do if a solver crashes on one in a thousand instances? What if it just hangs forever?
  - What if it sometimes produces the wrong answer?
- Most academics are not programmers.
- Most academics don't check code written by their students.
- Most reviewers don't check code.
- Commercial solvers also have problems, and there is a commercial incentive to keep bugs secret.
- Software licenses? Dependencies? Language version? Cirrus has a fairly conservative set of software installed.

Writing a Good Solver is Hard, and There is Little Help

- Little training or support available:
  - The Software Sustainability Institute.
  - A very small number of EPSRC Research Software Engineer Fellowships.
  - Software development is basically unregulated, even for medical and safety critical systems.
- Software engineering courses teach the wrong lessons for small-scale code (e.g. testing won't find bugs in SAT solvers). We don't teach much algorithm engineering, and most CS graduates can't implement binary search correctly.

Few People Care

- Your reviewers probably don’t care whether your code is robust and correct, if they care about the implementation at all.
- Grants can be rejected for containing “too much software development”.
- Most empirical papers just do horse races.
- For some reason “I spent three months re-implementing someone else’s algorithm” can be a hard sell as a research output.

However…

- Most computer scientists can’t or won’t do this stuff, so if you can and will, maybe it gives you an edge?

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

ciaran.mccreesh@glasgow.ac.uk