Combinatorial optimization on quantum computers

Ashley Montanaro

Phasecraft Ltd
School of Mathematics, University of Bristol

Advert

We are hiring at the University of Bristol! We're looking for postdoctoral-level researchers and PhD students.

Further reading:

- Quantum algorithms: an overview. AM, npj Quantum Information volume 2, 15023 (2016)

Combinatorial optimization

Combinatorial optimization problems are characterised by needing to search over exponentially many possible solutions.

For example:

- Colouring a graph with the minimal number of colours such that no adjacent vertices share a colour;
- Finding the lowest-cost route that visits all of a set of cities;
- Determining if a system of linear equations over integers $\{0, 1\}$ has a solution.

Quantum computers can sometimes achieve a speedup in solving optimization problems over our best classical algorithms.

However, the speedup (when provable) is usually quadratic, as opposed to exponential (e.g. running time $2^n \to 2^{n/2}$)

Today's talk

Today I will discuss some quantum algorithms for solving optimization and constraint satisfaction problems (CSPs):

- Grover's algorithm implies quadratic speedup over classical unstructured optimization
- Quantum speedup of backtracking and branch-and-bound algorithms [AM '15, AM '19]
- Quantum speedup of dynamic programming algorithms [Ambainis et al '19]

These algorithms generally have provable bounds on their performance, but still have exponential running time and require a fault-tolerant quantum computer.

I will finish by discussing concrete bounds on the performance of these algorithms and how realistic they are [Campbell et al '19, Sanders et al '20, Cade et al '22].

From Grover's algorithm to unstructured optimisation

If we run Grover's algorithm with K "marked" elements ($\{z : f(z) = 1\}$), we can find a marked element with $O(\sqrt{2^n/K})$ expected uses of f (and we don't need to know K).
We can use this to solve hard optimisation problems too!

Imagine we have $f : \{0,1\}^n \to \mathbb{Z}$ and we want to find z such that $f(z)$ is minimised.

1. Maintain a threshold t, initially set to $t = f(0^n)$. Then repeatedly:
    1. Use Grover's algorithm to find x such that $f(x) < t$, if such an x exists
    2. If successful, set $t = f(x)$, otherwise stop and output the last x found.

The overall expected number of uses of f is $O(\sqrt{2^n})$. Why?

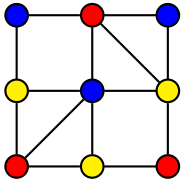
- Each use of Grover gives us a random element smaller than the threshold
- So the expected number of marked elements drops by a factor of 2 each time
- So the overall complexity is $O(\sqrt{2^n/K}) + O(\sqrt{2 \times 2^n/K}) + \dots + O(\sqrt{2^n}) = O(\sqrt{2^n})$, where K is the number of elements initially below threshold.

Beyond unstructured optimisation

Often we can achieve a better complexity than unstructured search or optimisation by using the structure of the problem we need to solve.

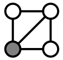
Two of the most prominent techniques for this are backtracking ("trial and error") and dynamic programming.

We can illustrate backtracking with graph k-colouring:

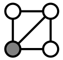


An NP-complete problem with a huge number of direct applications, including register allocation; scheduling; frequency assignment problems; …

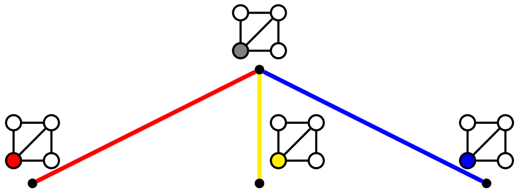
Colouring by backtracking

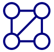


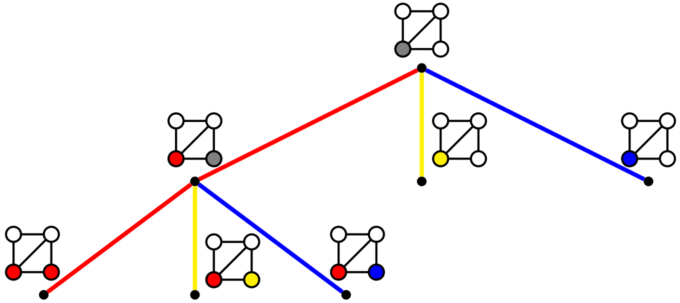
Colouring by backtracking

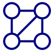


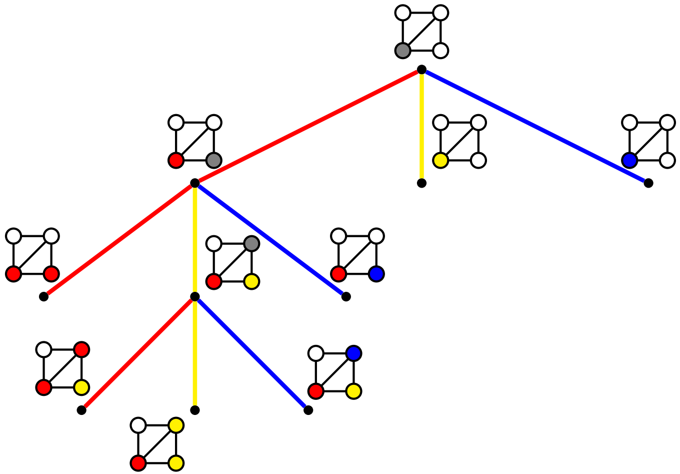
Colouring by backtracking

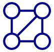


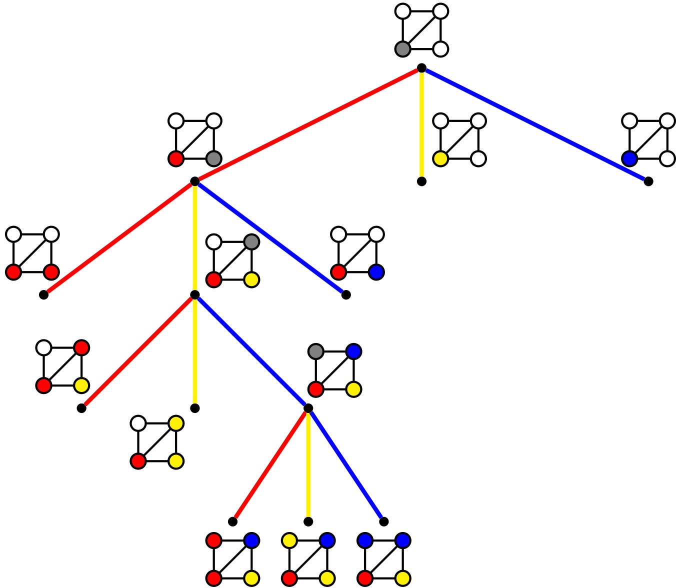
Colouring  by backtracking



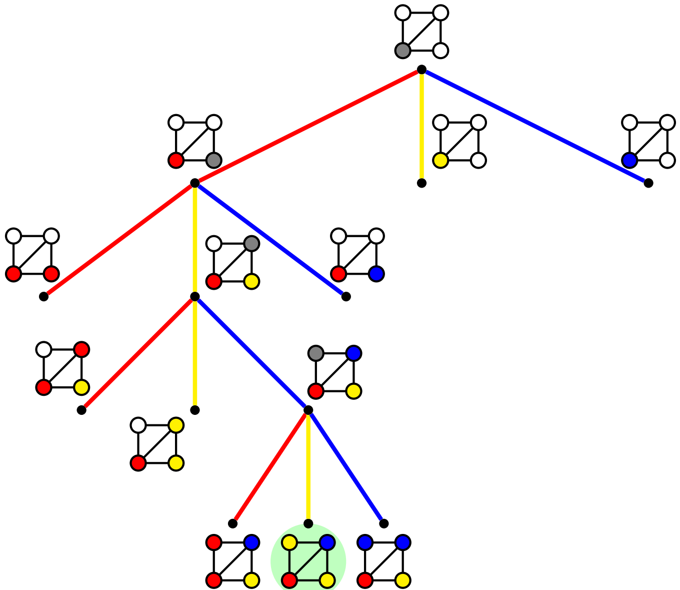
Colouring  by backtracking



Colouring  by backtracking

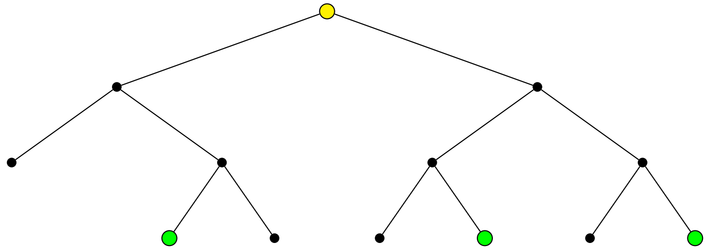


Colouring by backtracking



Search in a tree

Imagine we want to find a “marked” vertex in a tree where we only have local knowledge, starting from the root.

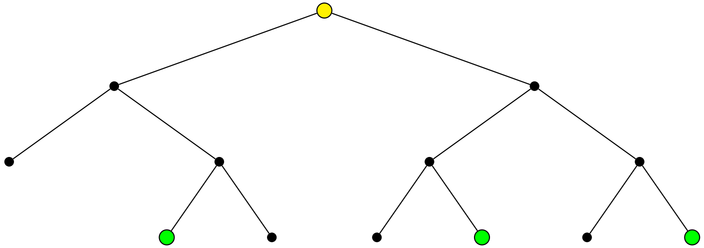


If the tree has T vertices, this requires $\sim T$ time classically in the worst case.

Quantum search in a tree

Theorem [Belovs '13]

There is a quantum algorithm that can detect existence of a marked vertex in a tree with T vertices and depth d, using $O(\sqrt{Td})$ queries.



The algorithm is based on a quantum walk in the tree.

From quantum search in trees to backtracking

- A backtracking algorithm solving a problem with n variables explores a tree of size T and depth n.
- The quantum walk algorithm for search in trees can be applied, yielding:

Theorem (informal) [AM '18]

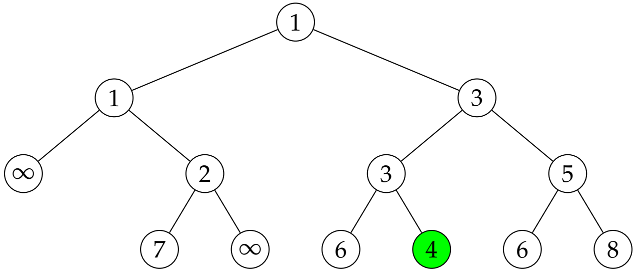
There is a corresponding quantum algorithm which finds a solution, or outputs that one does not exist, in time $O(\sqrt{T}\,\mathrm{poly}(n))$, with 1% probability of error.

- We normally think of T as exponential in n; in this case, the speedup is near-quadratic.
- Subsequent improvements to this algorithm: [Ambainis and Kokainis '17], [Jarret and Wan '18]

Branch-and-bound algorithms

The backtracking approach can be generalised to solve optimisation problems via a technique known as branch-and-bound, which is applicable whenever we have:

- A branching procedure that splits a set of potential solutions into subsets;
- A bounding procedure that returns a lower bound on the cost of any solution in a subset.



Quantum speedup of branch-and-bound

Theorem (informal) [AM'19]

Assume there is a classical algorithm that solves an optimisation problem using the branch and bound procedures T times. Then there is a quantum algorithm that solves the same problem using these procedures $O(\sqrt{T})$ times (up to lower-order terms).

- The quantum algorithm is based on the use of the backtracking algorithm as a subroutine.
- It can be applied to find ground states of the Ising model:

$$\min_{z\in\{\pm 1\}^n} \sum_{i<j} a_{ij} z_i z_j$$

- e.g. Sherrington-Kirkpatrick model $a_{ij} \sim N(0,1)$: runtime $O(2^{0.226n})$ or better, beating Grover search $O(2^{0.5n})$.

Other developments on quantum backtracking

Applications:

- lattice-based cryptography, e.g. [Alkim et al '16, del Pino et al '16]
- Travelling Salesman Problem [Moylett et al '17]
- exact satisfiability [Mandrà et al '16]
- constraint programming [Booth et al '21]

Experimental implementation (in simulation) for 2-colouring a graph with 4 vertices [Martiel and Remaud '19]

Quantum speedup of dynamic programming [Ambainis et al '19]

Dynamic programming is a very widely-used technique in classical algorithm design, where we solve an overall problem more efficiently by storing the answers to subproblems.

Some classical algorithms based on dynamic programming can be accelerated using quantum algorithms.

For example, the travelling salesman problem:

- We are given a graph $G = (V, E)$ on n vertices with weights (costs) $w(u, v)$ on each edge.
- We look for a tour of all vertices in G that travels by valid edges and minimises the total cost.

Trying each path in turn would give a complexity of $O(n!)$, which can be accelerated to $O(\sqrt{n!})$ using Grover's algorithm...
...but there is a better classical algorithm running in time $O(2^n)$, up to polynomial factors.

Quantum algorithm for TSP [Ambainis et al '19]

We use the following dynamic programming recurrence:

- For each subset S, let $f(S,u,v)$ be the length of the shortest path in the graph induced by S that: starts at u; finishes at v; visits all vertices in S exactly once.
- Then we can write

$$(\diamondsuit) \quad f(S,u,v) = \min_{t \in N(u) \cap S, t \neq v} w(u,t) + f(S \setminus \{u\}, t, v), \quad f(\{v\}, v, v) = 0$$

- Gives a $\sim O(2^n)$ time classical algorithm by computing and storing $f(S,u,v)$ "from the bottom up".
- But we can also write, for any k,

$$(\heartsuit) \quad f(S,u,v) = \min_{\substack{X \subseteq S, |X| = k \\ u \in X, v \notin X}} \min_{\substack{t \in X \\ t \neq u}} f(X,u,t) + f((S \setminus X) \cup \{t\}, t, v)$$

Quantum algorithm for TSP [Ambainis et al '19]

We can use this within the following quantum algorithm:

1. Calculate $f(S,u,v)$ for all $|S| \leqslant (1-\alpha)n/4$ classically using DP.
2. Use quantum minimum finding to compute

$$\min_{\substack{S\subseteq V\\|S|=n/2}} \min_{\substack{u,v\in S\\u\neq v}} f(S,u,v)+f((V\backslash S)\cup\{u,v\},v,u)$$

3. To compute the required $f(S,u,v)$ values:
    - For $|S|=n/2$: use quantum minimum finding within $(\heartsuit)$, choosing $k=n/4$
    - For $|S|=n/4$: use quantum minimum finding within $(\heartsuit)$, choosing $k=\alpha n/4$
    - For $|S|=\alpha n/4$ or $|S|=(1-\alpha)n/4$: use results of classical preprocessing

Overall complexity, up to polynomial factors, is

$$O\left(\binom{n}{\leqslant(1-\alpha)n/4}\right)+O\left(\sqrt{\binom{n}{n/2}\binom{n/2}{n/4}\binom{n/4}{\alpha n/4}}\right)=O(2^{0.79n})$$

if we choose α optimally (we achieve $O(2^{0.81n})$ even choosing $\alpha=0$).

The true complexity of quantum algorithms for combinatorial optimisation

Will these quantum algorithms yield a speedup in practice?

Although they achieve asymptotic speedups, these are "only" up to quadratic and may be washed out by overheads associated with slow and noisy quantum hardware.

We go through one example of this, for backtracking, where we [Campbell et al '19]:

1. applied the quantum backtracking algorithm to graph colouring.
2. optimised the time complexity (circuit depth) of the algorithm.
3. estimated the likely runtime when applied to random instances.
4. calculated the physical runtimes and other complexity measures, for various hardware parameter regimes.
5. compared against the likely performance of a leading classical algorithm (DSATUR).

Cost model

We work out the runtime and space usage of quantum algorithms based on the use of the surface code [Fowler et al ’12] for quantum error-correction.

We then convert this to real-world runtimes based on various regimes corresponding to different parameters for quantum-computing hardware:

Parameter	Realistic	Plausible	Optimistic
Measurement time	50ns	5ns	0.5ns
2-qubit gate time	30ns	3ns	0.3ns
Gate error rate	10^{-3}	10^{-4}	10^{-5}

“Realistic” is (approximately!) achievable today; other two columns represent order-of-magnitude improvements.

Summary of results: good and bad news

- In the most optimistic hardware parameter regime, we could see speedup factors of $> 10^4$ (compared with a standard desktop PC)
- This speedup gets substantially smaller when considering parameters corresponding to quantum hardware available today.
- If we additionally take into account the cost of classical error-correction processing, this speedup disappears.
- The number of physical qubits used is very large (e.g. $> 10^{12}$), almost all of which are used for fault-tolerance.
- This strongly motivates the design of improved fault-tolerance techniques!

Summary of results

	Realistic	Plausible	Optimistic
Max n	113	128	144
T-depth	1.70×10^{12}	1.53×10^{13}	1.62×10^{14}
T/Toffoli count	8.24×10^{17}	9.94×10^{18}	1.24×10^{20}
Factory qubits	6.29×10^{13}	9.26×10^{12}	3.59×10^{12}
Speedup factor	7.25×10^{0}	5.17×10^{2}	4.16×10^{4}

Table: Likely speedup factors for graph colouring via backtracking achievable in different regimes.

Complexity estimates for other algorithms and CSPs were obtained by [Sanders et al '20].

Cost of classical processing

For a true cost comparison, we should also take into account the cost of classical error-correction processing.

- We start with the runtimes for error-correction reported by [Delfosse and Nickerson '17].
- We then extrapolate this to more exotic hardware platforms (GPUs, ASICs).

N	Realistic	Plausible	Optimistic
10^{12}	4.17×10^{7}	4.30×10^{4}	9.15×10^{-1}
10^{16}	2.29×10^{12}	7.76×10^{8}	2.23×10^{4}
10^{20}	3.10×10^{16}	3.07×10^{13}	3.28×10^{8}

Table: Classical processing required to implement N Toffoli gates under different regimes. Measured in processor-days (where type of processor is CPU, GPU and ASIC respectively in realistic, plausible and optimistic regimes). Assumes that the speedup offered by GPUs and ASICs over CPUs is a factor of 100 and 10^6 respectively.

Conclusions

We might be able to achieve quite a fairly significant quantum speedup for common and practically combinatorial optimisation problems…

…but there are some major challenges to be addressed before this becomes realistic. Improved fault-tolerance techniques would make a big difference.

Thanks!