Quantum Speedups for Exponential-Time Dynamic Programming Algorithms

Andris Ambainis, Kaspars Balodis, Jānis Iraids,
Martins Kokainis, Krišjānis Prūsis, Jevgēnijs Vihrovs

Faculty of Computing, University of Latvia

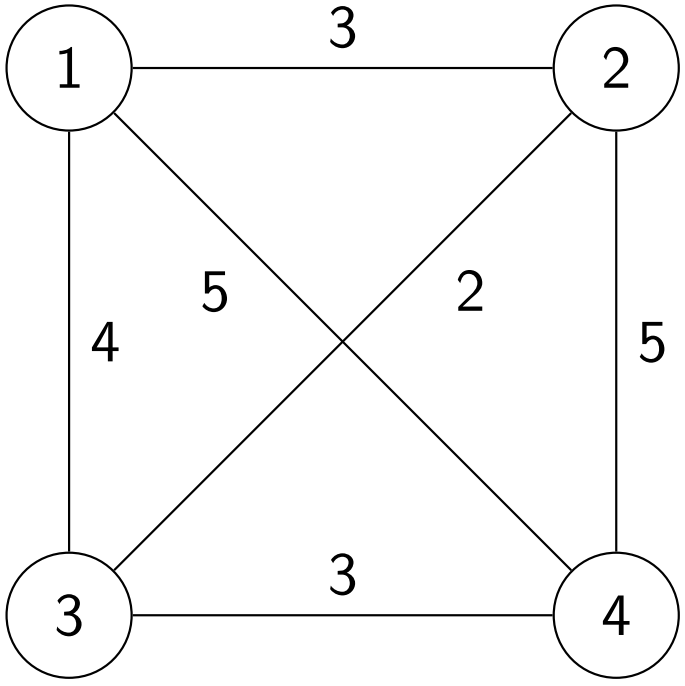
12th October, 2018 Joint Estonian-Latvian Theory Days

Quantum algorithms for NP-hard problems

- Algorithms for many NP-hard problems are widely used despite the exponential running time.
- Can we speed these algorithms up using quantum computation?

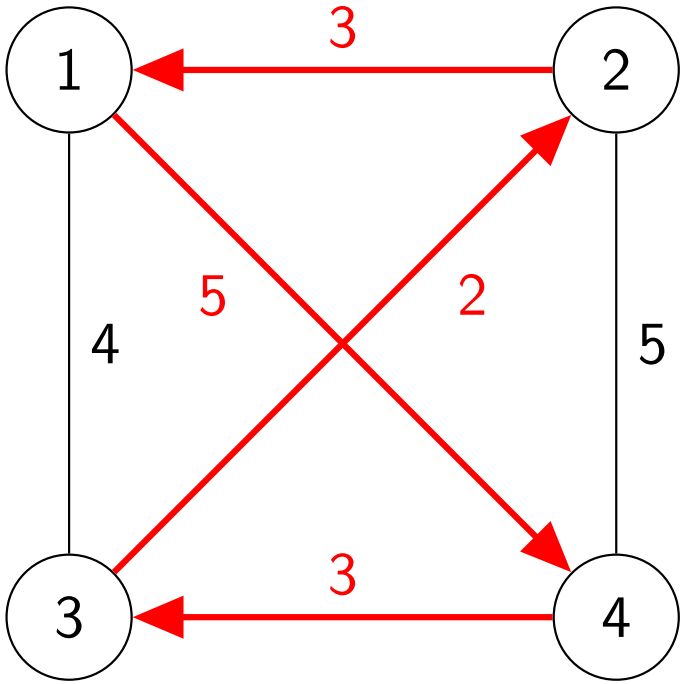
Example: the travelling salesman problem

- We are given a graph $G(V, E)$ of n cities, with edge weights denoting the travel time between each pair.



Example: the travelling salesman problem

- A travelling salesman wants to visit every city exactly once and return back to his starting position as fast as possible.



- Classically can be solved in $O^*(2^n)$.

Grover's search

- A quantum algorithm to find an answer from n independent procedures, each taking some time T, in time $O(T\sqrt{n})$.

Quantum speedup for SAT

- SAT over n variables is believed to not be solvable faster than $O^*(2^n)$ classically.
- Grover's search over all 2^n possible assignments gives an $O^*(\sqrt{2^n})$ quantum algorithm.
- Can we achieve a similar quadratic improvement for other NP-hard problems?

The naive algorithm for the TSP

- Try every possible order of cities.
- There are $(n-1)!$ possible orders, checking the length of a path takes $O(n)$ time.
- This gives a total complexity of

$$O(n \cdot (n-1)!) = O(n!).$$

Quantum version of the naive algorithm

- Perform Grover's search over every possible order of cities.
- This gives a quadratic improvement:

$$O\left(\sqrt{n!}\right).$$

The Bellman-Held-Karp algorithm

- A DP algorithm solving the TSP in $O^*(2^n)$ time.

- For each subset $S \subseteq V$ and pair $start, end \in S$, compute the shortest distance $D(S, start, end)$ from *start* visiting every city in S and ending in *end*.

Bellman-Held-Karp Algorithm dynamic programming

- $D(\{v\}, v, v) = 0$.

- $D(S, \text{start}, \text{end}) = \min_{\substack{\text{prev} \in S \\ \text{prev} \neq \text{end}}} \{D(S \setminus \{\text{end}\}, \text{start}, \text{prev}) + w(\text{prev}, \text{end})\}$.

- Answer: $\min_{\text{start}, \text{end} \in V} \{D(V, \text{start}, \text{end}) + w(\text{end}, \text{start})\}$.

Bellman-Held-Karp algorithm complexity

- There are 2^n subsets.
- Each subset has at most n choices of *start*, *prev*, *end*.
- This gives a complexity of

$$O(n^3 \cdot 2^n) = O^*(2^n).$$

Quantum algorithm outline

- Precompute $D(S, \text{start}, \text{end})$ for small sets S.
- Use Grover's search to find the optimal way to to combine these small paths into a cycle.

Classical precomputation

- Computing $D(S, \text{start}, \text{end})$ for every $S : |S| \leq t \cdot n$ and every *start*, *end* takes time

$$O^*\left(\sum_{k=0}^{k=t\cdot n} \binom{n}{k}\right) = O^*\left(2^{H(t)n}\right)$$

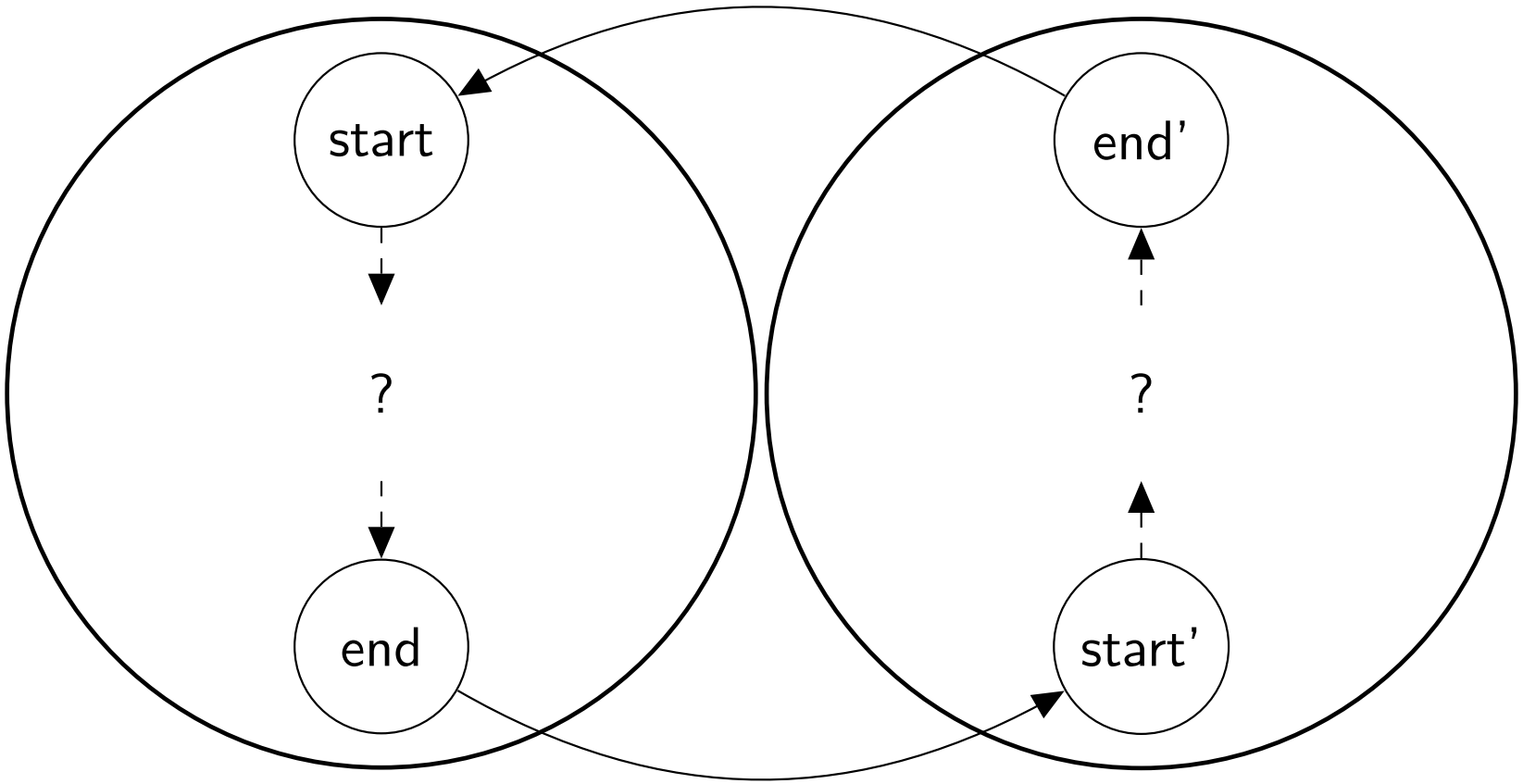
  where H is the binary entropy function.

- Our algorithm starts by doing this for sets of size up to $0.24n$, with a time complexity of

$$O^*\left(2^{H(0.24)n}\right) \approx O^*(1.73^n).$$

City set splits

- We can split the set into halves of size $n/2$ in $\binom{n}{n/2}$ ways.
- For each split, also choose the start and end vertices, giving only an n^4 factor.



City set splits

- Now we find paths in each half separately.
- We can split these further into parts of size $n/4$.
- Finally, we split those into parts of size $0.24n$ and $0.01n$, where we have already computed the optimal paths.

Quantum search over splits

- Classically searching over these splits takes time

$$O^*\left(\binom{n}{n/2}\binom{n/2}{n/4}\binom{n/4}{0.24n}\right).$$

- Grover's search takes time

$$O^*\left(\sqrt{\binom{n}{n/2}\binom{n/2}{n/4}\binom{n/4}{0.24n}}\right) \approx O^*(1.73^n).$$

Overall complexity

- the value 0.24 balances the O^* complexity of the classical precompution and the quantum search.

- This gives an $O^*(1.73^n)$ quantum algorithm for the TSP, an improvement over the $O^*(2^n)$ classical one.

Generalization

- This algorithm works for optimization problems OPT with the following property:

$$\mathrm{OPT}(S) = \min_{\substack{T \subseteq S \\ |T|=k}} \{\mathrm{OPT}(T) + \mathrm{OPT}(S \setminus T) + f(S, T)\}.$$

- This includes Feedback Arc Set and, with some modifications, Minimum Set Cover.

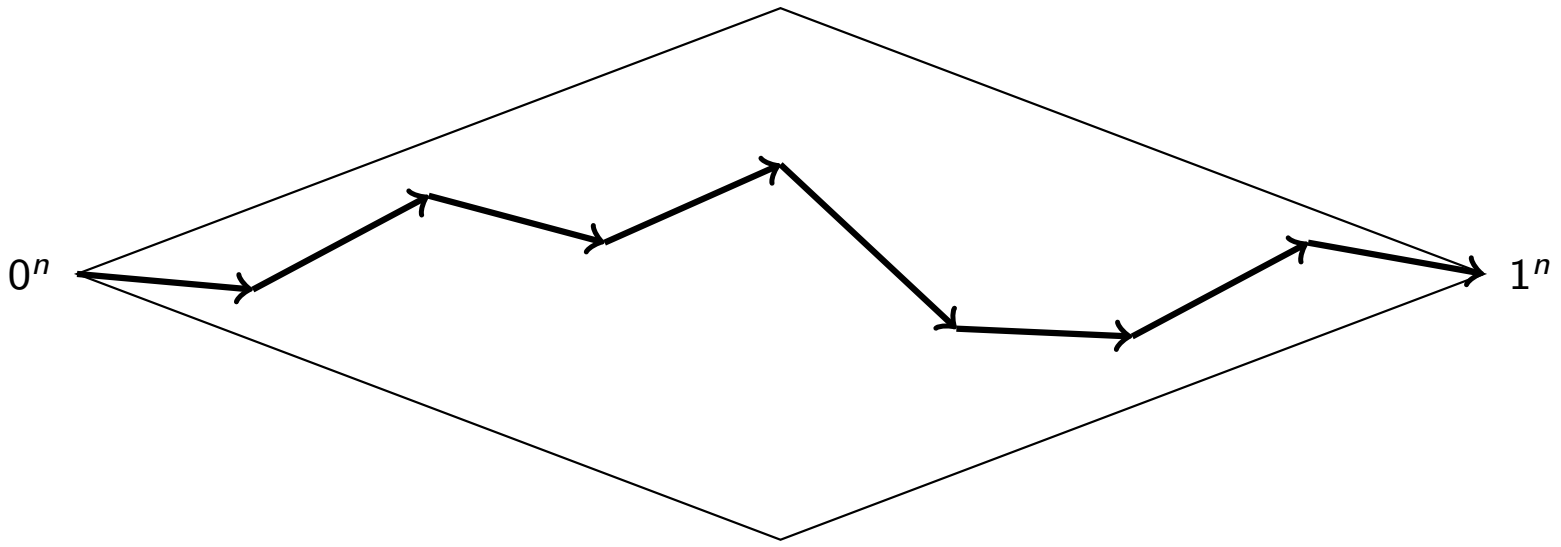
BHK algorithm generalization

- The Bellman-Held-Karp algorithm does not require such splits.

- For it a weaker property is sufficient:

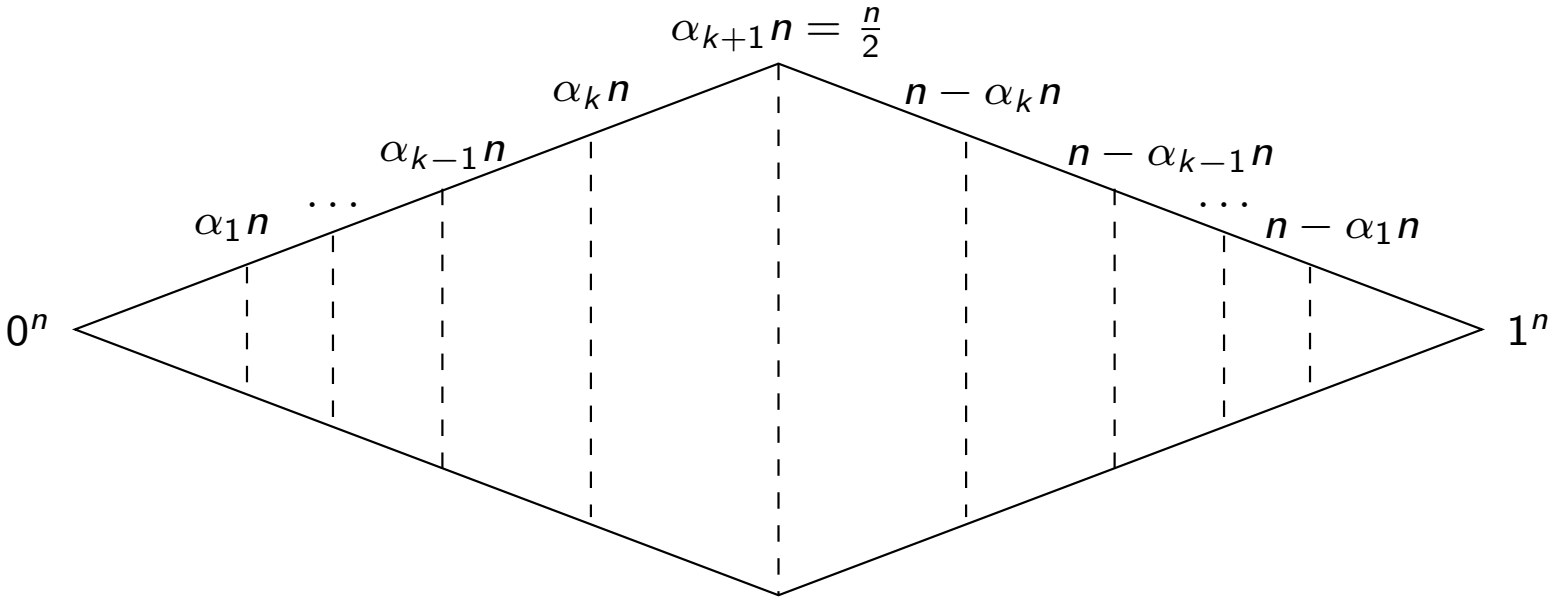
$$\mathrm{OPT}(S) = \min_{v \in S} \{\mathrm{OPT}(S \setminus \{v\}) + f(S, v)\}.$$

Path in the Hypercube



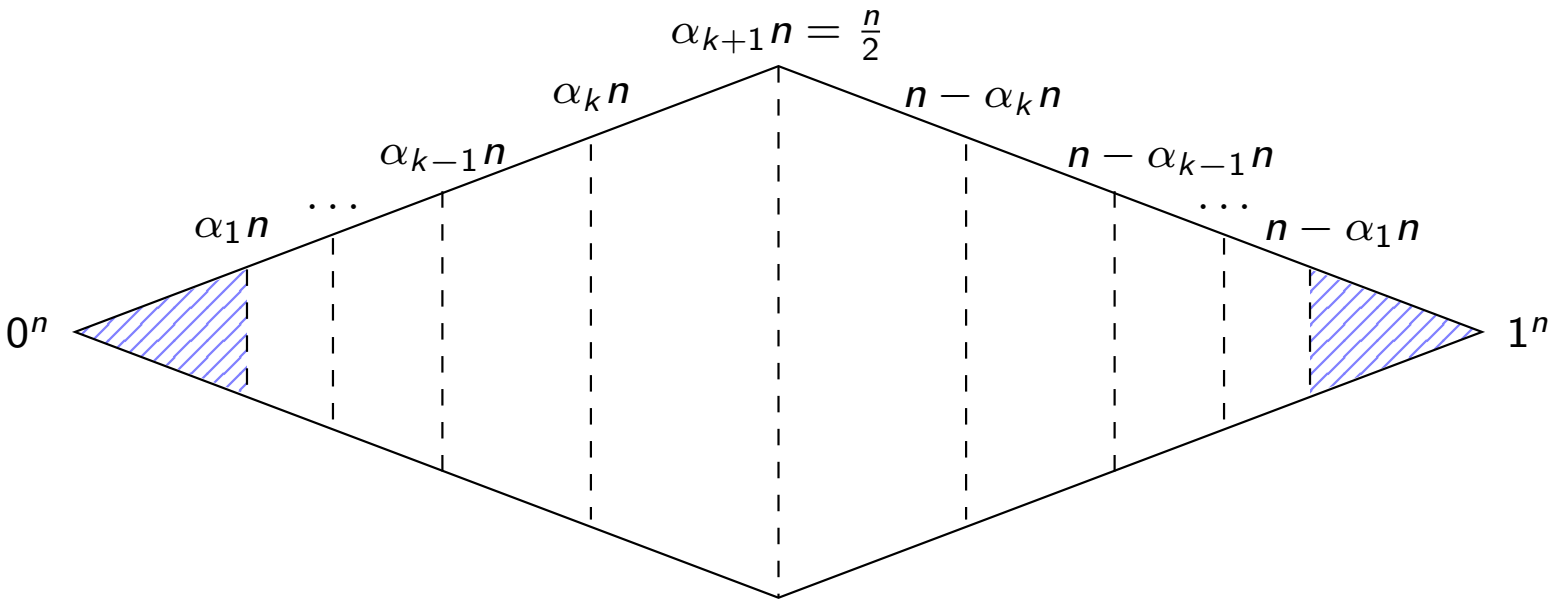
- Boolean hypercube $\{0,1\}^n$, edges $x \to y$ if $|x| + 1 = |y|$.
- Input: a subgraph G.
- Output: is there a path from 0^n to 1^n in G?

Layers



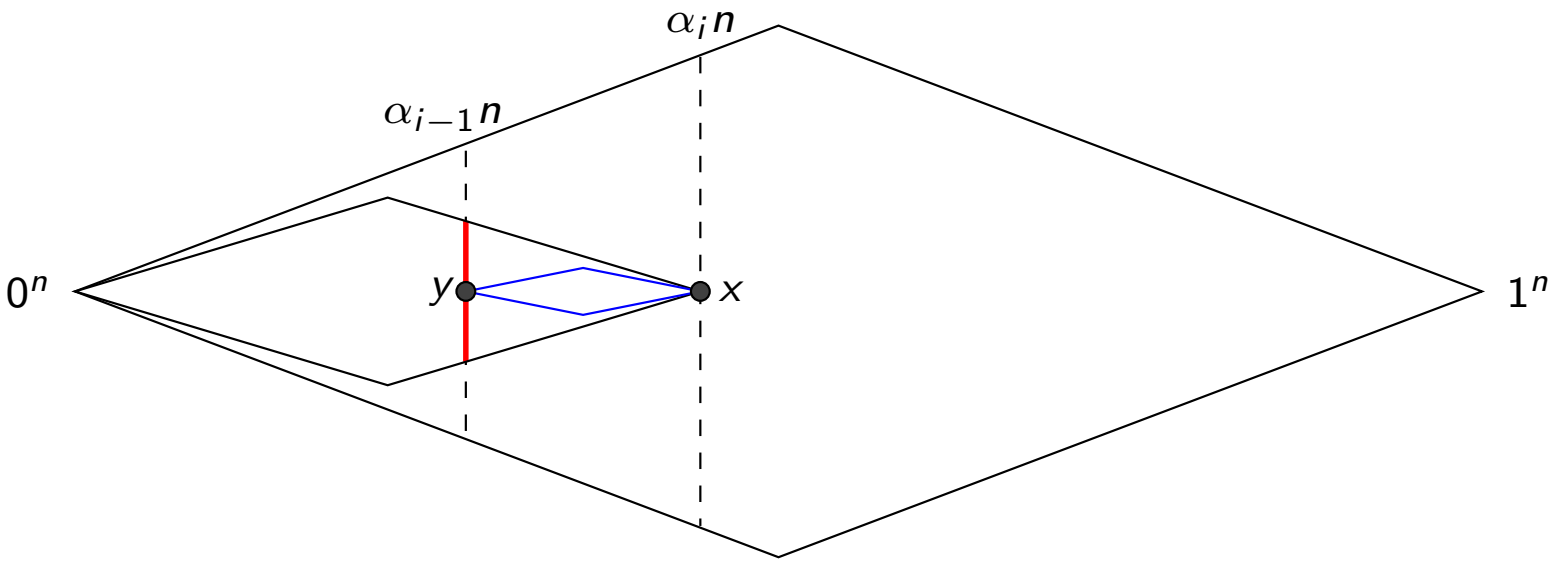
- Select a constant number of layers with a fixed Hamming weight.

Classical precomputation



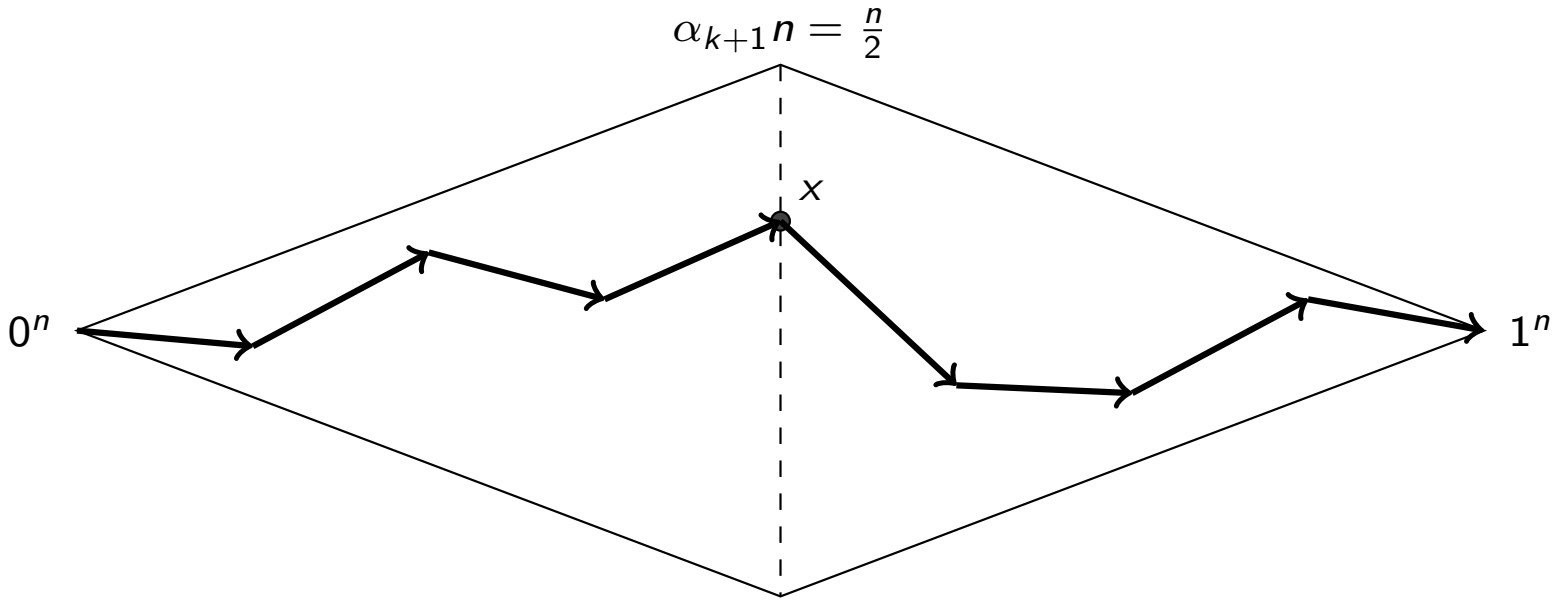
- Classically precompute which vertices are reachable from 0^n up to the first layer.
- Symmetrically find paths from the last layer to 1^n.

Quantum Search



- To find whether a vertex x in a further layer is reachable, perform Grover's search over vertices of the previous layer.
- Find each such vertex y, find a path between y and x recursively.

Overall Complexity



- Perform Grover's search to find a vertex x in the middle layer reachable from both sides.
- The overall complexity is $O^*(1.817^n)$.

Applications

- Vertex ordering problems:
  Treewidth*, Minimum Fill-In*, Pathwidth, Sum Cut, Minimum Interval Graph Completion, Cutwidth and Optimal Linear Arrangement.

- With modifications to take advantage of knowing the number of edges in G, Graph Bandwidth.

Summary

	Classical (best known)	Quantum (this work)
Travelling Salesman Problem	$O(n^2 2^n)$	$O^*(1.728^n)$
Feedback Arc Set	$O^*(2^n)$	$O^*(1.728^n)$
Minimum Set Cover	$O(nm2^n)$	$O(\text{poly}(m,n)1.728^n)$
Path in the Hypercube	$O(n2^n)$	$O^*(1.817^n)$
Vertex Ordering Problems	$O^*(2^n)$	$O^*(1.817^n)$
Graph Bandwidth	$O^*(4.383^n)$	$O^*(2.946^n)$

Open questions

- Quadratic speedup?
- Lower bounds?
- Subexponential memory?

Thank you!