Quantum Speedup of the Travelling Salesman Problem for Bounded Degree Graphs

Dominic J. Moylett[1]

School of Physics and Department of Electrical and Electronic Engineering
Quantum Engineering Centre for Doctoral Training
University of Bristol

dominic.moylett@bristol.ac.uk

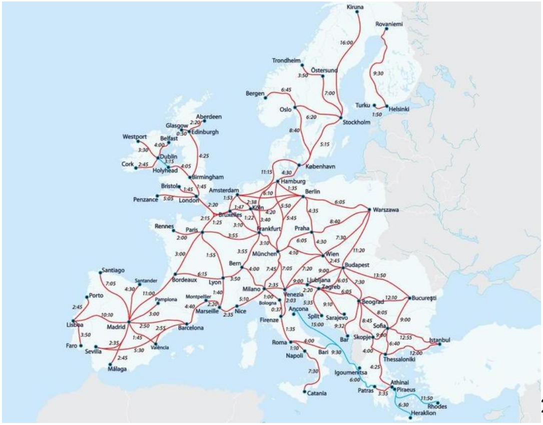
October 21, 2016

[1]Work completed in collaboration with Ashley Montanaro and Noah Linden.

Based on a True Story

I graduated in June 2015.
To celebrate, I wanted to do a tour of Europe.

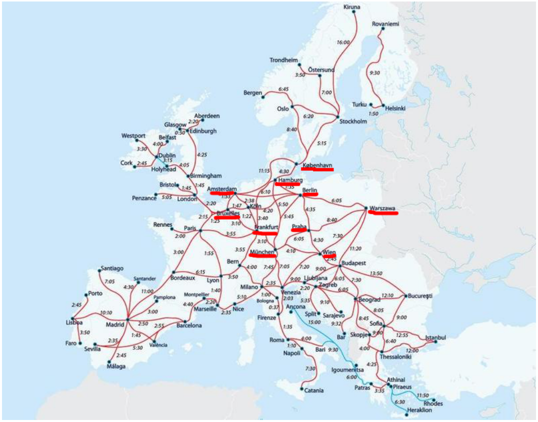
Interrail



[2]

[2] http://www.interrail.eu/plan-your-trip/interrail-railway-map

The Grand Tour



What's the fastest way of visiting every location?

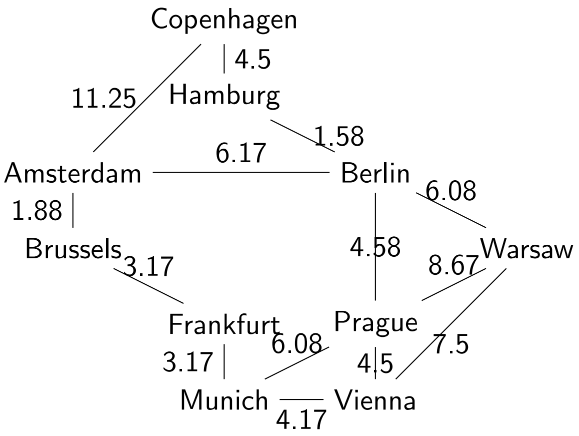
The Travelling Salesman Problem

Given a graph $G = (V, E)$ with cost matrix C, what is the Hamiltonian cycle with the lowest cost?

While route finding is the oldest instance of the problem, there are a number of other applications, such as circuit board drilling[3].

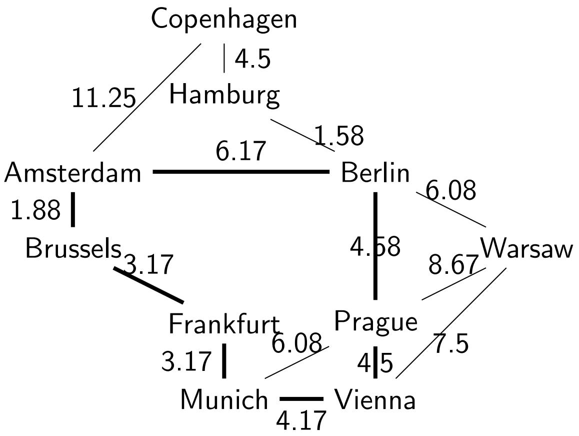
[3] *Zeitschrift für Operations Research*, **35**(61) (1991)

Example



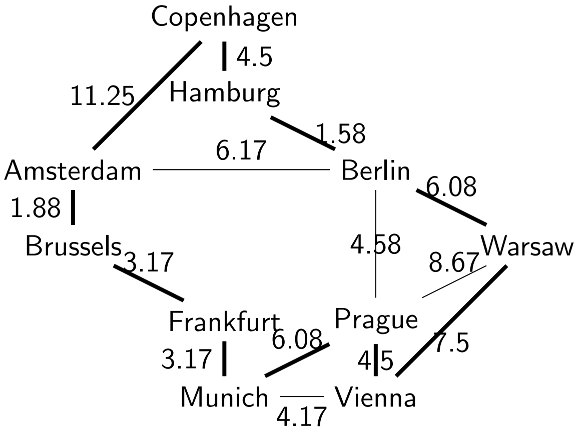
Our example graph.

Example



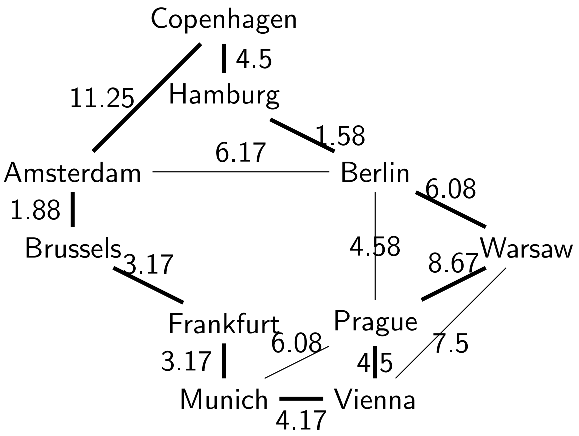
✗ Not a Hamiltonian cycle.

Example



✗ Not the shortest Hamiltonian cycle.

Example



✓ Our solution to the Travelling Salesman Problem.

Difficulty of the Travelling Salesman Problem

The Travelling Salesman Problem is *NP*-hard, so finding a polynomial time solution would prove that $P = NP$.

A naïve solution would take $O(|V|!)$ time by searching over every permutation of the vertices.

Dürr and Høyer's quantum algorithm for minimum finding[4] could be applied to this approach to achieve $O(\sqrt{|V|!})$ time.

But other more efficient classical algorithms exist. Can these be sped up using quantum algorithms?

[4] *arXiv:quant-ph/9607014*

Our Results

- A quadratic speedup for the Travelling Salesman Problem when the degree of any vertex in the graph is at most 3.
- This is using a classical algorithm by Eppstein[5], combined with a quantum speedup by Montanaro[6].
- Speedups for graphs of higher bounded degrees are found by reducing to this case.

[5] *Journal of Graph Algorithms and Applications*, **11**(1), pp. 61–81 (2007)
[6] *arXiv:1509.02374*

The Difficulty of a Quantum Speedup

Key components for many algorithms have limited quantum speedup:

- The Held-Karp algorithm[7] relies on dynamic programming (no known speedup).
- Christofides' algorithm[8] relies on finding a minimum weight perfect matching (polynomial speedup for bipartite graphs).
- Algorithms which use Nearest Neighbour can be sped up from $O(|V|^2)$ to $O(|V|^{3/2})$ time via minimum finding.

Other algorithms do not even have known performance[9].

[7] *Journal of the Society for Industrial and Applied Mathematics*, **10**(1), pp. 196-210 (1962)

[8] *Management Sciences Research Report 388, Graduate School of Industrial Administration, CMU* (1976)

[9] *Science*, **220**(4598), pp. 671–680 (1983)

Eppstein's Algorithm

Solves the Travelling Salesman Problem when the degree of any vertex is at most 3.

It takes as input a graph $G = (V, E)$ and a set $F \subseteq E$ of "forced" edges and returns the shortest Hamiltonian cycle which includes every edge in F.

Main steps:

1. Reduce (G, F) to (G', F') by changing specific subgraphs in G.
2. If F' contains a Hamiltonian cycle then return F'.
3. If a Hamiltonian cycle cannot be made, then abort.
4. If $G \setminus F$ is a collection of disjoint cycles of length four then solve and return.
5. Pick an edge $xy \in E' \setminus F'$ according to specific rules.
6. Call Eppstein's algorithm on $(G', F' \cup \{xy\})$ and $(G' \setminus \{xy\}, F')$.
7. Return the shortest Hamiltonian cycle or abort if no cycle is found.

Edge Picking

The main bottleneck of Eppstein's algorithm is the recursive calls. So how is the edge to force or remove chosen?

1. If $G' \setminus F'$ has a cycle of four vertices such that two of the vertices are incident to forced edges, pick one vertex x in the cycle that is not incident to a forced edge and another vertex y not in the cycle.
2. Otherwise if $F' \neq \emptyset$, pick an edge $(y, z) \in F$ and then chooses an adjacent edge $(x, y) \notin F$ as long as (x, y) is not part of a disjoint cycle of four vertices in $G \setminus F$.
3. Otherwise pick any edge.

Performance

Eppstein showed that these branching rules broke the problem size into one of two cases:

1. A subproblem with two more forced edges and one more cycle of 4 unforced edges, and a subproblem with three fewer vertices.
2. Two subproblems with at least three more forced edges.
3. A subproblem with at least two more forced edges, and a subproblem with at least five more forced edges.

Linear recurrence solving shows that the overall runtime is $O(2^{n/3})$.

Why not use minimum finding search?

Dürr and Høyer's algorithm for minimum finding can be used if we can map a binary string to each result of the algorithm.

We can map binary strings to Eppstein's algorithm by removing an edge if the next bit is 0 and forcing it if the next bit is 1.

However, the longest bit string we might need is $n/2$ bits, because of the case which only adds two forced edges to the graph.

Thus we have a runtime of $O(2^{n/4})$.

We need to take advantage of the problem's structure if we want to do even better.

Backtracking Algorithms

Backtracking algorithms are algorithms to solve constraint satisfaction problems, where we have n variables to assign values to such that they satisfy m constraints.

They work by using a predicate P and heuristic h as follows, given a partial assignment V:

```
1 If $P(V) = \top$ then return $V$.
2 Else if $P(V) = \bot$ then abort.
3 Let $v_i \leftarrow h(V)$.
4 For all possible values $a$ that we can assign to $v_i$:
    1 Call recursively on $(V \cup (v_i, a))$ to get assignment $V'$.
    2 If call does not abort then return assignment $V'$.
5 Abort.
```

Quantum Speedup for Backtracking Algorithms

Montanaro proved the following:

Theorem (Montanaro)

Let T be the number of recursive calls made by a backtracking algorithm. For any $0 < \delta < 1$, there is a quantum algorithm which evaluates P and h $O(\sqrt{T} n^{3/2} \log(n) \log(1/\delta))$ times, and outputs a partial assignment V such that $P(V)$ is true or aborts if no such V exists. The algorithm fails with probability δ.

This speedup works by visualising the backtracking algorithm's calls as a tree and performing a quantum walk on the tree.

Applying Montanaro's Speedup to Eppstein's Algorithm

Eppstein's algorithm is an example of a backtracking algorithm. The variables are the edges in the graph, of which there are at most $3|V|/2$, and the values we can assign to them are whether an edge is forced or removed.

To fit the framework used by Montanaro's quantum algorithm, we need to rewrite Eppstein's algorithm in terms of the predicate and heuristic functions.

The Predicate

1. Apply any already known assignments to the graph.
2. Reduce (G, F) to (G', F') by changing specific subgraphs in G.
3. If F' contains a Hamiltonian cycle then return $\top$.
4. If a Hamiltonian cycle cannot be made, then return $\perp$.
5. If $G \setminus F$ is a collection of disjoint cycles of length four then return $\top$.
6. Otherwise return ?

The Heuristic

1. Apply any already known assignments to the graph.
2. Reduce (G, F) to (G', F') by changing specific subgraphs in G.
3. Pick an edge $xy \in E' \setminus F'$ according to specific rules.
4. If xy is made up of several edges in E then return one of the edges in xy.
5. Otherwise return xy.

Finding the Shortest Hamiltonian Cycle

The algorithm as it is currently described will return a randomly chosen Hamiltonian cycle with probability δ in $O(2^{|V|/6}\,\mathrm{poly}(|V|)\log(1/\delta))$ time. However, this is not guaranteed to be the shortest Hamiltonian cycle.

To find the shortest Hamiltonian cycle, we will repeat the process, but with the extra condition that the predicate returns $\perp$ if the combined cost of all the cycles in F' is above a threashold T. We then use binary search to find the shortest Hamiltonian cycle after $O(\log N)$ calls of the algorithm, where

$$N = \sum_{i=0}^{n-1} \max_{j=0}^{n-1} \{c_{i,j}\}$$

is an upper bound for the length of any Hamiltonian cycle.

Quantum Speedup on Cubic Graphs

Theorem

There is a quantum algorithm which solves the TSP for graphs of bounded degree 3 with probability δ in time $O(\log N 2^{n/6} \operatorname{poly}(n) \log(\log N/\delta))$.

Beyond Cubic Graphs

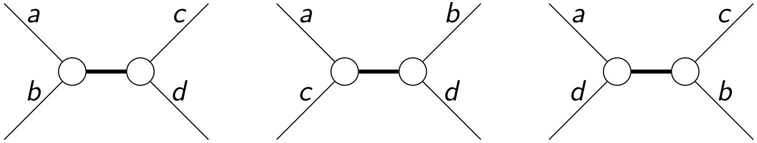
After achieving a speedup for cubic graphs, we looked at whether or not graphs of higher degree could also be sped up.

This is done through two components:

1. Reducing the graph to instances of lower degree graphs.
2. Speeding up the search over these lower degree instances via Dürr and Høyer's algorithm for minimum finding.

Degree 4 Vertices

There are three ways we can split a degree 4 vertex into two vertices of degree 3 connected by a forced edge:



If a and b are the two edges which are part of the shortest Hamiltonian cycle, there are two ways of splitting the vertex which preserve the answer.

Bounded Degree 4 Graphs

Let k be the number of degree 4 vertices in the graph.

There are 3^k ways of reducing the graph to a degree 3 graph which we can solve, of which 2^k will have the shortest Hamiltonian cycle.

Note that this splitting does increase the number of vertices, but it also increases the number of forced edges, so the problem size as a whole does not increase.

By applying Dürr and Høyer's algorithm for minimum finding, we can find the shortest Hamiltonian cycle after $(3/2)^{k/2}$ repeated calls. Thus our runtime is

$$O\left(\left(\frac{3}{2}\right)^{\frac{k}{2}} \log N 2^{\frac{n}{3}} \operatorname{poly}(n) \log\left(\frac{\log N}{\delta}\right)\right).$$

Bounded Degree 4 Graphs

Since $k \leq n$, we can bound this and work out the runtime to be

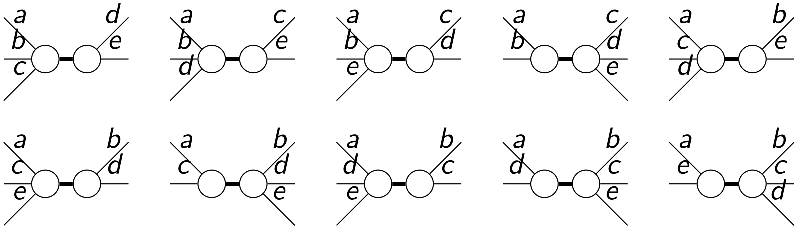
$$O(1.375^n \log N \operatorname{poly}(n) \log(\log N/\delta)).$$

This beats the best known classical runtime of $O(1.692^n)$ by Xiao and Nagamochi[10].

[10] *Theory of Computing Systems*, **58**(2), pp. 241–272 (2016)

Degree 5 Vertices

There are ten ways we can split a degree 5 vertex into a vertex of degree 4 and another of degree 3 connected by a forced edge:



If *a* and *b* are the two edges which are part of the shortest Hamiltonian cycle, there are six ways of splitting the vertex which preserve the answer.

Bounded Degree 5 Graphs

There are 10^k ways of reducing the graph to a bounded degree 4 graph, of which 6^k will have the shortest Hamiltonian cycle.

By using the same techniques as before, we get a runtime of

$$O\left(\left(\frac{10}{6}\right)^{\frac{k}{2}} 1.375^n \log N \operatorname{poly}(n) \log\left(\frac{\log N}{\delta}\right)\right)$$

.

Bounded Degree 5 Graphs

Since $k \leq n$, we can bound this and work out the runtime to be

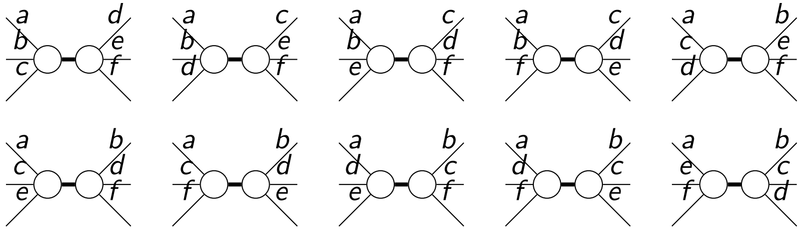
$$O(1.775^n \log N \operatorname{poly}(n) \log(\log N/\delta)).$$

This beats the best known classical runtime by Björklund et al. of $O(1.932^n \operatorname{poly}(n))$[11].

[11] *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, pp. 198–209 (2008)

Degree 6 Vertices

There are ten ways we can split a degree 6 vertex into two vertices of degree 4 connected by a forced edge:



If *a* and *b* are the two edges which are part of the shortest Hamiltonian cycle, there are six ways of splitting the vertex which preserve the answer.

Bounded Degree 6 Graphs

As we have the same splitting results as we had for bounded degree 5, we have the same runtime of $O(1.775^n \log N \operatorname{poly}(n) \log(\log N/\delta))$.

This beats the best known classical time by Björklund et al. of $O(1.967^n \operatorname{poly}(n))$.

Bounded Degree 7

There are $\binom{7}{4} = 35$ ways of splitting a vertex of degree 7 into one vertex of degree 5 and another of degree 4 connected by a forced edge. Of these, $35 - \binom{4}{2} - \binom{5}{2} = 26$ will succeed. Thus we achieve a runtime of

$$\begin{aligned} &O\left(\left(\frac{35}{26}\right)^{k/2} 1.768^n \log N \operatorname{poly}(n) \log\left(\frac{1}{\delta}\right)\right) \\ =&O(2.0513^n \log N \operatorname{poly}(n) \log(1/\delta)). \end{aligned}$$

Thus we are no longer beating Björklund et al., which has a runtime of $O(1.9840^n \operatorname{poly}(n))$.

Faster Algorithms

Eppstein's algorithm was the first backtracking algorithm to solve the Travelling Salesman Problem. But other backtracking algorithms have faster runtimes:

- Iwama and Nakashima[12] have an algorithm for bounded degree 3 which runs in $O(1.251^n)$ time.
- Liśkiewicz and Schuster[13] have an algorithm for bounded degree 3 which runs in $O(1.2553^n)$ time and challenge the result of Iwama and Nakashima.
- Xiao and Nagamochi[14][15] have given runtimes of $O(1.2312^n)$ and $O(1.692^n)$ for bounded degrees 3 and 4, respectively.

[12] 13th Annual International Conference on Computing and Combinatorics, pp. 108–117 (2007)

[13] *Journal of Discrete Algorithms*, **27**, pp. 1–20 (2014)

[14] *Algorithmica*, **74**(2), pp. 713–741 (2016)

[15] *Theory of Computing Systems*, **58**(2), pp. 241–272 (2016)

Conclusion

The Travelling Salesman Problem is one of the most famous problems in all of Computer Science.

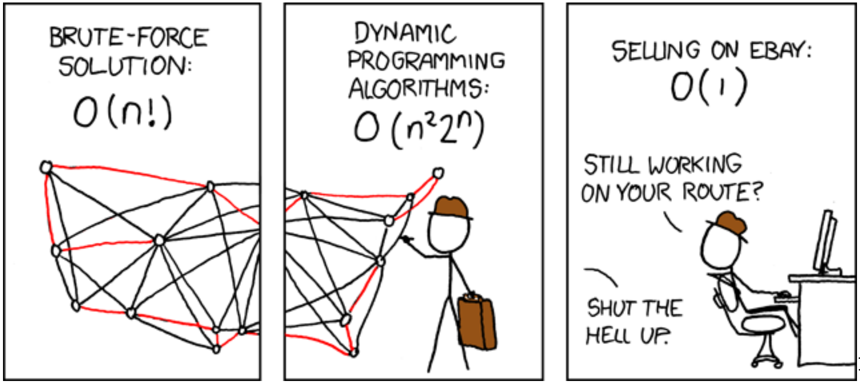
We have demonstrated quantum speedups for the Travelling Salesman Problem for graphs where the degree of any vertex is at most 6.

Future Steps:

- Apply further analysis to algorithms by Xiao and Nagamochi to see if we can yield an even better quantum speedup.
- ???
- Publish!

The End

Any questions?


BRUTE-FORCE SOLUTION: O(n!)
DYNAMIC PROGRAMMING ALGORITHMS: O(n²2ⁿ)
SELLING ON EBAY: O(1)
STILL WORKING ON YOUR ROUTE?
SHUT THE HELL UP.
[16]

[16] https://xkcd.com/399/

Post-credits

This is not the slide you're looking for.