CS 163 & CS 265: Graph Algorithms

Week 5: More paths

Lecture 5b: Exact TSP algorithm

David Eppstein
University of California, Irvine

Winter Quarter, 2025

Overview

Reminder: The traveling salesperson problem

Today, the distance matrix version is the more convenient one

Preprocessing

- Number the vertices as $v_0, v_1, \ldots, v_{n-1}$
- Compute all-pairs shortest paths

The main problem

- Input: Matrix $D[i, j]$ of distances
- Output: List X of vertex numbers (permutation of $0, \ldots n-1$)
- Distance between consecutive vertices is $D[X[i-1], X[i]]$
  (for $i = 0$ use Python convention that $X[-1]$ is last entry)
- Goal: Minimize total length of tour $\sum D[X[i-1], X[i]]$

Hardness

Because it's NP-hard, no polynomial-time algorithm is known that will always find the optimal solution, and it is believed that no such algorithm exists

But there are algorithms! (Just not polynomial time ones)

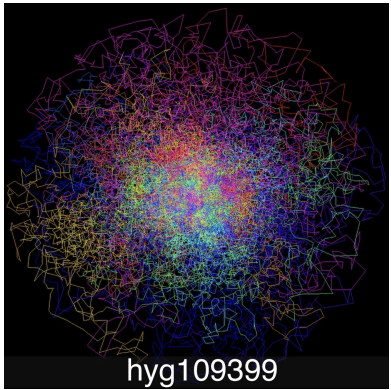
Most obviously: Try all permutations and return the best one

Time $O(n!)$ (with some care in generating permutations so we can construct each one and find its length in constant time)

This doesn't allow n to be very large (maybe 15?)

Plenty of room for improvement between factorial and polynomial

Typical instances are much easier than worst case



Optimal tour of over 100,000 nearby stars computed by the Concorde TSP solver (`http://www.math.uwaterloo.ca/tsp/star/hyg.html`)

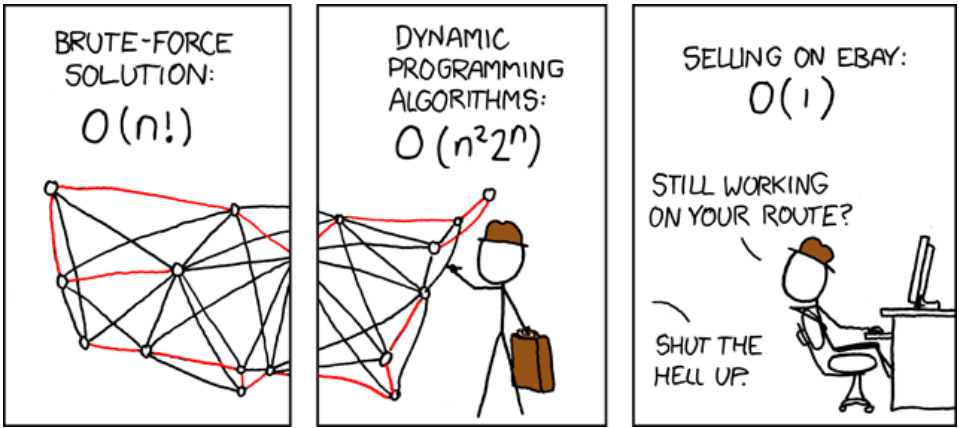
Moderately large problems can be solved optimally

Main ideas:

- Backtracking search
- Use linear programming to bound quality each branch can hope to reach
- Stop early when bound is worse than best tour found so far

No theoretical guarantees on runtime

But we can also much better in worst case



https://xkcd.com/399/

Today's main topic: the $O(n^2 2^n)$ algorithm

Dynamic programming algorithm

Main algorithm outline

- Preprocessing (all pairs shortest paths to construct distance matrix)
- Choose an arbitrary starting vertex s (doesn't matter which one)
- Compute, for every subset A of remaining vertices, and every $v \in A$, the best path from s to v covering all vertices in A
- Choose v such that path through all vertices to v + one more step from v back to s has minimum total length

[Bellman 1962; Held and Karp 1962]

Numbering sets

Remember that we numbered the vertices as $v_0, v_1, \ldots v_{n-1}$?

That wasn't just to use them as indices into distance array $D[i, j]$

Represent set $\{v_i, v_j, v_k, \ldots\}$ as the number $2^i + 2^j + 2^k + \ldots$

Set operations take time $O(1)$ (for # vertices smaller than # bits in a machine word, true for graphs small enough to use this algorithm) using bitwise operations:

$\{v_i\}$	$\Longleftrightarrow$	`1 << i`	(singleton set)
$A \cap B$	$\Longleftrightarrow$	`A & B`	(intersection)
$A \cup B$	$\Longleftrightarrow$	`A | B`	(union)
$A + v_i$	$\Longleftrightarrow$	`A | (1<<i)`	(add element)
$A - v_i$	$\Longleftrightarrow$	`A & ~(1<<i)`	(remove element)
$v_i \in A$	$\Longleftrightarrow$	`(1<<i)&A != 0`	(membership test)
$V(G)$	$\Longleftrightarrow$	`(1<<n) - 1`	(set of all vertices)

Subproblems for dynamic program

Let's define another array $L[A, v]$:

The length of the shortest tour starting at s, ending at v,
covering all vertices in set A

Computing this is the missing step in the main algorithm outline

If we knew second-to-last vertex u in the shortest tour, we could compute $L[A, v]$ easily from value of L at a smaller set:

$$L[A, v] = L[A - v, u] + D[u, v]$$

Other vertices than u give longer tours (bigger values for the same formula), so we can find the shortest one by minimizing:

$$L[A, v] = \min_{u \in A - v} L[A - v, u] + D[u, v]$$

So loop over all sets using this formula to fill in table L

A minor optimization

We can choose our starting vertex s arbitrarily

Let's choose $s = v_{n-1}$, the last one in the numbering

That way, the sets not containing s are numbered $0, 1, 2, \ldots 2^{n-1} - 1$

The nonempty ones are numbered $1, 2, \ldots 2^{n-1} - 1$

and remember the computer code for $2^{n-1} - 1$ is `(1<<(n-1))-1`

Pseudocode for shortest tour length

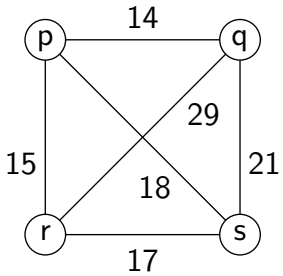
number vertices and use all-pairs shortest paths to compute D

for A in $1, 2, \ldots 2^{n-1} - 1$: (nonempty subsets of $V(G) - s$)
    for v in A:
        if $A - v$ is empty: (base case)
            $L[A, v] = D[n-1, v]$
        else: (use the formula)
            $L[A, v] = \min\{L[A - v, u] + D[u, v] \text{ for } u \in A - v\}$

return $\min\{L[V(G) - s, v] + D[v, n-1] \text{ for } v \in V(G) - s\}$

The tour itself can be found by keeping track of which vertex u gives the minimum for each entry in L and backtracking

Example: Setup



L:

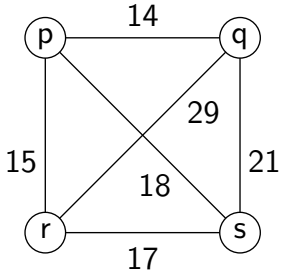
	p	q	r	
{p}				1
{q}				2
{p,q}				3
{r}				4
{p,r}				5
{q,r}				6
{p,q,r}				7
	0	1	2	

spqr = srqp = 18+14+29+17 = 78

sprq = sqrp = 18+15+29+21 = 83

sqpr = srpq = 21+14+15+17 = 67

Example: Base cases



L:

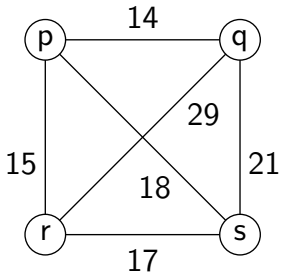
	p	q	r	
{p}	18	–	–	1
{q}	–	21	–	2
{p,q}				3
{r}	–	–	17	4
{p,r}				5
{q,r}				6
{p,q,r}				7
	0	1	2	

spqr = srqp = 18+14+29+17 = 78

sprq = sqrp = 18+15+29+21 = 83

sqpr = srpq = 21+14+15+17 = 67

Example: Two-element sets, no choice for u



L:

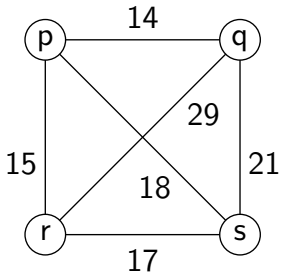
	p	q	r	
{p}	18	–	–	1
{q}	–	21	–	2
{p,q}	21+14			3
{r}	–	–	17	4
{p,r}				5
{q,r}				6
{p,q,r}				7
	0	1	2	

spqr = srqp = 18+14+29+17 = 78

sprq = sqrp = 18+15+29+21 = 83

sqpr = srpq = 21+14+15+17 = 67

Example: Two-element sets, filled in



L:

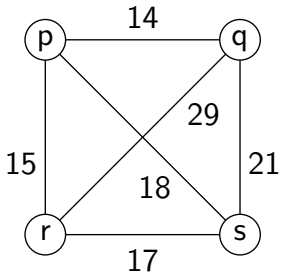
	p	q	r	
{p}	18	–	–	1
{q}	–	21	–	2
{p,q}	35	32	–	3
{r}	–	–	17	4
{p,r}	32	–	33	5
{q,r}	–	46	50	6
{p,q,r}				7
	0	1	2	

spqr = srqp = 18+14+29+17 = 78

sprq = sqrp = 18+15+29+21 = 83

sqpr = srpq = 21+14+15+17 = 67

Example: Final row, two choices for u



L:

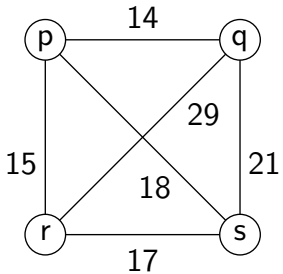
	p	q	r	
{p}	18	–	–	1
{q}	–	21	–	2
{p,q}	35	32	–	3
{r}	–	–	17	4
{p,r}	32	–	33	5
{q,r}	–	46	50	6
{p,q,r}	46+14 50+15			7
	0	1	2	

spqr = srqp = 18+14+29+17 = 78

sprq = sqrp = 18+15+29+21 = 83

sqpr = srpq = 21+14+15+17 = 67

Example: Final row, filled in



L:

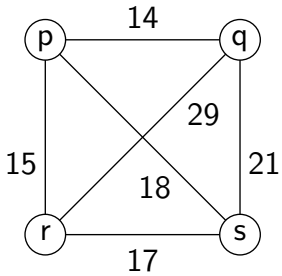
	p	q	r	
{p}	18	–	–	1
{q}	–	21	–	2
{p,q}	35	32	–	3
{r}	–	–	17	4
{p,r}	32	–	33	5
{q,r}	–	46	50	6
{p,q,r}	60	46	50	7
	0	1	2	

spqr = srqp = 18+14+29+17 = 78

sprq = sqrp = 18+15+29+21 = 83

sqpr = srpq = 21+14+15+17 = 67

Example: Return min (last row + return to s)



$spqr = srqp = 18+14+29+17 = 78$

$sprq = sqrp = 18+15+29+21 = 83$

$sqpr = srpq = 21+14+15+17 = 67$

L:

	p	q	r	
{p}	18	–	–	1
{q}	–	21	–	2
{p,q}	35	32	–	3
{r}	–	–	17	4
{p,r}	32	–	33	5
{q,r}	–	46	50	6
{p,q,r}	60	46	50	7
	0	1	2	

$60+18$ $46+21$ $50+17$

A new tiny improvement

From $O(2^n n^2)$

To $\frac{2^n n^2}{2^{\Omega(\sqrt{\log n})}}$

via (min,plus) matrix multiplication

[Stoian 2024]

The morals of the story

When polynomial time is out of reach, don't give up

Careful algorithm design (e.g. dynamic programming) can be significantly faster than brute force search

Maybe more important for these problems because they are more limited by algorithm speed than faster algorithms would be

Practical methods without guaranteed time bounds can be faster

Closing gap between theory and practice is a big open problem

References

Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9:61–63, 1962. doi:10.1145/321105.321111.

Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10: 196–210, 1962. doi:10.1137/0110015.

Mihail Stoian. TSP escapes the $O(2^n n^2)$ curse. Electronic preprint arxiv:2405.03018, 2024.