Intractable Problems I

Summer 2017 • Lecture 08/08

A Few Notes

Homework 6

- Due 8/13 at 11:59 p.m. on Gradescope.

Final Logistics

Final

Will occur on 8/18 at 8:30-11:30 a.m.

Alternate Final

Will occur on 8/17 at 3:30-6:30 p.m.

Exam special cases

Expect confirmation emails from me in the next day or two.

Practice exam

We'll release the practice exam tomorrow.

The Rest of the Quarter

- Lectures 1-11 covered the bulk of the material, congrats!
  - This material will be emphasized on the final (~90% of points).
- Lectures 12-14 will cover additional topics.
  - Intractable problems, approximation algorithms, amortized analysis

Outline for Today

Intractable Problems

- Background
- Traveling Salesman Problem
- 0/1 Knapsack, revisited

Background

Defining Efficiency

What is an efficient algorithm?

An algorithm is efficient iff it runs in polynomial time on a serial computer.

Runtimes of “efficient” algorithms: $O(n)$, $O(n\log(n))$, $O(n^8\log^4(n))$, $O(n^{1,000,000})$.

Runtimes of “inefficient” algorithms: $O(2^n)$, $O(n!)$, $O(1.0000001^n)$.

Some Caveats

Parallelism Some problems can be solved in polynomial time on machines with a polynomial number of processors.

Are all efficient algorithms parallelizable?

Randomization Some algorithms can be solved in expected polynomial time, or have poly-time Monte Carlo algorithms that work with high probability.

Are randomized efficient algorithms efficient solutions? P $\subseteq$? RP$\subseteq$? NP.

Quantum computation Some algorithms can be solved in polynomial time on a quantum computer.

Are quantum efficient algorithms efficient solutions?

These are all open problems!

Tractability

A problem is called **tractable** iff there is an efficient (i.e. polynomial time) algorithm that solves it.

A problem is called **intractable** iff there is no efficient algorithm that solves it.

NP

A **decision problem** is a problem with a yes/no answer.

The class **NP** consists of all decision problems where “yes” answers can be verified efficiently.

- Is the kth order statistic of A equal to x?
- Is there a cut in G of size at least k?

All tractable decision problems are in **NP**, plus a lot of problems whose difficulty is unknown.

NP-Completeness

The **NP-complete** problems are (intuitively) the hardest problems in NP.

Either every **NP**-complete problem is tractable or no **NP**-complete problem is tractable.

- This is an open problem: the P = NP question!

There are no known polynomial-time algorithms for any **NP**-complete problem.

NP-Hardness

A problem (which may or may not be a decision problem) is called **NP-hard** if (intuitively) it is at least as hard as every problem in **NP**.

As before: no polynomial-time algorithms are known for any **NP**-hard problem.

NP-Hardness

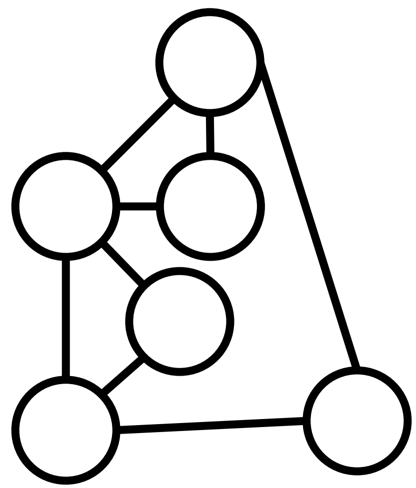
Assuming that P $\neq$ NP, all NP-hard problems are intractable.

- This does not mean that brute-force algorithms are the only option.
- This does not mean that it is hard to get approximate answers.

Traveling Salesperson Problem

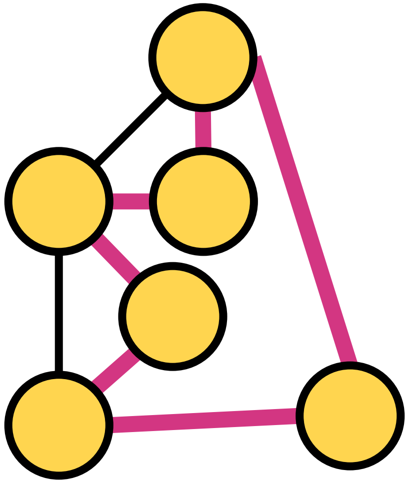
TSP

A **Hamiltonian cycle** in an undirected graph G is a simple cycle that visits every vertex in G.



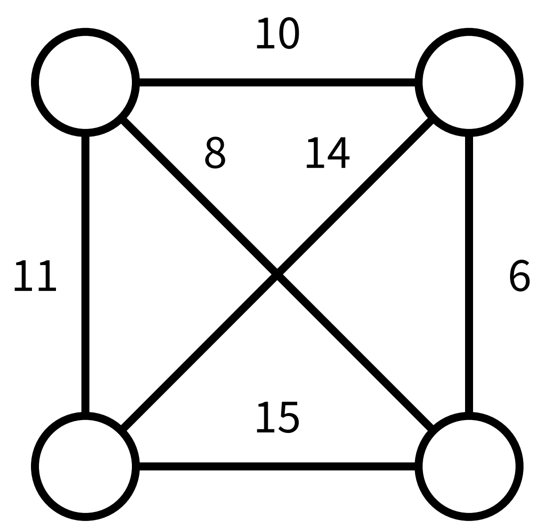
TSP

A **Hamiltonian cycle** in an undirected graph G is a simple cycle that visits every vertex in G.



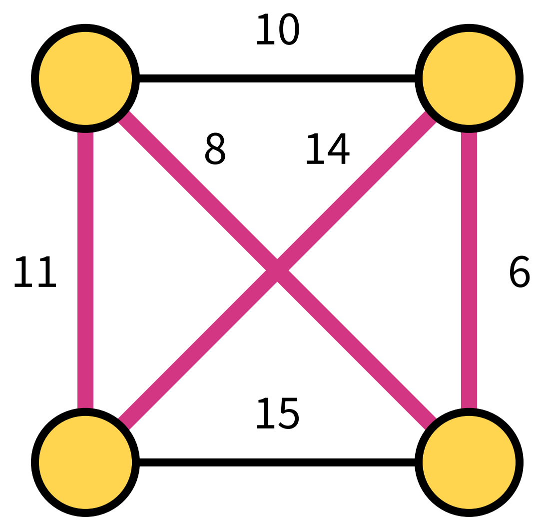
TSP

Given a complete, undirected weighted graph G, the traveling salesman problem is to find a Hamiltonian cycle in G of least total cost.



TSP

Given a complete, undirected weighted graph G, the traveling salesman problem is to find a Hamiltonian cycle in G of least total cost.



TSP

Formally Given a complete, undirected G and a set of positive integer edge weights, the TSP is to find a Hamiltonian cycle in G with least total weight.

Note that since G is complete, there must be at least one Hamiltonian cycle. The challenge is finding the cycle with least cost.

This problem is known to be **NP**-hard.

TSP

- Try all possible Hamiltonian cycles in the graph?
  - How many Hamiltonian cycles are there? $(n-1)! / 2$
  - Since each cycle takes $O(n)$-time, the total time is **`O(n!)`**.

TSP

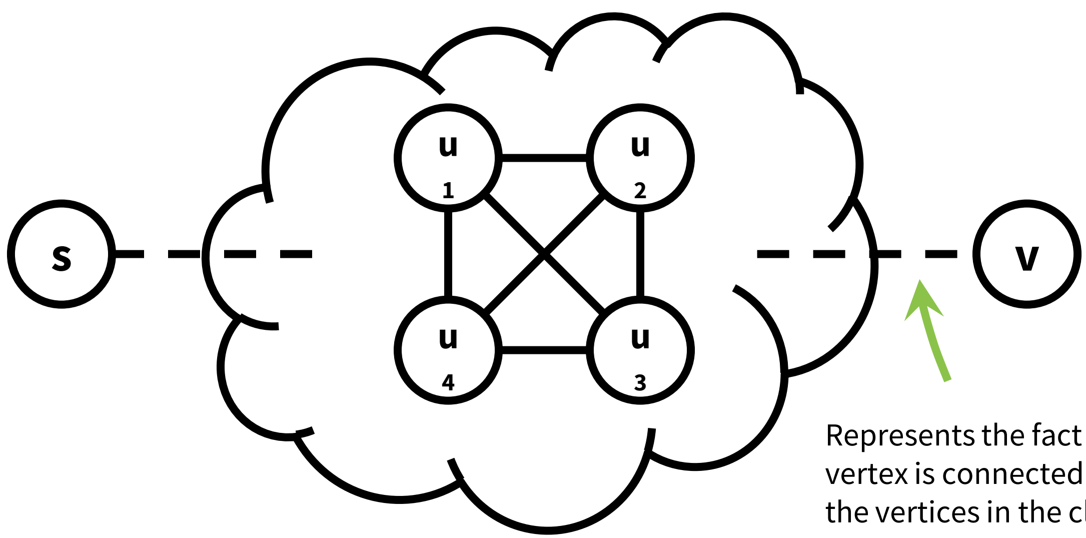
Let OPT(v, S) be the minimum cost of an s - v path that visits exactly the vertices in S. We assume $v \in S$. Let w(u, v) be the weight of the edge (u, v).

Claim OPT(v, S) satisfies the recurrence:

$$
\texttt{OPT(v,S)} = \begin{cases} 0 & \text{if } v = s \text{ and } S = \{s\} \\ \infty & \text{if } s \notin S \\ \min_{u \in S-\{v\}} \{\texttt{OPT(u,S-\{v\})} + \texttt{w(u,v)}\} & \text{otherwise} \end{cases}
$$

TSP

$$\mathrm{OPT}(v,S) = \begin{cases} 0 & \text{if } v = s \text{ and } S = \{s\} \\ \infty & \text{if } s \notin S \\ \min_{u \in S-\{v\}} \{\mathrm{OPT}(u, S-\{v\}) + w(u,v)\} & \text{otherwise} \end{cases}$$



Represents the fact that this vertex is connected to all of the vertices in the cloud.

TSP

To solve OPT(v, S), a problem of size |S|, we need to solve subproblems of size |S| - 1.

Idea Evaluate the recurrence on sets of size 1, 2, 3 …, n.

There are 2^n possible subsets of a set S, of which 2^{n-1} contain s.

TSP

```
algorithm tsp(G):
  n = |G.V|
  DP = []  # n × 2^(n-1) table
  s = random vertex from G.V
  DP[s][{s}] = 0
  for k = 2 to n:
    for all sets S ⊆ V where |S| = k and s ∈ S:
      for all v ∈ S - {s}:
        DP[v][S] = min_{u∈S-{v}}{DP[u][S-{v}] + w(u,v)}
  return min_{v≠s}{DP[v][V] + w(v,s)}
```

Runtime: $O(2^n n^2)$

TSP

```
algorithm tsp(G):
  n = |G.V|
  DP = []  # n × 2^(n-1) table
  s = random vertex from G.V
  DP[s][{s}] = 0
  for k = 2 to n:
    for all sets S ⊆ V where |S| = k and s ∈ S:
      for all v ∈ S - {s}:
        DP[v][S] = min_{u∈S-{v}}{DP[u][S-{v}] + w(u,v)}
  return min_{v≠s}{DP[v][V] + w(v,s)}
```

Runtime: $O(2^n n^2)$

We'll talk about this in a bit.

TSP

Each subset of V containing s can be mapped to a unique integer in 0, 1, 2, …, 2^{n-1} - 1.

- Think of the number as a bitvector where the present elements are 1s and the absent elements are 0s.
- Takes `O(n)`-time to compute the above number and index into the table, the cost per subproblem.

TSP

- Each subset of V containing s can be mapped to a unique integer in 0, 1, 2, …, $2^{n-1} - 1$.
  - Think of the number as a bitvector where the present elements are 1s and the absent elements are 0s.
  - Takes $O(n)$-time to compute the above number and index into the table, the cost per subproblem.
- $O(2^n n^2)$ total time.
  - $O(2^n n)$ total subproblems (cells in the table).
  - Solving each subproblem requires us to look at $O(n)$ different subproblems, and $O(1)$-time for each one.
  - Map all subsets of V to bitvectors in $O(n)$-time.

TSP

What's the difference between $n!$ and $2^n n^2$?

- Compare $20!$ and $2^{20}20^2$:
  - $20! \approx 2.4 \times 10^{18}$
  - $2^{20}20^2 \approx 4.2 \times 10^{8}$

TSP

What’s the difference between $n!$ and $2^n n^2$?

- Compare $20!$ and $2^{20}20^2$:
  - $20! \approx 2.4 \times 10^{18}$
  - $2^{20}20^2 \approx 4.2 \times 10^8$
- Compare $30!$ and $2^{30}30^2$:
  - $30! \approx 2.6 \times 10^{32}$
  - $2^{30}30^2 \approx 9.7 \times 10^{11}$

TSP

What's the difference between $n!$ and $2^n n^2$?

- Compare $20!$ and $2^{20}20^2$:
  - $20! \approx 2.4 \times 10^{18}$
  - $2^{20}20^2 \approx 4.2 \times 10^{8}$
- Compare $30!$ and $2^{30}30^2$:
  - $30! \approx 2.6 \times 10^{32}$
  - $2^{30}30^2 \approx 9.7 \times 10^{11}$
- Compare $40!$ and $2^{40}40^2$:
  - $40! \approx 8.2 \times 10^{47}$
  - $2^{40}40^2 = 1.8 \times 10^{15}$

TSP

Why this matters?

- Improving upon brute-force (e.g. n!) increases the size of problems that can be solved with exact answers.
- Though there might not exist a poly-time solution, an exponential solution often offers a considerable improvement.

0/1 Knapsack, revisited

Knapsack

weight	6	2	4	3	11
value	20	8	14	13	35

0/1 Knapsack

Suppose I only have one copy of each item.

What's the most valuable way to fill the knapsack?

capacity: 10

Total weight: 9

Total value: 35

Task Find the items to put in a 0/1 knapsack.

0/1 Knapsack

What I didn't say is this problem is known to be **NP**-hard.

- `O(n2^n)` Brute-force solution: try all possible subsets of the items and find the feasible set with the largest total value.
- `O(nW log(n))` Greedy solution: sort items by their "unit value" v_k / w_k.
- `O(nW)` Dynamic programming solution

0/1 Knapsack

Did we just prove P = NP?

A poly-time algorithm is one that runs in time polynomial in the total number of bits required to write out the input to the problem.

Therefore, $O(nW)$ is exponential in the number of bits required to write out the input (e.g. adding one more bit to the end of the representation of W doubles Its size and doubles the runtime).

The DP runtime of $O(nW)$ is better than our brute-force runtime of $O(n2^n)$, provided that W = $o(2^n)$.

That's a little-o, not a big-O.

For any fixed W, this algorithm runs in linear time!

Parameterized Complexity

Parameterized complexity is a branch of complexity theory that studies the hardness of problems with respect to different "parameters" of the input.

- In the case of 0/1 Knapsack, `O(nW)` has two parameters: the number of items (n) and capacity (W).
- Often, **NP**-hard problems aren't entirely infeasible as long as some parameter of the problem is fixed.

Fixed Parameter Tractability

Suppose that the input to a problem P can be characterized by two parameters, n and k.

P is called fixed-parameter tractable iff there is some algorithm that solves P in time `O(f(k)p(n))`.

- f(k) is an arbitrary function and p(n) is a polynomial in n..
- Intuitively, for any fixed k, the algorithm runs in a polynomial in n since that polynomial p(n) does not depend on choice of k.

Next Time

Approximation Algorithms, my fave!