CSE 521: Design & Analysis of Algorithms I

Dealing with NP-completeness

What to do if the problem you want to solve is NP-hard

- Can you take advantage of special structure?
  - e.g., in practice, the graphs that actually arise are far from arbitrary
    - maybe they have some special characteristic that allows you to solve the problem in your special case
      - for example the Independent-Set problem is easy on "interval graphs"
        - Exactly the case for interval scheduling!
    - search the literature to see if special cases already solved

What to do if the problem you want to solve is NP-hard

- Try an algorithm that is provably fast "on average".
  - To even try this one needs a model of what a typical instance is.
  - Typically, people consider "random graphs"
    - e.g. all graphs with a given # of edges are equally likely
  - Problems:
    - real data may not look like the random graphs
    - distributions of real data aren't easily analyzable

What to do if the problem you want to solve is NP-hard

- Use algorithms that are not efficient
  - Branch and bound, brute force
- Use algorithms that are not provably optimal
  - Heuristics such as local search, simulated annealing
  - Use approximation algorithms.

What to do if the problem you want to solve is NP-hard

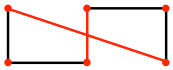
- Use heuristic algorithms and hope they give good answers
  - No guarantees of quality
  - Many different types of heuristic algorithms
  - Many different options, especially for optimization problems, such as TSP, where we want the best solution.

Heuristic algorithms for NP-hard problems

- **local search** for optimization problems
  - need a notion of two solutions being neighbors
  - Start at an arbitrary solution **S**
  - While there is a neighbor **T** of **S** that is better than **S**
    - **S←T**
- Usually fast but often gets stuck in a local optimum and misses the global optimum
  - With some notions of neighbor can take a long time in the worst case

e.g., Neighboring solutions for TSP

Solution S



Solution T

Two solutions are neighbors iff there is a pair of edges you can swap to transform one to the other

7

Heuristic algorithms for NP-hard problems

- **randomized local search**
  - start local search several times from random starting points and take the best answer found from each point
    - **more expensive than plain local search but usually much better answers**
- **simulated annealing**
  - like local search but at each step sometimes move to a worse neighbor with some probability
    - probability of going to a worse neighbor is set to decrease with time as, presumably, solution is closer to optimal
    - helps avoid getting stuck in a local optimum but often **slow to converge** (much more expensive than randomized local search)
    - analogy with slow cooling to get to lowest energy state in a crystal.

8

What to do if the problem you want to solve is NP-hard

- Try to find an approximation algorithm
  - Maybe you can't get the size of the best Vertex Cover but you can find one within a factor of **2** of the best
    - Given graph **G**=(**V**,E), start with an empty cover
    - **While** there are still edges in E left
      - **Choose** an edge **e**={**u**,**v**} in **E** and add both **u** and **v** to the cover
      - Remove all edges from **E** that touch either **u** or **v**.
    - Edges chosen don't share any vertices so optimal cover size must be at least # of edges chosen

9

What to do if the problem you want to solve is NP-hard

- Try to find an approximation algorithm
  - A c-approximation algorithm is a polynomial time algorithm that always produces a solution within a factor of c of optimal.
  - (If algorithm is randomized, require expected cost within a factor of c of optimal.)
- In last 20 years, huge progress:
  - Complexity – powerful techniques for proving no c-approx algs exist unless P=NP
  - Algorithms – new techniques

10

What to do if the problem you want to solve is NP-hard

- 4 classes of techniques:
  - Old-fashioned: greedy, dynamic programming based, etc.
  - LP-based rounding
  - Primal-dual schema
  - Semi-definite programming based.

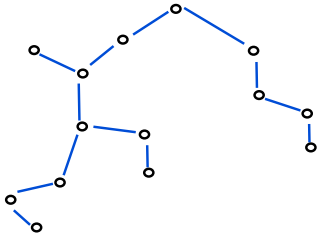
11

Travelling Salesperson Problem

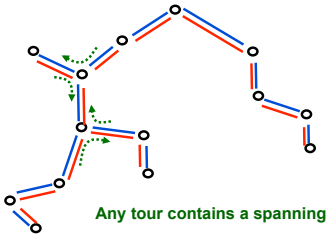
- TSP
  - Given a weighted graph **G** find of a smallest weight tour that visits all vertices in **G**
- **NP**-hard
- Start with classic result.

12

Minimum Spanning Tree Approximation



Minimum Spanning Tree Approximation: Factor of 2

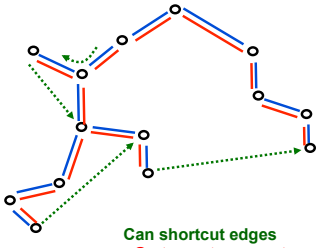


$MST(G) \le TOUR_{OPT}(G) \le 2\ MST(G) \le 2\ TOUR_{OPT}(G)$

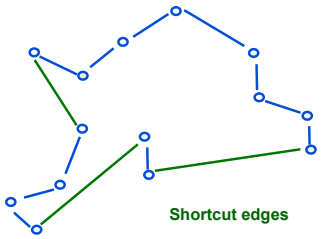
Why did this work?

- We found an Euler tour on a graph that used the edges of the original graph (possibly repeated).
- The weight of the tour was the total weight of the new graph.
- Suppose now
  - All edges possible
  - Weights satisfy triangle inequality
    - $c(u,w) \le c(u,v)+c(v,w)$

Minimum Spanning Tree Approximation: Triangle Inequality



Minimum Spanning Tree Approximation: Factor of 2

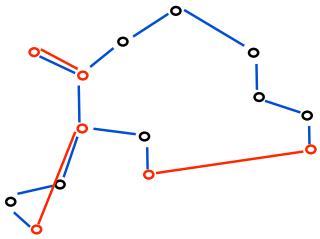


$TOUR_{OPT}(G) \le 2\ MST(G) \le 2\ TOUR_{OPT}(G)$

Christofides Algorithm: A factor 3/2 approximation

- Any Eulerian subgraph of the weighted complete graph will do
  - Eulerian graphs require that all vertices have even degree so
- Christofides Algorithm
  - Compute an MST **T**
  - Find the set **O** of odd-degree vertices in **T**
  - Add a minimum-weight perfect matching **M** on the vertices in **O** to **T** to make every vertex have even degree
    - There are an even number of odd-degree vertices!
    - **M** is polytime computable for non-bipartite graphs (not easy)
  - Use an Euler Tour **E** in **T∪M** and then shortcut as before
- **Claim:** $Cost(E) \le 1.5\ TOUR_{OPT}$

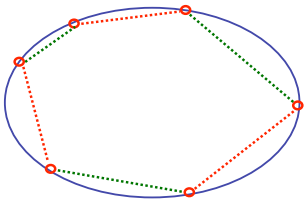
Christofides Approximation



19

Christofides Approximation

Any tour costs at least the cost of two matchings on O



Claim: 2 Cost(M) ≤ $TOUR_{OPT}$

20

LP-Based Algorithms

- Formulate problem as 01P
- Relax to linear program.
- Solve linear program
- Round linear programming solution, hopefully without changing the value of the objective too much.
  - Deterministic rounding
  - Randomized rounding

21

Semidefinite-Programming Based Algorithms

- Formulate problem as 01 quadratic program
- Relax to semi-definite program (SDP).
- Solve SDP
- Round solution

22

What to do if the problem you want to solve is NP-hard

- More on approximation algorithms
  - Recent research has classified problems based on what kinds of approximations are possible if **P≠NP**
    - Best: **(1+ε)** factor for any **ε>0**.
      - packing and some scheduling problems, TSP in plane
    - Some fixed constant factor **> 1**, e.g. **2, 3/2, 100**
      - Vertex Cover, TSP in space, other scheduling problems
    - **Θ(log n)** factor
      - Set Cover, Graph Partitioning problems
    - Worst: **$\Omega(n^{1-\varepsilon})$** factor for any **ε>0**
      - Clique, Independent-Set, Coloring

23

What to do if the problem you want to solve is NP-hard

- Polynomial-time approximation algorithms for **NP**-hard problems can sometimes be ruled out unless **P=NP**
  - E.g. **Coloring Problem:** Given a graph **G=(V,E)** find the smallest **k** such that **G** has a **k**-coloring.
    - No approximation ratio better than **4/3** is possible unless **P=NP**
      - The graph in our **NP**-completeness reduction is always **4**-colorable. This would let us figure out if it is **3**-colorable.

24

PCP Theorem and Hardness of Approximation

- PCP (Probabilistically Checkable Proofs) Theorem: Every $A \in NP$ has a polytime verifier V that looks at only 3 random bits of its certificate c such that
  - $x \in A \Rightarrow$ There is a certificate c such that $V(x,c)$ always outputs YES
  - x not $\in A \Rightarrow$ For every certificate c, $V(x,c)$ outputs YES with probability < 0.99999
- Implies that there is a polytime reduction f such that
  - $F \in 3SAT \Rightarrow f(F) \in 3SAT$
  - F not $\in$ #SAT $\Rightarrow$ any truth assignment to $f(F)$ satisfies at most 88% ($< 7/8+\varepsilon$) of clauses of F

25