CSE 421
Introduction to Algorithms

Lecture 27: Dealing with NP-completeness:
Approximation Algorithms
Local Search
Exponential-time Algorithms

Reminder/Announcement

- The Final Exam is Monday December 9, 2:30-4:20 pm here but we may be able to extend this to 4:45 pm
  - If nobody has a conflict that would prevent them staying longer, I will extend the time available.
  - Email me by the end of day today if you have a conflict with staying longer

Suggestion: Please finish everything by Thursday even if you have late days so you have time to study.

- I sent an email over the weekend with information about the exam and a sample final
  - It will be comprehensive and similar in style to the midterm.

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

2nd thing to try if your problem is a minimization or maximization problem

- Try to find a polynomial-time worst-case approximation algorithm
  - For a minimization problem
    - Find a solution with value $\leq \boldsymbol{K}$ times the optimum
  - For a maximization problem
    - Find a solution with value $\geq \mathbf{1}/\boldsymbol{K}$ times the optimum

Want $\boldsymbol{K}$ to be as close to $\mathbf{1}$ as possible.

Travelling-Salesperson Problem (TSP)

Travelling-Salesperson Problem (TSP):

Given: a set of n cities $v_1, \ldots, v_n$ and distance function d that gives distance $d(v_i, v_j)$ between each pair of cities

Find the shortest tour that visits all n cities.

MetricTSP:

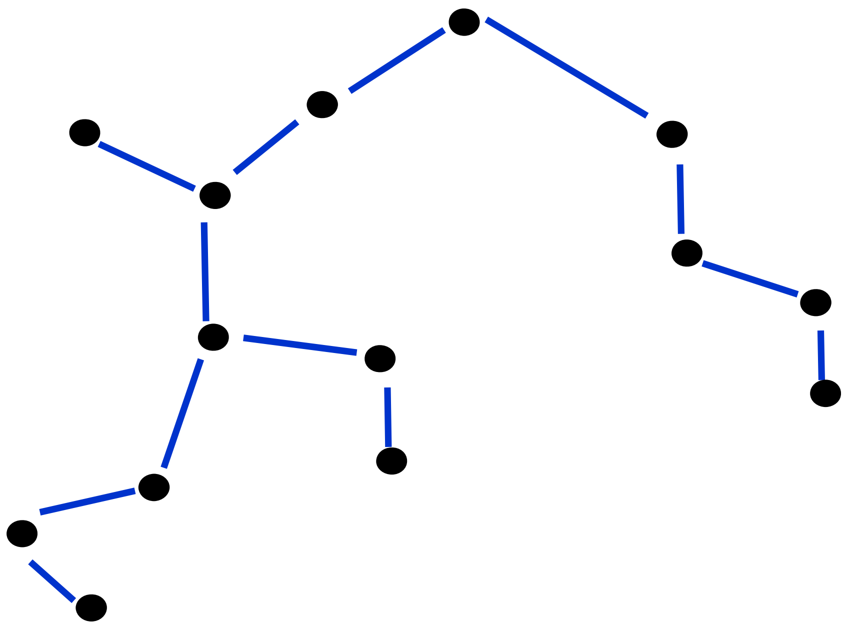
The distance function d satisfies the triangle inequality:

$$d(u, w) \leq d(u, v) + d(v, w)$$

Proper tour: visit each city exactly once.

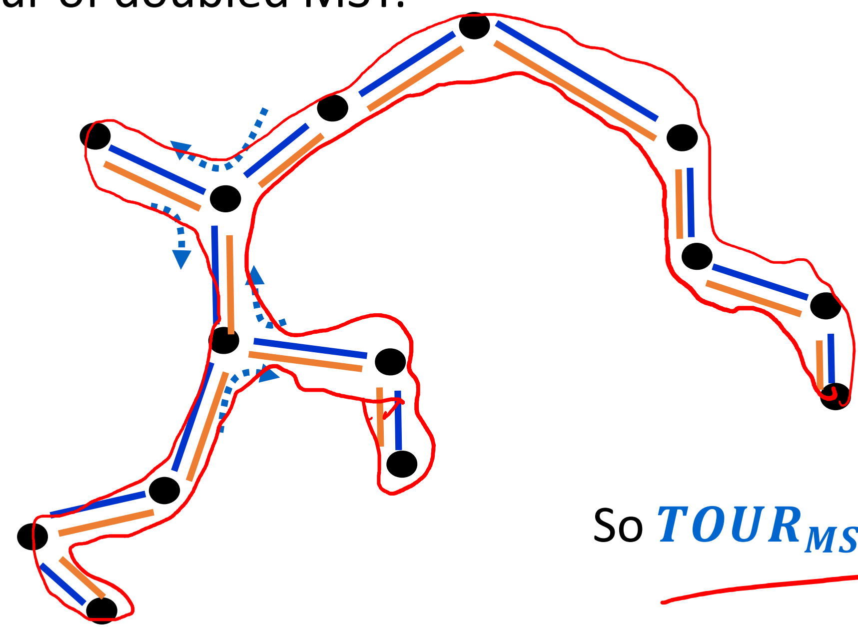
Still NP-complete

Minimum Spanning Tree Approximation: Factor of 2



TSP: Minimum Spanning Tree Factor 2 Approximation

Euler Tour of doubled MST:



Euler tour covers each edge twice
so $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G)}$

Any tour contains a spanning tree
so $\boldsymbol{MST(G) \leq TOUR_{OPT}(G)}$

So $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G) \leq 2\ TOUR_{OPT}(G)}$

This visits each node more than once, so not a "proper" tour.

Why did this work?

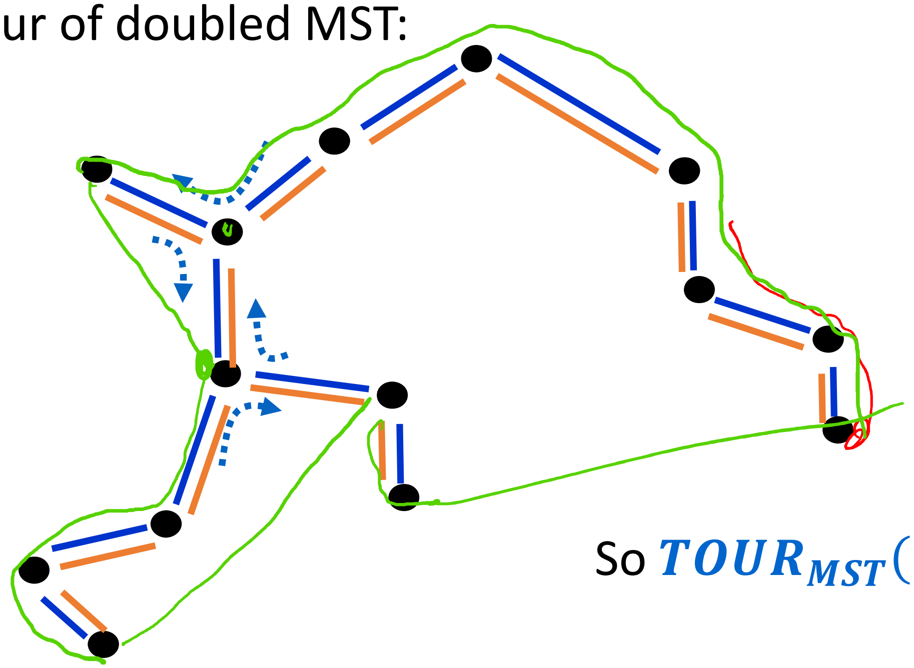
cycle that touches every edge exactly once!

Requires "all even degree"!

- 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 the triangle inequality (MetricTSP)

MetricTSP: Minimum Spanning Tree Factor 2 Approximation

Euler Tour of doubled MST:



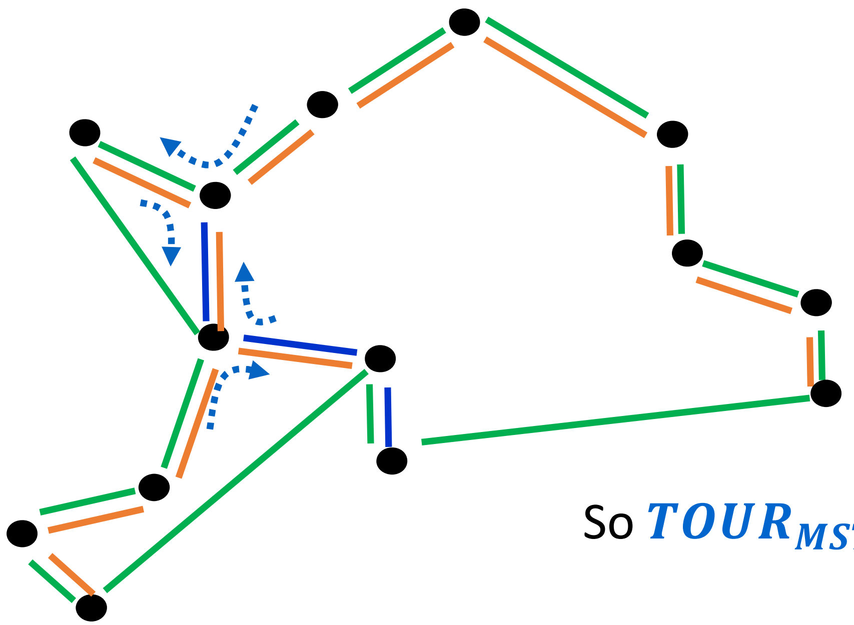
Euler tour covers each edge twice so $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G)}$

Any tour contains a spanning tree so $\boldsymbol{MST(G) \leq TOUR_{OPT}(G)}$

So $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G) \leq 2\ TOUR_{OPT}(G)}$

Instead: take shortcut to next unvisited vertex on the Euler tour
By triangle inequality this can only be shorter.

MetricTSP: Minimum Spanning Tree Factor 2 Approximation



Euler tour covers each edge twice
so $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G)}$

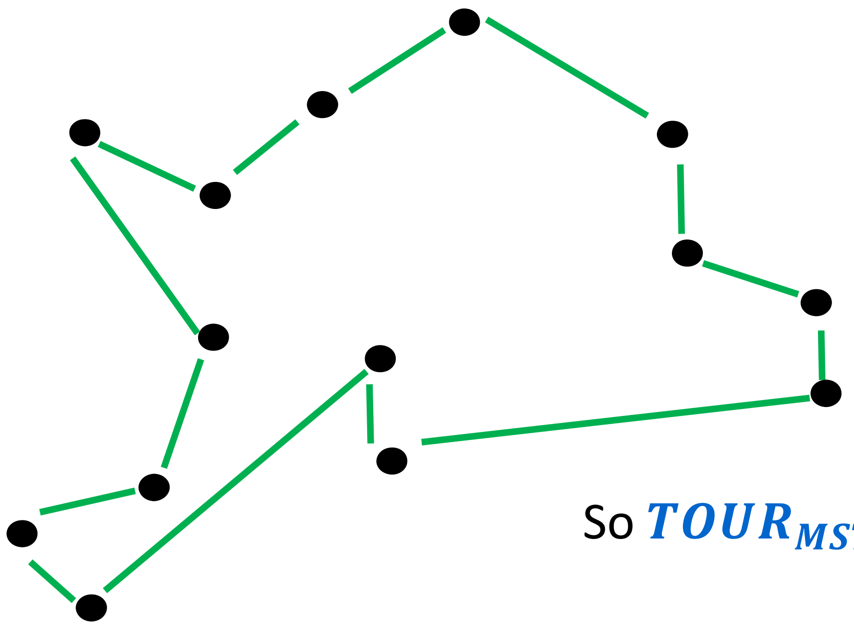
Any tour contains a spanning tree
so $\boldsymbol{MST(G) \leq TOUR_{OPT}(G)}$

So $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G) \leq 2\ TOUR_{OPT}(G)}$

Instead: take shortcut to next unvisited vertex on the Euler tour
By triangle inequality this can only be shorter.

MetricTSP: Minimum Spanning Tree Factor 2 Approximation

Final:



Euler tour covers each edge twice
so $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G)}$

Any tour contains a spanning tree
so $\boldsymbol{MST(G) \leq TOUR_{OPT}(G)}$

So $\boldsymbol{TOUR_{MST}(G) = 2\ MST(G) \leq 2\ TOUR_{OPT}(G)}$

Instead: take shortcut to next unvisited vertex on the Euler tour
By triangle inequality this can only be shorter.

Christofides Algorithm: A factor 3/2 approximation

Any subgraph of the weighted complete graph that has an Euler Tour will work also!

Fact: To have an Euler Tour it suffices to have all degrees even.

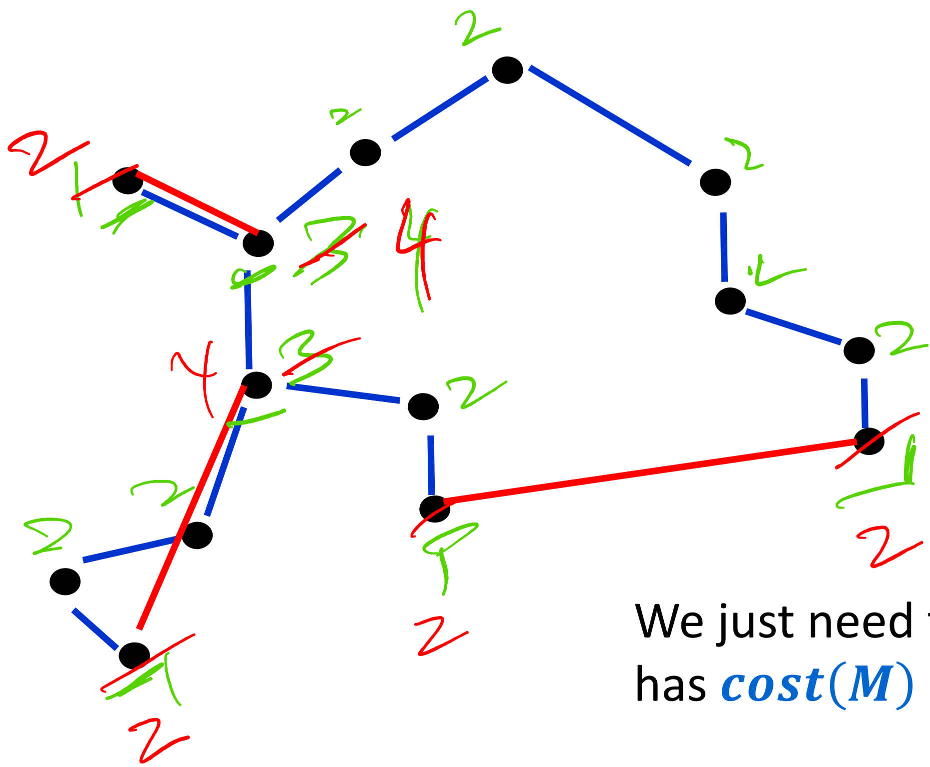
Christofides Algorithm:

- Compute an MST $\boldsymbol{T}$
- Find the set $\boldsymbol{O}$ of odd-degree vertices in $\boldsymbol{T}$
- Add a minimum-weight perfect matching* $\boldsymbol{M}$ on the vertices in $\boldsymbol{O}$ to $\boldsymbol{T}$ to make every vertex have even degree
  - There are an even number of odd-degree vertices!
- Use an Euler Tour $\boldsymbol{E}$ in $\boldsymbol{T} \cup \boldsymbol{M}$ and then shortcut as before

Theorem: $\boldsymbol{Cost(E) \leq 1.5\, TOUR_{OPT}}$

*Requires finding optimal matchings in general graphs, not just bipartite ones

Christofides Approximation

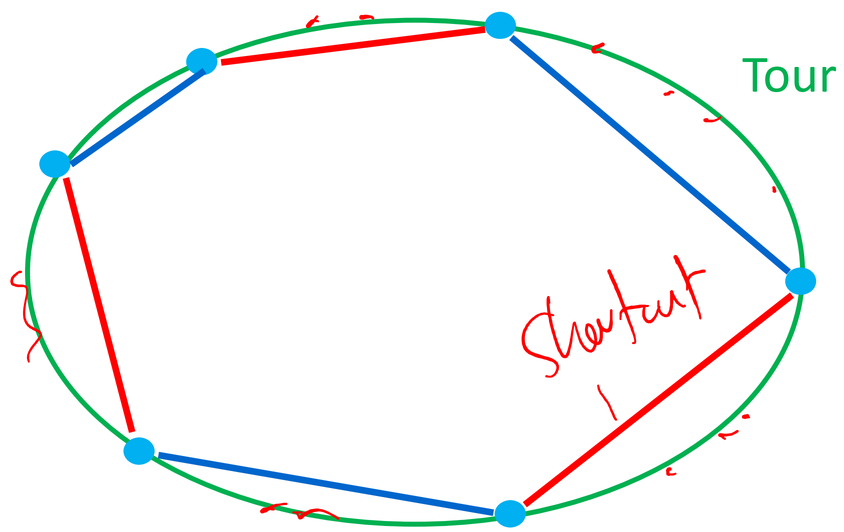


Any tour contains a spanning tree so $MST \leq TOUR_{OPT}$

We just need to show that the matching M has $cost(M) \leq TOUR_{OPT}/2$

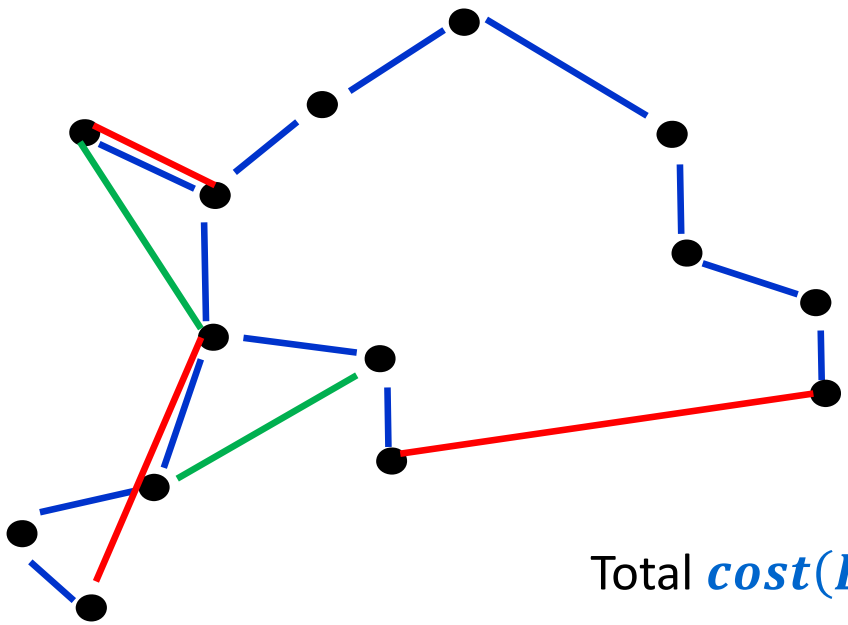
Christofides Approximation

Any tour costs at least the cost of two matchings M_1 and M_2 on O



$$2\, cost(M) \le cost(M_1) + cost(M_2) \le TOUR_{OPT}$$

Christofides Approximation Final Tour



Total $\boldsymbol{cost(E) \leq 3\ TOUR_{OPT}/2}$

Max-3SAT Approximation

Max-3SAT: Given a 3CNF formula $\boldsymbol{F}$ find a truth assignment that satisfies the maximum possible # of clauses of $\boldsymbol{F}$.

Observation: A single clause on 3 variables only rules out $\mathbf{1/8}$ of the possible truth assignments since each literal has to be false to be ruled out.

$\Rightarrow$ a random truth assignment will satisfy the clause with probability $\mathbf{7/8}$.

So in expectation, if $\boldsymbol{F}$ has $\boldsymbol{m}$ clauses, a random assignment satisfies $\mathbf{7}\boldsymbol{m}/\mathbf{8}$ of them.

A greedy algorithm can achieve this: Choose most frequent literal appearing in clauses that are not yet satisfied and set it to true.

If $\mathbf{P} \neq \mathbf{NP}$ no better approximation is possible ~~$\frac{7}{8}+\varepsilon$~~

Knapsack Problem

Each item has a value v_i and a weight w_i.

Maximize $\sum_{i \in S} v_i$ with $\sum_{i \in S} w_i \leq W$.

$O(nW)$ — n bits — not poly

Theorem: For any $\varepsilon > 0$ there is an algorithm that produces a solution within $(1+\varepsilon)$ factor of optimal for the Knapsack problem with running time $O(n^2/\varepsilon^2)$

"Polynomial-Time Approximation Scheme" or PTAS

Algorithm: Maintain the high order bits in the dynamic programming solution.

Approximation Algorithms using Linear Programming

The generic approach to creating approximation algorithms for **NP**-optimization problems using Linear Programming:

1. Express the original problem as an Integer Program (ILP) or 01-Program (01-LP)
2. Keep same linear constraints but remove the integer requirement to get an LP. (Called the "LP relaxation".)
3. Solve the LP to yield a fractional solution
4. "Round" the fractional solution to an integer solution that satisfies all constraints.

Prove a bound on the ratio of the integer solution to the fractional LP solution

Observation: The LP optimum has at least as good an objective function value as the original problem since the LP allows all the ILP solutions plus some other fractional ones.

Recall: Greedy Approximation for Vertex-Cover

```
On input G = (V, E)
W ← ∅
E' ← E
while E' ≠ ∅
    select any e = (u, v) ∈ E'
    W ← W ∪ {u, v}
    E' ← E' \ {edges e ∈ E' that touch u or v}
```

Claim: At most a factor $\mathbf{2}$ larger than the optimal vertex-cover size.

Proof: Edges selected don't share any vertices so any vertex-cover must choose at least one of $\boldsymbol{u}$ or $\boldsymbol{v}$ each time.

Weighted Vertex Cover

Weighted Vertex Cover:

Given graph $\boldsymbol{G = (V, E)}$ with each vertex $\boldsymbol{v}$ having a weight $\boldsymbol{w_v \geq 0}$.

Find a vertex cover $\boldsymbol{C \subseteq V}$ of $\boldsymbol{G}$ that minimizes $\sum_{v \in C} w_v$.

The greedy approximation approach doesn't work for this weighted version because for each edge, one of the two endpoints might have much larger weight than the other.

Weighted Vertex-Cover as an Integer Program

Variables x_v for $v \in V$

Minimize $\sum_{v \in V} w_v \cdot x_v$

subject to

$x_u + x_v \geq 1$ for each edge $\{u, v\} \in E$

$x_v \in \{0, 1\}$ for each node $v \in V$

The last line is equivalent to:

$0 \leq x_v \leq 1$ for each node $v \in V$

x_v integral for each node $v \in V$

Write OPT for the optimum cover weight

LP relaxation:

Minimize $\sum_{v \in V} w_v \cdot x_v$

subject to

$x_u + x_v \geq 1$ for each edge $\{u, v\} \in E$

$0 \leq x_v \leq 1$ for each node $v \in V$

Write OPT_{LP} for the optimum LP value

How do we round a LP solution achieving this value?

$OPT_{LP} \leq OPT$

LP-Rounding to Approximate Weighted Vertex Cover

1. Solve the LP Relaxation
   a) Solution gives values $x_v \in [0,1]$ for each $v \in V$
   b) $x_u + x_v \geq 1$ for each edge (u,v)
2. Round: Define $C \subseteq V$ to be $\{v : x_v \geq 1/2\}$
3. Observe that C is a vertex cover:
   - By 1 b), for each edge (u,v), at least one of $x_u \geq 1/2$ or $x_v \geq 1/2$ is true so either $u \in C$ or $v \in C$.
4. Since $x_v \geq 1/2$ for every $v \in C$, the total weight of C is

$$\sum_{v \in C} w_v \leq \sum_{v \in C} w_v \cdot (2x_v)$$
$$= 2\sum_{v \in C} w_v \cdot x_v \leq 2\sum_{v \in V} w_v \cdot x_v = 2\, OPT_{LP} \leq 2\, OPT.$$

Factor 2 approximation!

More on LP and Related Approximation Methods

More sophisticated methods for rounding variables $\boldsymbol{x_i} \in [\mathbf{0}, \mathbf{1}]$

- Randomized: View each $\boldsymbol{x_i}$ as a probability and independently produce solution $\boldsymbol{y_i} = \begin{cases} \mathbf{1} & \text{with probability } \boldsymbol{x_i} \\ \mathbf{0} & \text{with probability } \mathbf{1} - \boldsymbol{x_i} \end{cases}$
- Correlated random sampling. Apply the above but "correlate" choices somehow

Instead of LP relaxations, use "Semi-Definite Programming (SDP)" relaxations.

- SDPs generalize LPs. They can also be solved efficiently using Ellipsoid and Interior Point Methods. They are a special case of convex programming.
- Currently yield the best approximations known for many **NP**-hard problems.

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

NP-completeness is a worst-case notion...

- Try an algorithm that is provably fast "on average".
  - To even show 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
    - In this case one can sometimes show that many NP-hard problems are easy
  - Problems:
    - real data doesn't look like the random graphs
    - distributions of real data aren't analyzable

Hardness of Approximation

Polynomial-time approximation algorithms for **NP**-hard optimization problems can sometimes be ruled out unless $\mathbf{P} = \mathbf{NP}$.

Easy example:

Coloring: Given a graph $G = (V, E)$ find the smallest k such that G has a k-coloring.

Because $\mathbf{3}$-coloring is **NP**-hard, no approximation ratio better than $\mathbf{4/3}$ is possible unless $\mathbf{P} = \mathbf{NP}$ because you would have to be able to figure out if a $\mathbf{3}$-colorable graph can be colored in $< \mathbf{4}$ colors. i.e. if it can be $\mathbf{3}$-colored.

Best known hardness $n^{1-o(1)}$ factor

- We now know a huge amount about the hardness of approximating **NP** optimization problems if $\mathbf{P} \neq \mathbf{NP}$.
- Approximation factors are very different even for closely related problems like **Vertex-Cover** and **Independent-Set**.

Approximation Algorithms/Hardness of Approximation

Research has classified many problems based on what kinds of polytime approximations are possible if $\mathbf{P} \neq \mathbf{NP}$

- **Best:** $(\mathbf{1}+\boldsymbol{\varepsilon})$ factor for any $\boldsymbol{\varepsilon} > \mathbf{0}$. (PTAS)
  - packing and some scheduling problems, TSP in plane
- Some fixed constant factor $> \mathbf{1}$. e.g. $\mathbf{2}, \mathbf{3/2}, \mathbf{8/7}, \mathbf{100}$
  - Vertex Cover, Max-3SAT, MetricTSP, other scheduling problems
  - Exact best factors or very close upper/lower bounds known for many problems.
- $\Theta(\log \boldsymbol{n})$ factor
  - Set Cover, Graph Partitioning problems
- **Worst:** $\Omega(\boldsymbol{n}^{\mathbf{1}-\boldsymbol{\varepsilon}})$ factor for every $\boldsymbol{\varepsilon} > \mathbf{0}$.
  - Clique, Independent-Set, Coloring

Heuristic Algorithms

These algorithms typically do not have proven bounds on solution quality:

The most important of these methods are based on variants of

Local search:

- Need a notion of two solutions being neighbors

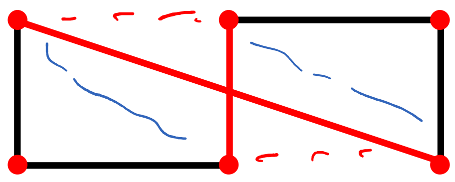
Start at an arbitrary solution S

While there is a neighbor T **of** S **that is better than** S

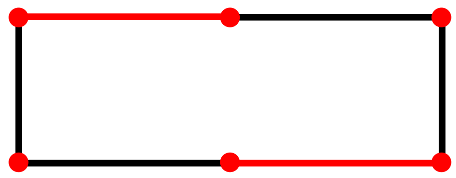
$S \leftarrow T$

e.g., Neighboring solutions for TSP

Solution ***S***



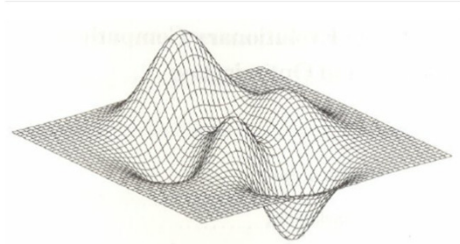
Solution ***T***



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

*These are called 2-OPT neighbors. There are other more sophisticated neighbor structures

Variants of Local Search



Basic local search (greedy)

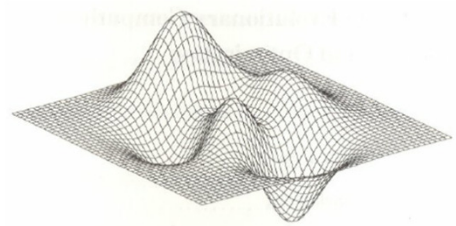
- *Usually fast but often gets stuck in a local optimum that is far from the global optimum*
- *With some notions of neighbor structure even this can take a long time in the worst case*

Randomized local search:

Start local search several times from random starting points and take the best answer found overall.

- *More expensive than plain local search but usually much better answers. It is usual easy to control the time spent so this is almost always better to do.*

Variants of Local Search



Metropolis Algorithm

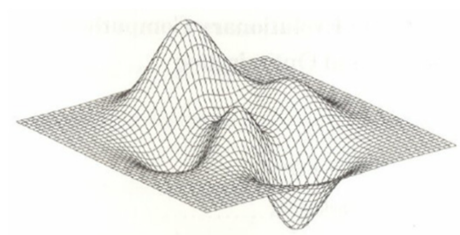
Like randomized local search except that at each step one always chooses a random neighbor but doesn't always move to it:

e.g. Always move to the neighbor if it is better but move to a worse neighbor with some fixed probability depending on how much worse it is.
(Fixed inverse temperature.) cf. CSE 312 Markov Chain Knapsack assignment.

Advantage: If local optima are not too deep/steep, will not get stuck there. However can still get stuck

Often used in practice. Drawback: Each run can be much longer than local search but one can hope to try to make it up with solution quality. A good option to compare with randomized local search. It is unclear which will be better in a given circumstance.

Variants of Local Search



Simulated Annealing

Like Metropolis algorithm but probability of going to a worse neighbor is set to decrease with time on a "cooling schedule" as, presumably, solution is closer to optimal

(analogy with slow cooling to get to lowest energy state in a crystal (or in forging a metal)

Much slower to converge than Metropolis.

M*ost improvement occurs at some fixed temperature.*

Answers usually not much better than Metropolis, if at all, so not generally worth the extra compute time.