

**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
- 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

2<sup>nd</sup> 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  $\leq K$  times the optimum
  - For a maximization problem
    - Find a solution with value  $\geq 1/K$  times the optimum

Want  $K$  to be as close to  $1$  as possible.

# Travelling-Salesperson Problem (TSP)

## Travelling-Salesperson Problem (TSP):

**Given:** a set of  $n$  cities  $v_1, \dots, v_n$  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.

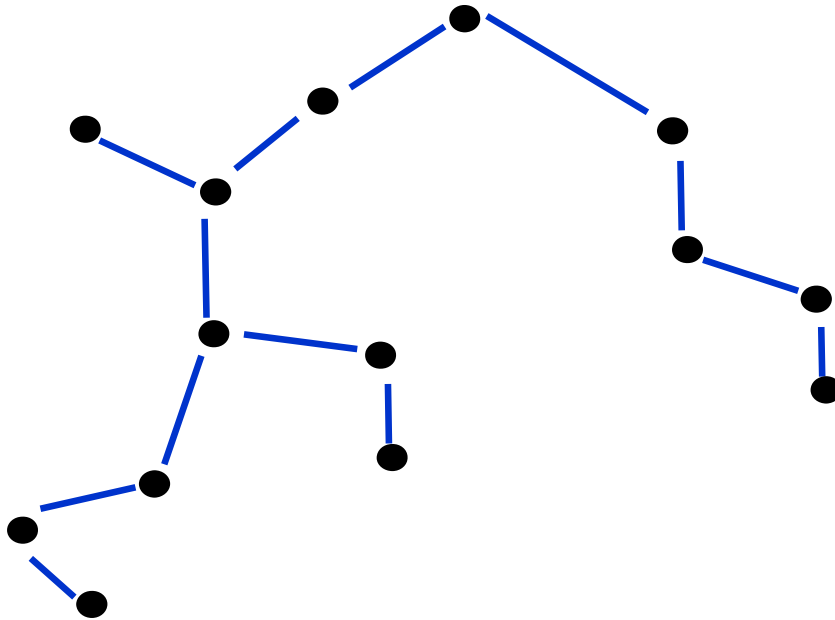
## MetricTSP:

The distance function  $d$  satisfies the triangle inequality:

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

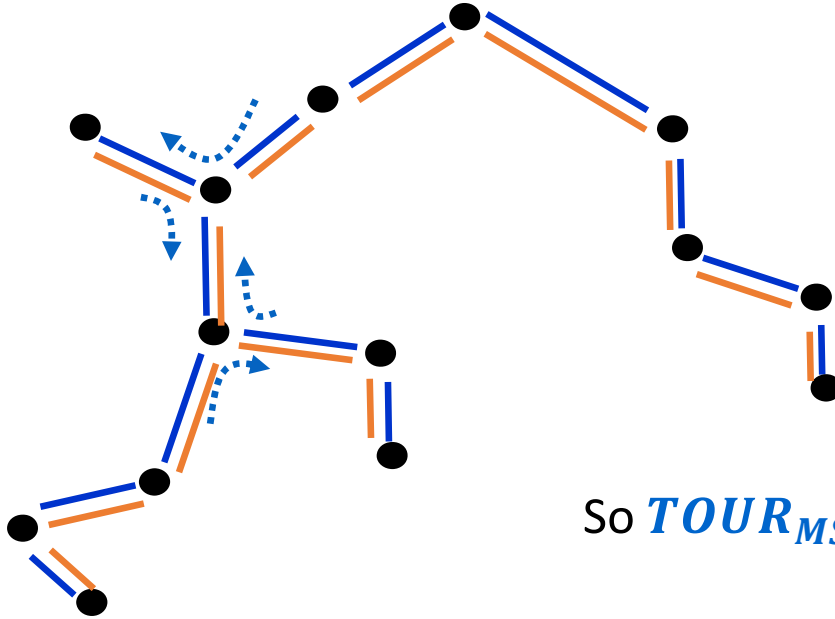
Proper tour: visit each city exactly once.

## 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  $TOUR_{MST}(G) = 2 MST(G)$

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

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

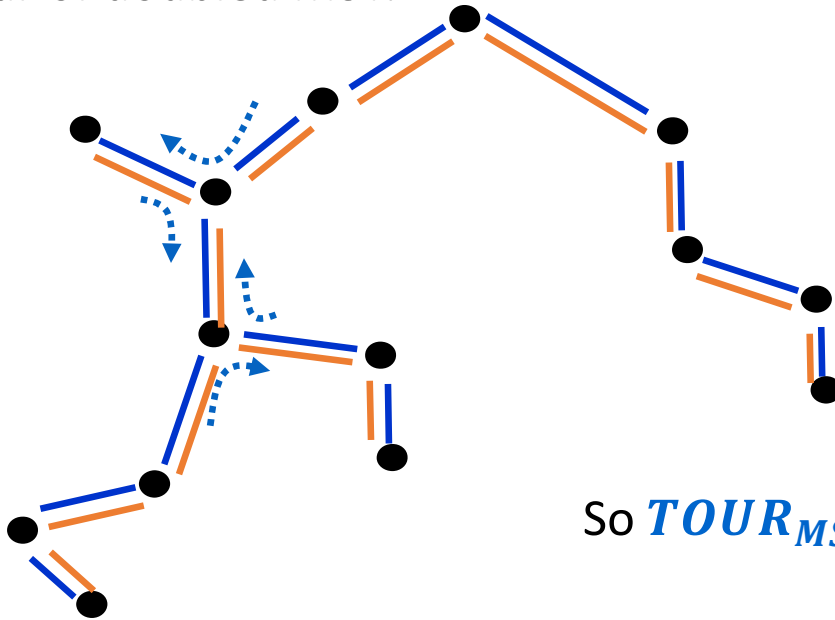
This visits each node more than once, so not a proper tour.

# Why did this work?

- 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  $TOUR_{MST}(G) = 2 MST(G)$

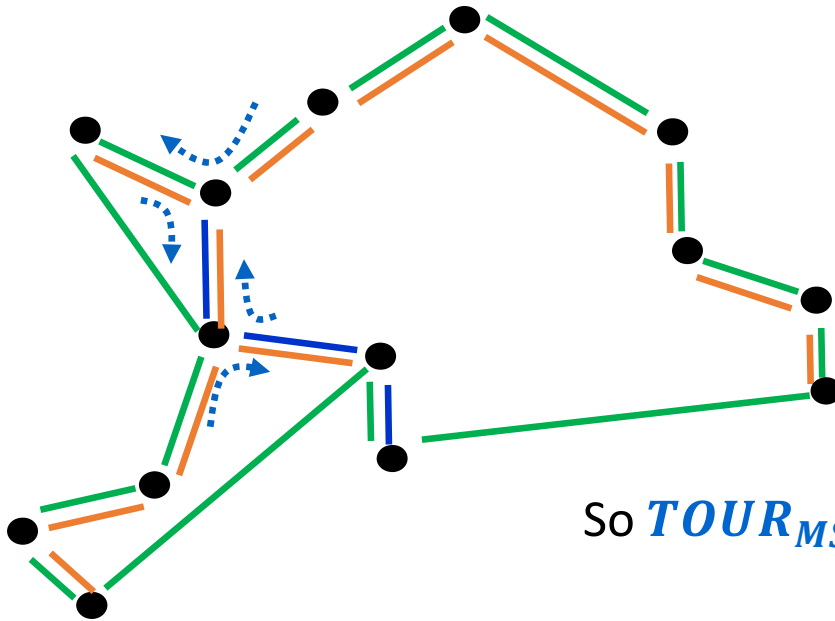
Any tour contains a spanning tree  
so  $MST(G) \leq TOUR_{OPT}(G)$

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

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  $TOUR_{MST}(G) = 2 MST(G)$

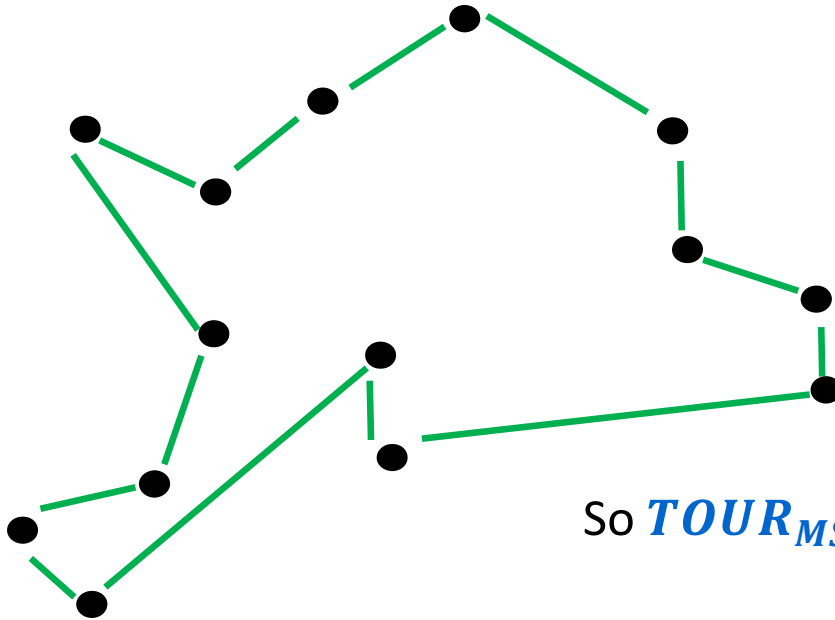
Any tour contains a spanning tree  
so  $MST(G) \leq TOUR_{OPT}(G)$

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

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  $TOUR_{MST}(G) = 2 MST(G)$

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

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

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.

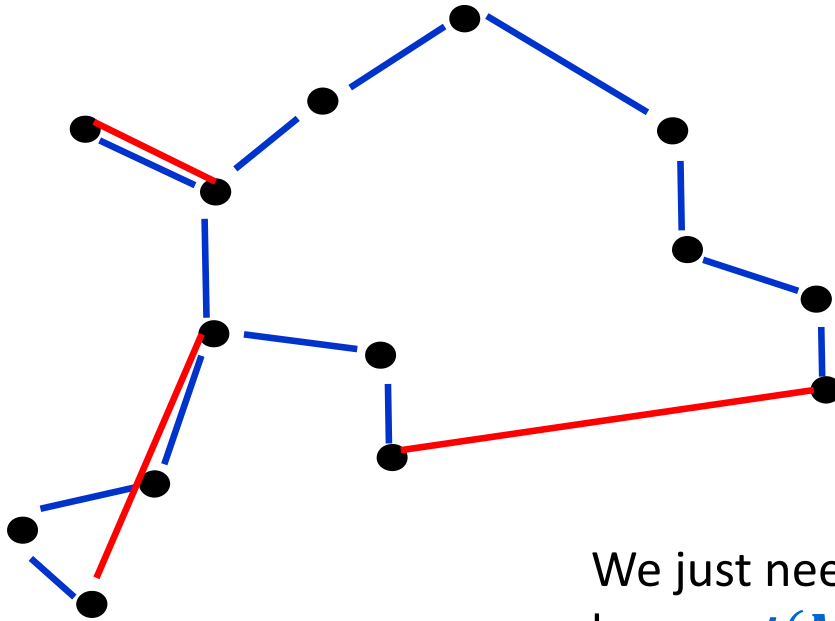
## Christofides Algorithm:

- Compute an MST  $T$
- 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!
- Use an Euler Tour  $E$  in  $T \cup M$  and then shortcut as before

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

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

# Christofides Approximation

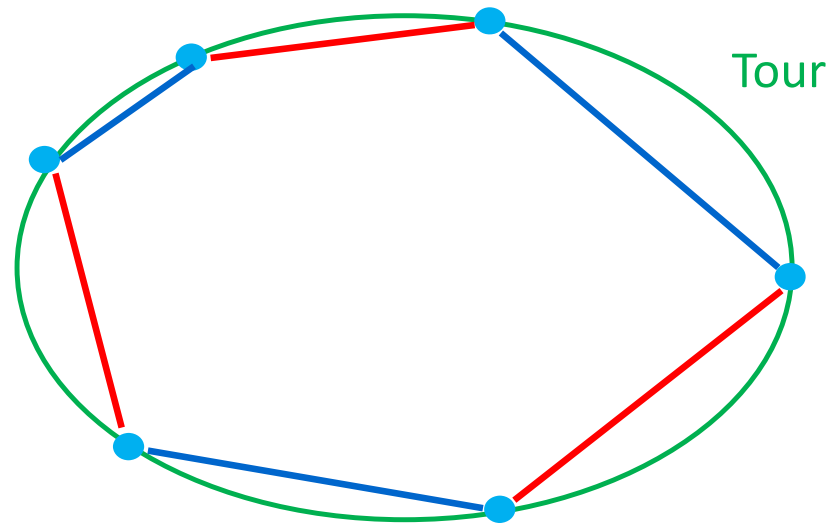


Any tour contains a spanning tree  
so  $MST \leq TOUR_{OPT}$

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

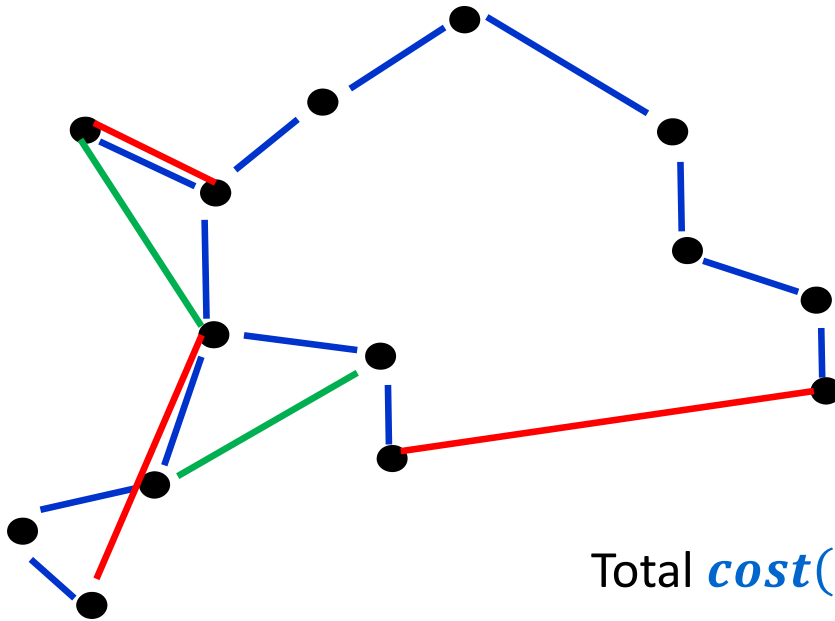
# Christofides Approximation

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



$$2 \text{ cost}(M) \leq \text{cost}(M_1) + \text{cost}(M_2) \leq \text{TOUR}_{OPT}$$

## Christofides Approximation Final Tour



$$\text{Total } \mathit{cost}(E) \leq 3 \mathit{TOUR}_{OPT}/2$$

# Max-3SAT Approximation

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

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

⇒ a random truth assignment will satisfy the clause with probability  $7/8$ .

So in expectation, if  $F$  has  $m$  clauses, a random assignment satisfies  $7m/8$  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  $P \neq NP$  no better approximation is possible

# Knapsack Problem

Each item has a value  $v_i$  and a weight  $w_i$ .

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

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

“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  $G = (V, E)$

$W \leftarrow \emptyset$

$E' \leftarrow E$

while  $E' \neq \emptyset$

    select any  $e = (u, v) \in E'$

$W \leftarrow W \cup \{u, v\}$

$E' \leftarrow E' \setminus \{\text{edges } e \in E' \text{ that touch } u \text{ or } v\}$

**Claim:** At most a factor **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  $u$  or  $v$  each time.

# Weighted Vertex Cover

## Weighted Vertex Cover:

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

Find a vertex cover  $C \subseteq V$  of  $G$  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?

# 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

$$\sum_{v \in C} w_v \leq \sum_{v \in C} w_v \cdot (2x_v)$$

$$= 2 \sum_{v \in C} w_v \cdot x_v \leq 2 \sum_{v \in V} w_v \cdot x_v = 2 \text{OPT}_{LP} \leq 2 \text{OPT}.$$

Factor 2 approximation!

# More on LP and Related Approximation Methods

More sophisticated methods for rounding variables  $x_i \in [0, 1]$

- Randomized: View each  $x_i$  as a probability and independently produce

$$\text{solution } y_i = \begin{cases} 1 & \text{with probability } x_i \\ 0 & \text{with probability } 1 - x_i \end{cases}$$

- 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 **P = NP**.

Easy example:

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

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

- We now know a huge amount about the hardness of approximating **NP** optimization problems if **P  $\neq$  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  $P \neq NP$

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

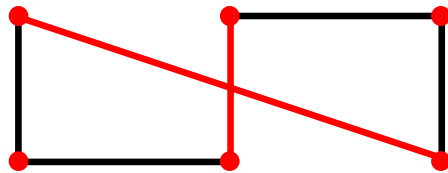
Start at an arbitrary solution  $S$

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

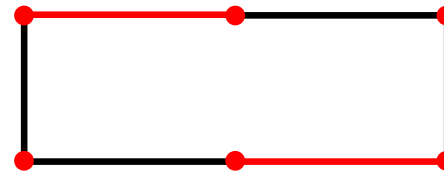
$S \leftarrow T$

## e.g., Neighboring solutions for TSP

Solution  $S$



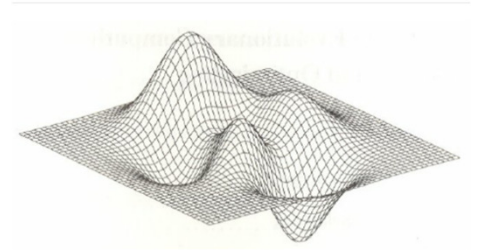
Solution  $T$



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:

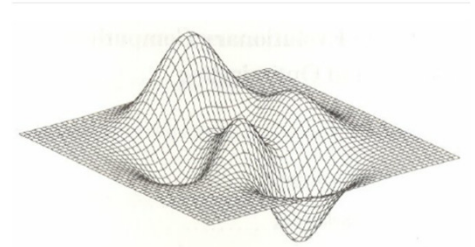
e.g. Always move to the neighbor if it is better but move to a worse neighbor with some fixed probability depending on how much worse it is.

(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

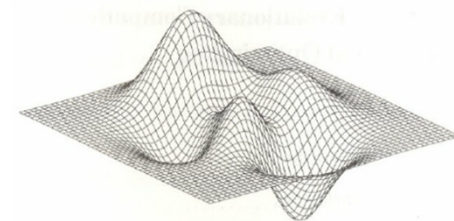
Like Metropolis algorithm but probability of going to a worse neighbor is set to decrease with time on a “cooling schedule” as, presumably, solution is closer to optimal

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

*Much slower to converge than Metropolis.*

*Most improvement occurs at some fixed temperature.*

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



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

Maybe you only need to solve it if the solution size is small...

- What if you only need find cliques or vertex covers of constant size?
- For both **Clique** and **Vertex Cover**, the obvious brute force algorithm would have time  $\Theta(n^k)$ : try all subsets of size  $k$ .
- For **Clique** the best algorithms known are all  $n^{\Omega(k)}$
- However, **Vertex Cover** has a much better algorithm with

The theory of **fixed parameter tractability** looks at **NP** problems using a second parameter  $k$  in addition to input size  $n$  and seeks algorithms with running times  $f(k) \cdot n^{O(1)}$  where  $f$  might be exponential.

- More later

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

Try to make an exponential-time solution as efficient as possible.

e.g. Try to search the space of possible hints/certificates in a more efficient way and hope that it is quick enough.

## Backtracking search

e.g., for **SAT**, search through the  $2^n$  possible truth assignments...

...but set the truth values one-by-one so we can be able to figure out whole parts of the space to avoid,

e.g. Given  $F = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_4 \vee \neg x_3) \wedge (x_1 \vee x_4)$

after setting  $x_1 = 1$  and  $x_2 = 0$  we don't even need to set  $x_3$  or  $x_4$  to know that it won't satisfy  $F$ .

**Next Class:** Much more clever backtracking search for **SAT** solutions



# Exponential-Time Algorithms

**Branch-and-bound search** for optimization problems:

- **Branch:** Use backtracking search through a tree representing partial solutions
- **Bound:** In addition to keeping track of the best full solution found so far, at each step produce a bound on the quality of the best possible completion of the current partial solution
  - If that best possible completion is worse than the best full solution found so far, prune the search and backtrack instead.

Example: In backtracking search for **MetricTSP** one can use linear programming to provide lower bounds

**Note:** An excellent exact solver for **MetricTSP** called **Concorde** combines branch-and-bound and LP/ILP methods and will solve problems involving thousands of cities.

# Other Heuristic Algorithms you might hear about

## Genetic algorithms:

- View each solution as a **string** (analogy with **DNA**)
- Maintain a **population of good solutions**
- Allow **random mutations** of single characters of individual solutions
- **Combine two solutions** by taking part of one and part of another (analogy with crossover in **sexual reproduction**)
- Get rid of solutions that have the worst values and make multiple copies of solutions that have the best values (analogy with **natural selection** -- survival of the fittest).

*Usually very slow. In the rare cases when they produce answers with better objective function values than other methods they tend to produce very **brittle** solutions – that are very bad with respect to small changes to the requirements.*

# Deep Neural Nets and NP-hardness?

- **Artificial neural networks**
  - based on very elementary model of human neurons
  - **Set up a circuit of artificial neurons**
    - each artificial neuron is an analog circuit gate whose computation depends on a set of **connection strengths**
  - **Train the circuit**
    - Adjust the connection strengths of the neurons by giving many positive & negative training examples and seeing if it behaves correctly
  - **The network is now ready to use**

*Despite their wide array of applications, they have not been shown to be useful for NP-hard problems.*

# Quantum Computing and NP-hardness?

Use physical processes at the quantum level to implement “weird” kinds of circuit gates based on unitary transformations

- Quantum objects can be in a “superposition” of many pure states at once
  - Can have  $n$  objects together in a superposition of  $2^n$  states
- Each quantum circuit gate operates on the whole superposition of states at once
  - Inherent parallelism but classical randomized algorithms have a similar parallelism: *not enough on its own*
  - Advantage over classical: **copies interfere with each other.**
- Exciting direction - theoretically able to factor efficiently.  
*Major practical problems wrt errors, decoherence to be overcome.*
- *Small brute force improvement but unlikely to produce exponential advantage for NP.*