GTI

Approximation Algorithms

A. Ada, K. Sutner

Carnegie Mellon University

Spring 2018

1. Optimization Problems

- Traveling Salesman Problem

Minimizing Cost

There are lots of combinatorial problems that take the following form:

- a set I of instances
- a solution function $\mathsf{sol} : I \to \mathfrak{P}_0(\Sigma^\star)$
- a cost function $\mathsf{cost} : \mathsf{solutions} \to \mathbb{N}_+$

The optimal value associated with an instance x is

$$\mathsf{optval}(x) = \min\big(\, \mathsf{cost}(z) \mid z \in \mathsf{sol}(x) \,\big)$$

We are interested in finding an optimal solution, some $z \in \mathsf{sol}(x)$ such that $\mathsf{cost}(z) = \mathsf{optval}(x)$.

Note that optimal solutions need not be uniquely determined, though their value is.

If you are a stickler for precision, you might want to deal with the case $\mathsf{sol}(x) = \emptyset$: we can just set $\mathsf{optval}(x) = \infty$ and everything will work fine.

Of course, type theorists are now having a cow since $\infty \notin \mathbb{N}$. Relax, smell the flowers, have a single malt ...

It is perfectly fine that sol(x) is a very simple set with a trivial membership test.

The difficulty in finding a solution of optimal value is that sol(x) is exponentially large, and we have no direct way of identifying the cheap guys.

Typical Example: Vertex Cover

Here sol$(G) =$ all vertex covers. Given a candidate set $C \subseteq V$, it is trivial to check that C is a solution.

Letting cost$(C) = |C|$ we want a minimum cardinality VC.

To connect to the complexity class $\mathbb{NP}$ we consider a (slightly artificial) decision version:

Problem:	**Foobag Problem**
Instance:	Instance x, a bound β.
Question:	Is there a solution of cost at most β?

In other words, we are asking whether optval$(x) \leq \beta$?

Very often a fast solution to the decision version also produces a fast solution to the optimization problem: we can build an optimal solution in stages.

Typical Example: Vertex Cover

Experience shows that lots of these optimization problems are $\mathbb{NP}$-complete (more precisely: their decision versions are).

- vertex cover
- independent set
- clique
- longest path
- longest cycle

Note that some of these are actually maximization problems. Alternatively, we can cook up artificial cost functions and minimize: e.g., for independent set could use $\text{cost}(X) = n - |X|$.

Since we presumable cannot solve the decision version in polynomial time, it is natural to relax the requirements a bit: instead of finding an optimal solution, we will make do with $z \in \mathsf{sol}(x)$ such that

$$\mathsf{cost}(z) \leq k \cdot \mathsf{optval}(x)$$

where k is some fixed constant.

A polynomial time algorithm that produces such a solution is called a k-approximation algorithm for the problem.

Note that a 1-approximation algorithm corresponds to a perfect solution, and thus is unlikely to exist.

Theorem (Gavril, Yannakakis)

There is 2-approximation algorithm for Vertex Cover.

Proof.

The algorithm is infuriatingly simple

```
1 C = empty
2 while( some edge {u,v} is not covered )
3         add u, v to C
```

But we need a performance guarantee.

Note that endpoints of the edges in a maximal matching necessarily form a vertex cover: otherwise we could add an edge.

But clearly the Gavril/Yannakakis algorithm produces a maximal matching (though not necessarily a maximum one).

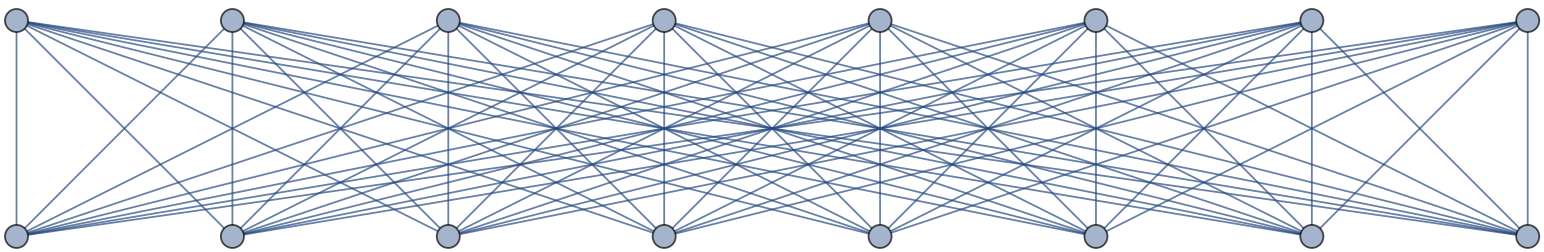
Proposition

optval$(G) \geq |M|$ *for any maximal matching.*

Clearly every vertex cover must contain at least one endpoint of every edge in any matching.

Done.

There is a simple scenario when our approximation algorithm produces a cover of size exactly twice the optimum: a complete bipartite graph.



The Gavril/Yannakakis algorithm is deceptively simple. Most algorithms people would probably try to optimize it a bit, along the lines of

```
1 C = empty
2 while( some edge is not covered )
3         find vertex x incident to  most uncovered edges
4         add x to C
```

Exercise

Figure out how good this "improved" approximation algorithm is (hint: not very).

Here is a much fancier way to get approximate solutions for vertex cover: use a powerful algorithm that solves the wrong problem, then fix things up.

First, an instance of Linear Programming (LP) expresses a minimization problem for n variables and m constraints, with a linear objective function.

More precisely, we have $A \in \mathbb{Z}^{m,n}$, $m \leq n$, $b \in \mathbb{Z}^m$ and a $c \in \mathbb{Z}^n$.

We want a real vector $x \in \mathbb{R}^n$ that

$$\begin{aligned} \text{minimize } z &= c \circ x \\ Ax &\geq b \\ x &\geq 0 \end{aligned}$$

The function $x \mapsto c \circ x = \sum c_i x_i$ is the objective function.

This is an LP in canonical form.

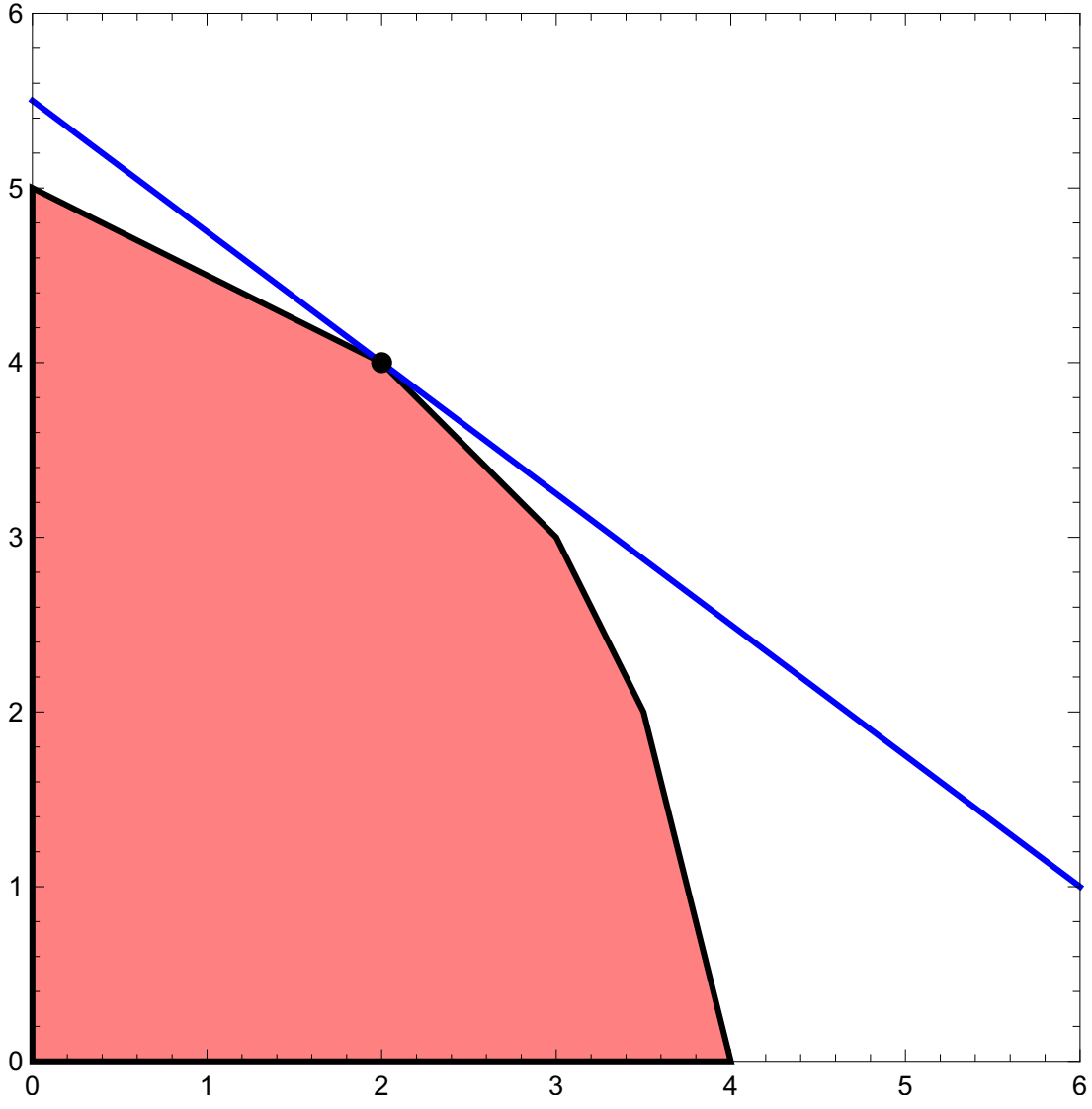
For canonical form LP's there is a natural geometric interpretation.

$$P = \{\, x \in \mathbb{R}^n \mid Ax \geq b \wedge x \geq 0 \,\}$$

is a convex polytope in n-dimensional space and contained in the first orthant.

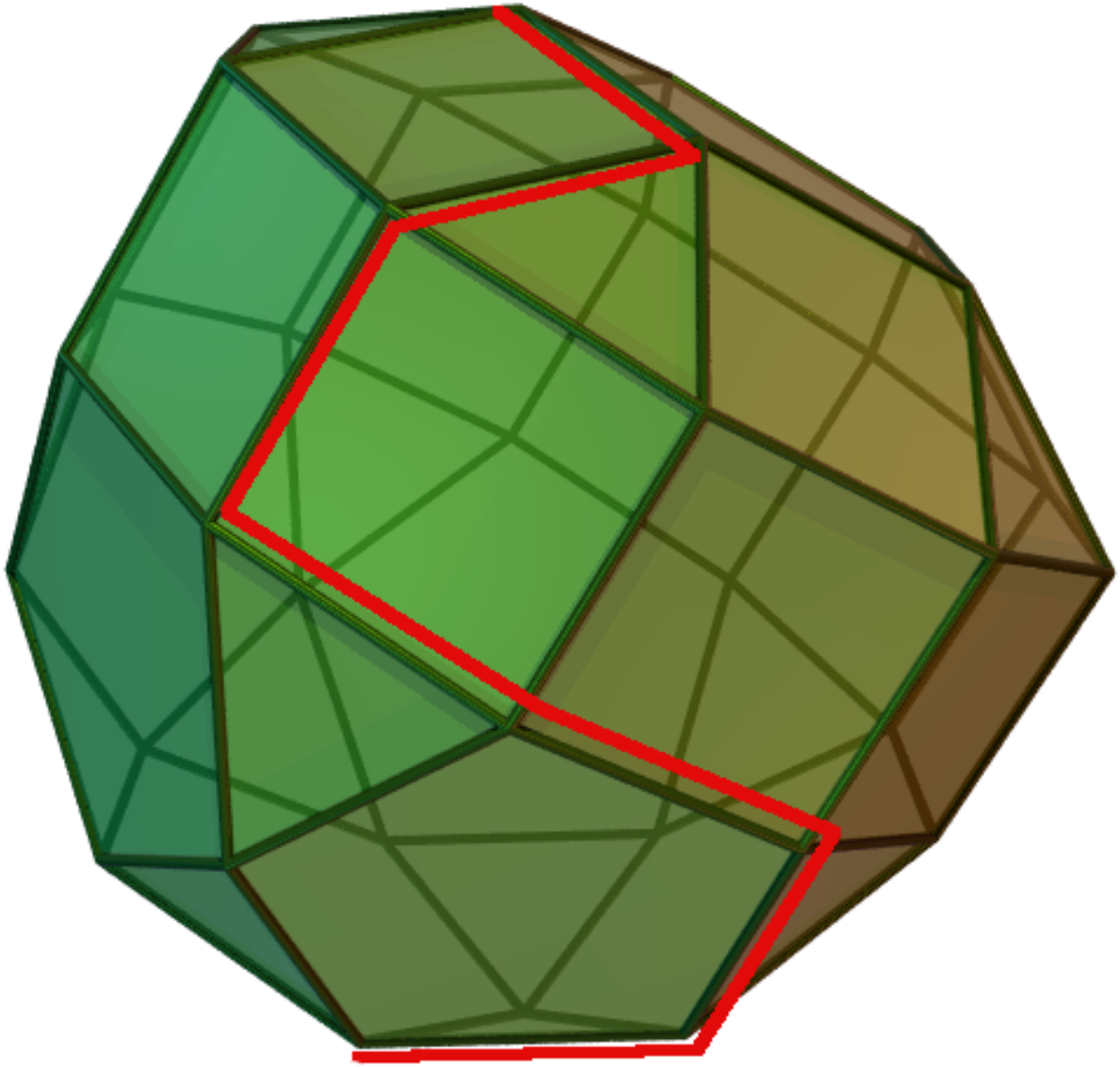
This is called the set of feasible solutions or the simplex. For any number d the set $\{\, x \in \mathbb{R}^n \mid c \circ x = d \,\}$ is a hyperplane perpendicular to c.

Thus we have to find the first point in P where a hyperplane perpendicular to c intersects P (if it is moved from infinity towards the simplex in the appropriate direction).


6
5
4
3
2
1
0
0
1
2
3
4
5
6

Simplex Algorithm There is a famous algorithm due to George Dantzig from 1947, arguably one of the most important algorithms period. It works well in most cases, but is exponential in the worst case. It is polynomial for some notion of average case.

Karmarkar's Algorithm Invented in 1984, an interior point method that is guaranteed to be polynomial time.



Alas, when we restrict the variables to be integral, $x \in \mathbb{Z}^n$, things turn sour: the corresponding Integer Programming problem is $\mathbb{NP}$-hard.

But IP is quite expressive and really one of the goto hard problems in $\mathbb{NP}$.

Also note that hardness for IP is not difficult to show, it was on the list of Karp's 21 problems.

But membership in $\mathbb{NP}$ requires a bit of work: we have to make sure that a solution does not require an absurd number of digits to write down.

Who Cares?

It turns out to be really easy to translate Vertex Cover into Integer Programming

Given $G = \langle V, E \rangle$ introduce a variable x_v for each vertex v.

Then write down some obvious constraints on the x_v and minimize their sum (which will turn out to be the size of a minimum cover).

A 0/1-Integer Programming Problem

Insist on $\boldsymbol{x} \in \mathbb{Z}^n$.

Minimize $\sum x_v$ subject to

$$x_u + x_v \geq 1 \qquad \{u, v\} \in E$$

$$0 \leq x_v \leq 1$$

Note that $C = \{\, v \mid x_v = 1 \,\}$ is a minimal vertex cover.

Great, but 0/1-Integer Programming is $\mathbb{NP}$-hard and we are going around in circles.

How about accepting a real LP solution $\boldsymbol{x} \in \mathbb{R}^n$, which somehow will produce a "fractional vertex cover" (of course, a priori fractions don't really make any sense).

So we may get solutions like $x_v = 1/3$, or $x_v = 7/8$.

Surprisingly,

$$C = \{ v \mid x_v \geq 1/2 \}$$

is a vertex cover, and has size at most twice the minimal one.

C clearly is a cover: $x_u + x_v \geq 1$ implies $x_u \geq 1/2$ or $x_v \geq 1/2$.

Write $\widehat{x}_v = 1$ whenever $x_v \geq 1/2$, and $\widehat{x}_v = 0$ otherwise.

$$\begin{aligned} |C| &= \sum \widehat{x}_v \\ &\leq 2 \cdot \sum x_v \\ &= 2 \cdot \mathsf{optval}_{LP} \\ &\leq 2 \cdot \mathsf{optval}_{IP} \\ &= 2 \cdot \mathsf{optval}_{VC} \end{aligned}$$

- Optimization Problems

2. Traveling Salesman Problem

Suppose we have cost function on the edges of a complete graph K_n.

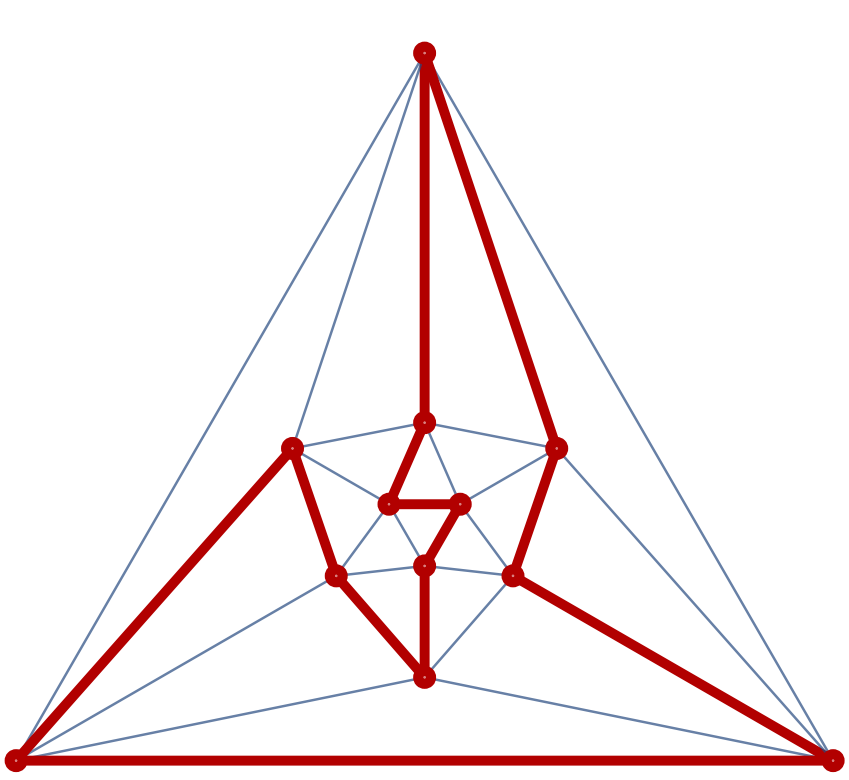
A tour of the graph is a permutation π of $[n]$: think of the cycle

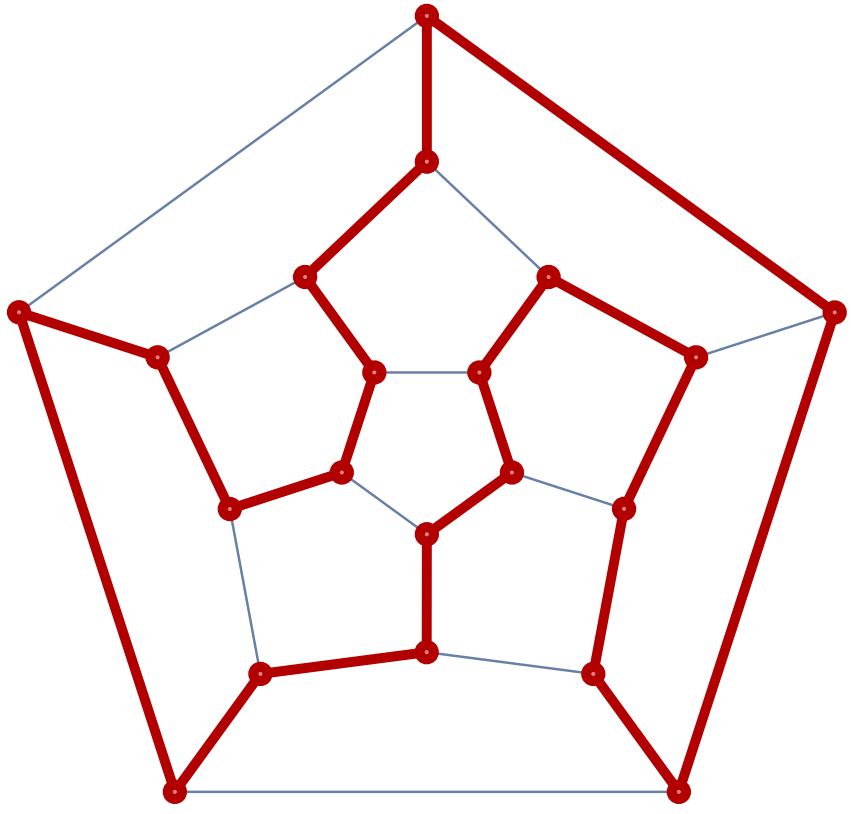
$$v_{\pi(1)}, v_{\pi(2)}, v_{\pi(3)}, \ldots, v_{\pi(n)}, v_{\pi(1)}$$

The cost of π is the sum of all the edge costs on the cycle.

Problem:	**Traveling Salesman Problem (TSP)**
Instance:	A cost function on the edges of K_n, a bound β.
Question:	Is there a tour of cost at most β?

Icosahedron and Dodecahedron





Cost is Euclidean distance if there is an edge, ∞ otherwise.

Albania to Spain

A variant where we leave out the last edge (that closes the cycle).

Theorem

TSP is $\mathbb{NP}$-complete.

Proof.

Reduction from Hamiltonian Cycle.

Suppose G is a ugraph on n points. Define a cost function on K_n as follows:

$$\text{cost}(e) = \begin{cases} 1 & \text{if } e \in E, \\ 2 & \text{otherwise.} \end{cases}$$

Then there is a tour of cost n iff G has a Hamiltonian cycle.

□

Lemma

There is no k-approximation algorithm for general TSP.

Proof.

Assume otherwise.

Again use Hamiltonian Cycle and let G be a ugraph on n points. Define a cost function on K_n as follows:

$$\text{cost}(e) = \begin{cases} 1 & \text{if } e \in E, \\ k \cdot n & \text{otherwise.} \end{cases}$$

Done. □

There are natural variants of TSP obtained by introducing more geometry:

Metric TSP cost is symmetric, and the triangle inequality holds:

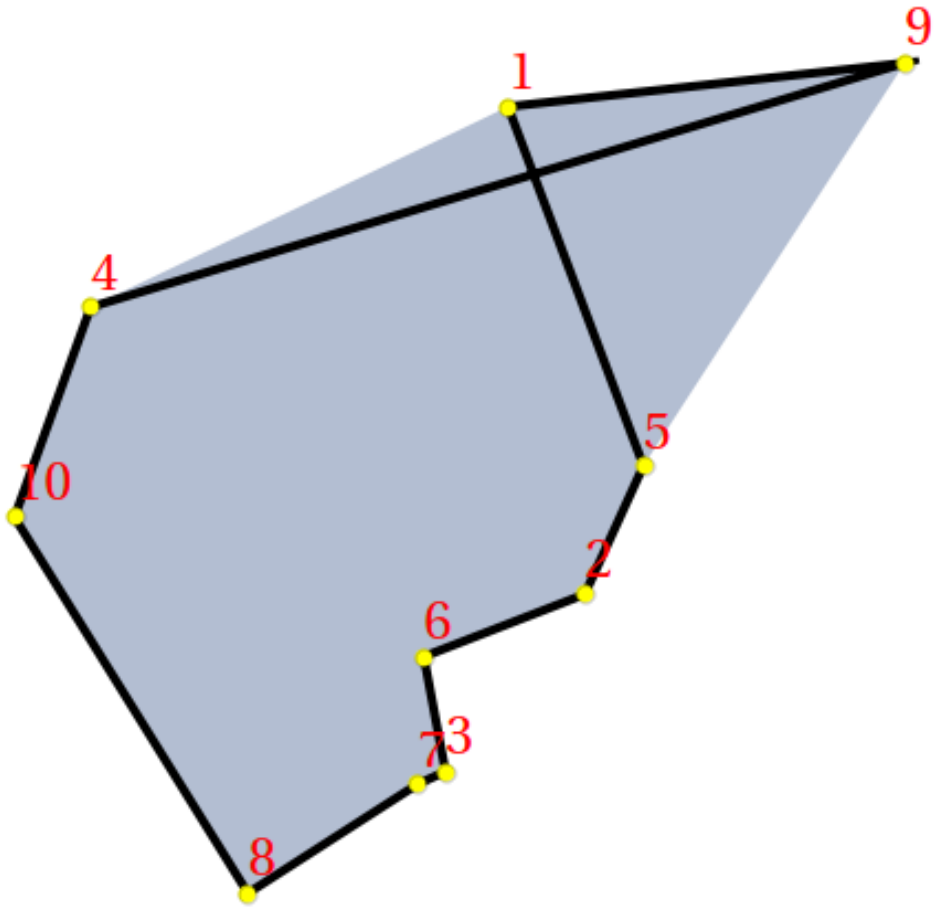
$$\text{cost}(x, y) \leq \text{cost}(x, z) + \text{cost}(z, y)$$

Euclidean TSP Vertices are points and distance is Euclidean distance.

These restrictions do not break $\mathbb{NP}$-hardness, but they make approximation algorithms easier. Note that membership in $\mathbb{NP}$ becomes problematic in the Euclidean setting.

Nearest Neighbor

Perhaps the most tempting strategy for a Metric TSP is to go greedy: start in some random place, then always go to the nearest untouched neighbor.



Unfortunately, the crossover between 1-5 and 4-9 is clearly wrong.

However, we can fix small problems like this one by a little post-processing: walk around the tour, and eliminate all crossovers. Clearly, this takes only polynomial time.

True, but this does not address global mistakes. With a little effort, one can make nearest-neighbor produce a catastrophically bad tour.

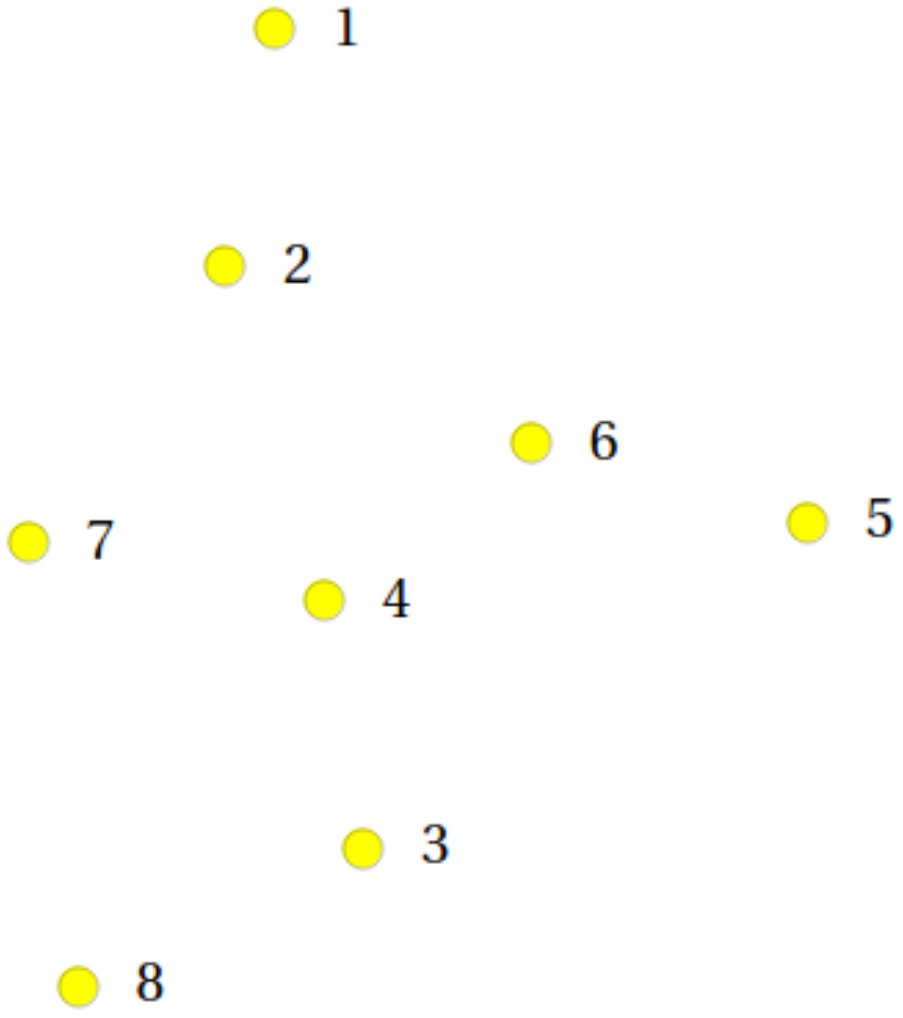
Proposition

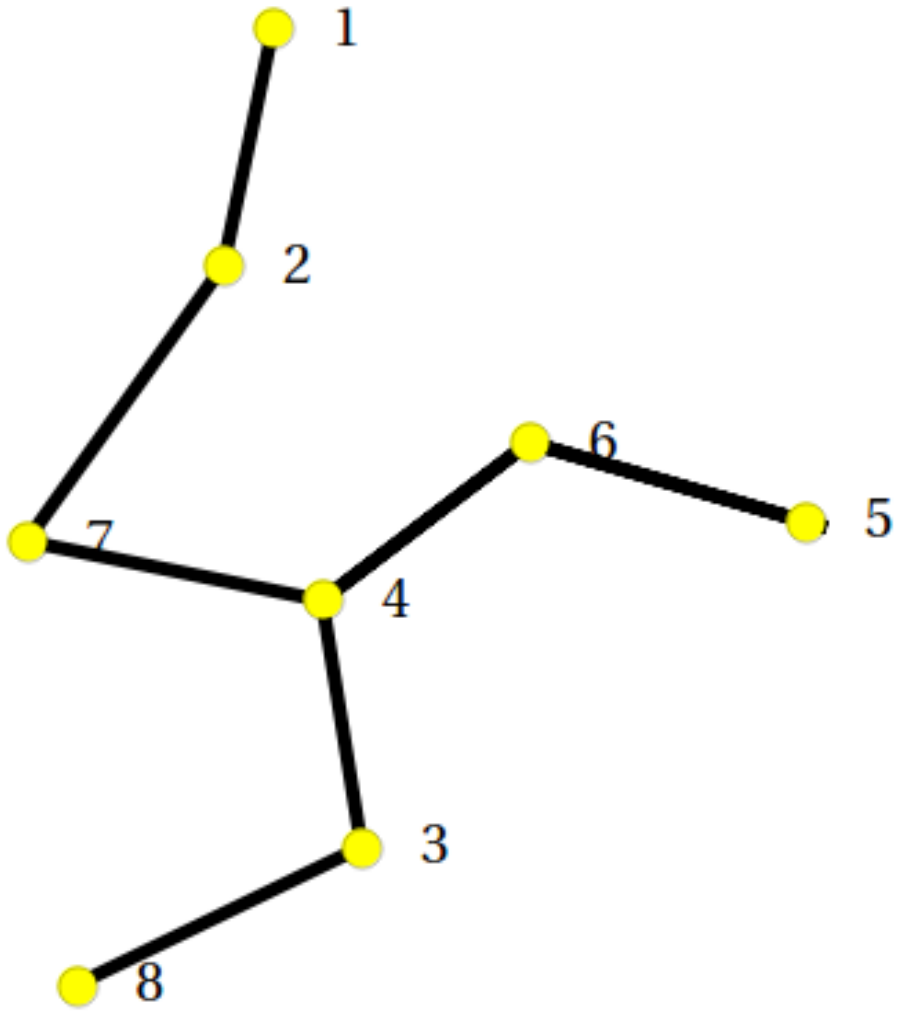
The nearest neighbor approach does not produce an k-approximation algorithm for any k.

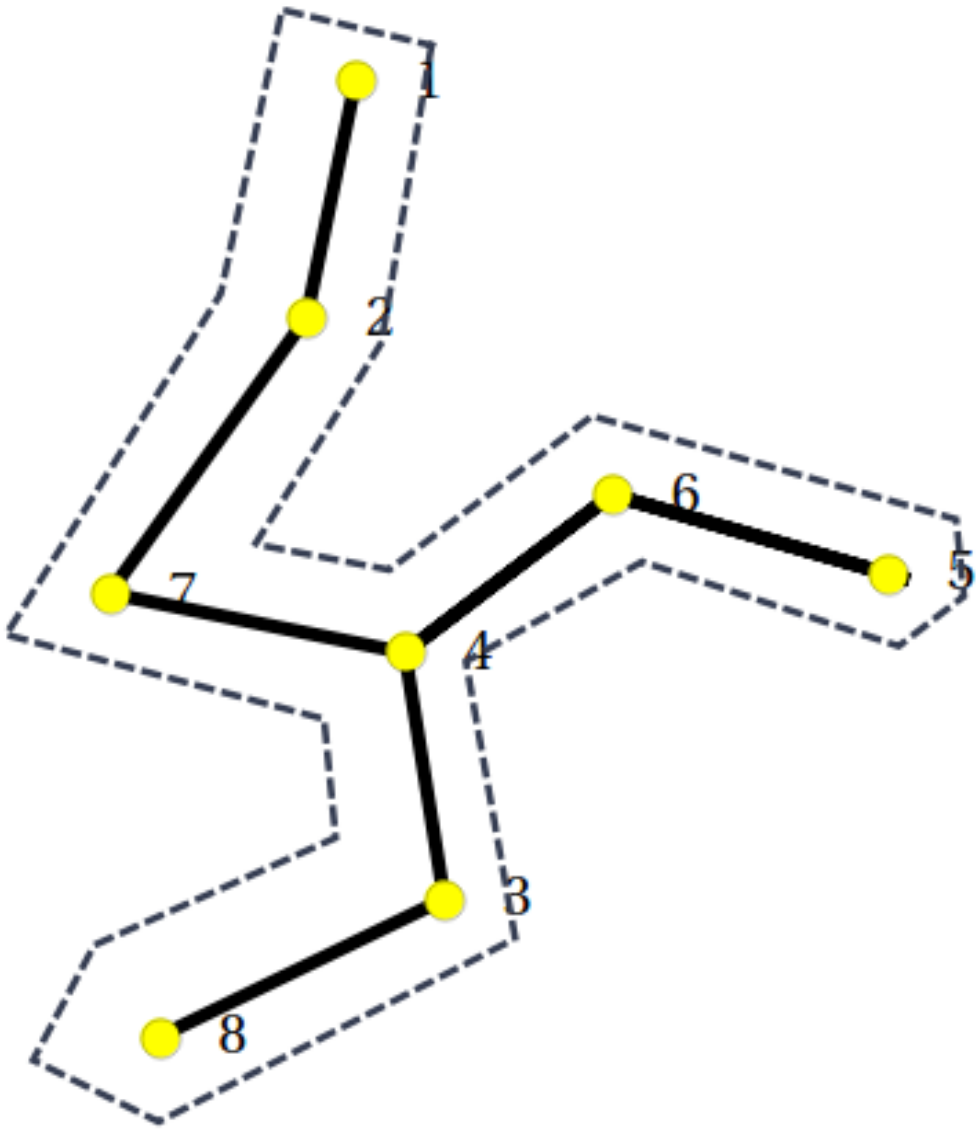
Here is a clever idea: we know that we can efficiently calculate minimum spanning trees. It is tempting to exploit a MST to buid a tour.

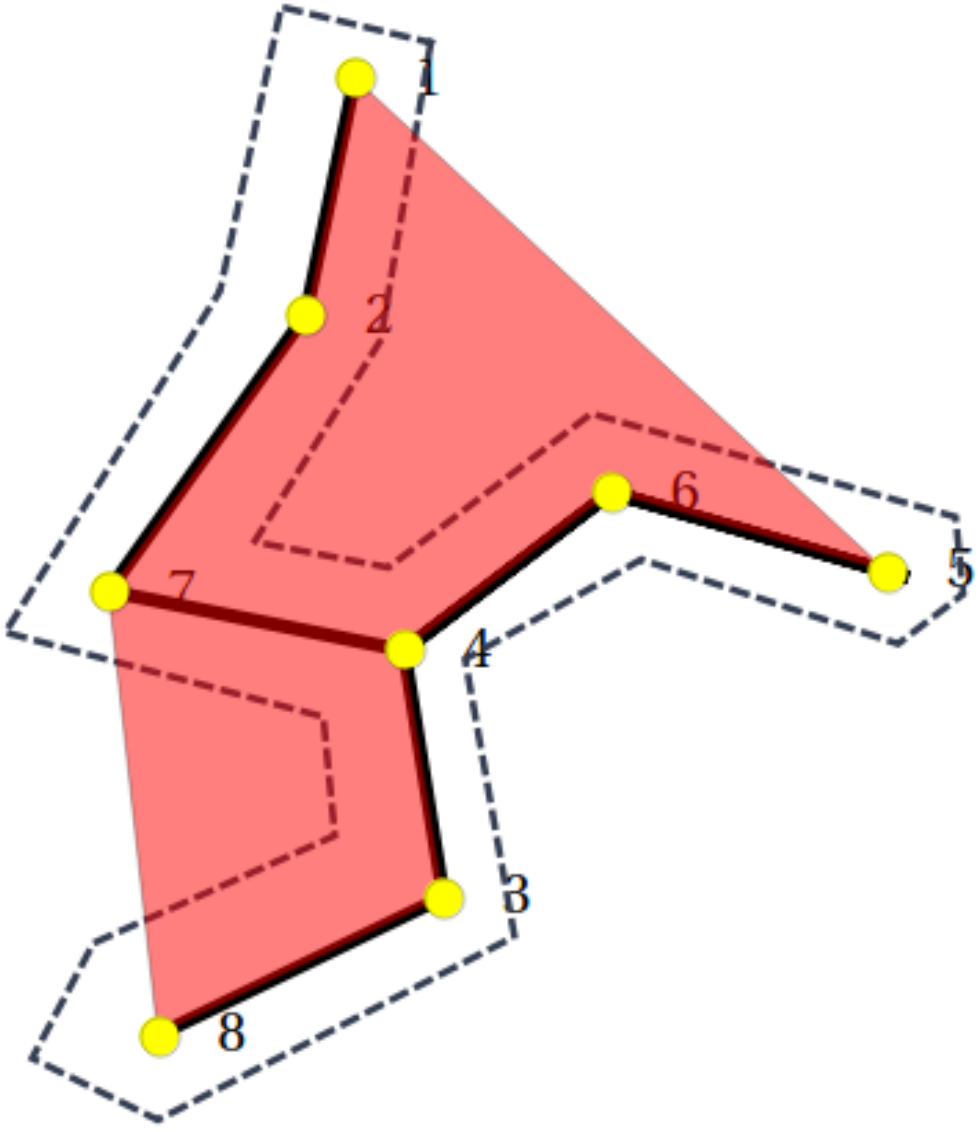
- Walk around the spanning tree, traversing each edge in the tree twice.
- Eliminate multiple occurrences of vertices by exploiting the triangle inequality.

In other words, we start with a cycle and wind up with a simple cycle of equal or better cost.


1
2
6
5
7
4
3
8


1
2
6
5
7
4
3
8


1
2
3
4
5
6
7
8


1
2
3
4
5
6
7
8

Theorem

Once-around-the-spanning-tree is a 2-approximation algorithm for Metric TSP.

Proof. Dropping one edge turns a tour into a spanning tree. □

Exercise

Explain exactly how to implement the contraction phase of the algorithm. What is the running time of your algorithm?

Doubling the edges in T essentially turns T into an Eulerian (multi-)graph. A better way to do this is to only add edges to the odd-degree points.

There is an even number of such points V_{odd}. We want a minimum cost matching for V_{odd}.

Claim

The cost of such a matching is at most $1/2$ the cost of an optimal tour.

Theorem

There is a $\frac{3}{2}$-approximation algorithm for metric TSP.

As it turns out, $3/2$ is not the end of the story.

Arora and Mitchell have constructed $(1+\varepsilon)$-approximation algorithms for Metric TSP. Note, though, that the running time increases when ε decreases.

Needless to say, these algorithms are more complicated and their analysis requires major work.

In the classical theory of computation, theorems are simply consequences of the axioms (Peano, or some fragment of set theory). Lots and lots of separation results are known, we basically understand the lay of the land. Typical example: semidecidable sets that lie strictly between Halting and decidable.

Theorem (Friedberg, Muchnik 1956/7)

There are intermediate semidecidable sets: $\emptyset <_T A <_T K$.

The proof is absolutely beautiful and very intricate. Unfortunately, it produces completely artificial examples.

Theorem (Ladner 1975)

If $\mathbb{P} \neq \mathbb{NP}$, then there are intermediate problems wrto polynomial time reducibility.

The proof is quite similar to the Friedberg/Muchnik construction and produces an entirely artificial example of an intermediate problem.

Alas, we currently have little hope to get rid of the annoying conditional: if such-and-such separation result holds, then such-and-such claim is true.

It's your job to remove the training wheels and produce unconditional results.

Obviously, if $\mathbb{P} = \mathbb{NP}$, then every $\mathbb{NP}$ problem has a 1-approximation algorithm.

- For Vertex Cover, $k = 2$ is quite easy. With effort we can get $2 = \Theta(1/\sqrt{\log n})$. But $k < 1.36$ collapses $\mathbb{P}$ and $\mathbb{NP}$.
- For TSP, $k = 3/2$ is not too hard. With effort, we can get arbitrarily close to $k = 1$. The only collapse would be $k = 1$.

The Maximum Coverage Problem is the following.

Given a collection $S_1, S_2, \ldots, S_m$ of subsets of $[n]$ and a bound β, maximize $|\bigcup_{i \in I} S_i|$ where $I \subseteq [m]$ and $|I| = \beta$.

Theorem

The Maximum Coverage Problem is $\mathbb{NP}$-complete.

Note that this time the optimal solution will have larger "cost."

There is a natural greedy algorithm for MCP: always try to hit as many uncovered elements of $[n]$ as possible.

```
for  i = 1 .. beta
     pick the set that covers the
          largest number of uncovered elements
```

Theorem

MCP admits a $(1 - 1/e)$-approximation algorithm.

Unless $\mathbb{P} = \mathbb{NP}$, this is the best we can do: any approximation algorithm better than $(1 - 1/e) \approx 0.632121$ would already collapse the two complexity classes.

Note that MCP is quite similar to Set Cover:

Again there is a collection $S_1, S_2, \ldots, S_m$ of subsets of $[n]$ and a bound β, but this time $\bigcup S_i = [n]$ and we want to find $I \subseteq [m]$ of cardinality at most β such that $|\bigcup_{i \in I} S_i = [n]|$.