Lecture 3

In which we prove the equivalence of three versions of the Traveling Salesman Problem, we provide a 2-approximate algorithm, we review the notion of Eulerian cycle, we think of the TSP as the problem of finding a minimum-cost connected Eulerian graph, and we revisit the 2-approximate algorithm from this perspective.

1 Variations of the Traveling Salesman Problem

Recall that an input of the Traveling Salesman Problem is a set of points X and a non-negative, symmetric, distance function $d : X \times X \to \mathbb{R}$ such that $d(x, y) = d(y, x) \geq 0$ for every $x, y \in X$.

The goal is to find a cycle $C = v_0 \to v_1 \to v_2 \to \cdots v_{m-1} \to v_m = v_0$ that reaches every vertex and that has minimal total length $cost_d(C) := \sum_{i=0}^{m-1} d(v_i, v_{i+1})$.

There are different versions of this problem depending on whether we require $d(\cdot, \cdot)$ to satisfy the triangle inequality or not, and whether we allow the loop to pass through the same point more than once.

1. *General TSP without repetitions* (General TSP-NR): we allow arbitrary symmetric distance functions, and we require the solution to be a cycle that contains every point exactly once;
2. *General TSP with repetitions* (General TSP-R): we allow arbitrary symmetric distance functions, and we allow all cycles as an admissible solution, even those that contain some point more than once;
3. *Metric TSP without repetitions* (Metric TSP-NR): we only allow inputs in which the distance function $d(\cdot, \cdot)$ satisfies the triangle inequality, that is

$$\forall x, y, z \in X.\ d(x, z) \leq d(x, y) + d(y, z)$$

and we only allow solutions in which each point is reached exactly once;

4. *Metric TSP with repetitions* (Metric TSP-R): we only allow inputs in which the distance function satisfies the triangle inequality, and we allow all cycles as admissible solutions, even those that contain some point more than once.

If we allow arbitrary distance functions, and we require the solution to be a cycle that reaches every point exactly once, then we have a problem for which no kind of efficient approximation is possible.

Fact 1 *If* $\mathbf{P} \neq \mathbf{NP}$ *then there is no polynomial-time constant-factor approximation algorithm for General TSP-NR.*

More generally, if there is a function $r : \mathbb{N} \to \mathbb{N}$ *such that* $r(n)$ *can be computable in time polynomial in* n *(for example,* $r(n) = 2^{100} \cdot 2^{n^2}$*), and a polynomial time algorithm that, on input an instance* (X, d) *of General TSP-NR with* n *points finds a solution of cost at most* $r(n)$ *times the optimum, then* $\mathbf{P} = \mathbf{NP}$*.*

The other three versions, however, are completely equivalent from the point of view of approximation and, as we will see, can be efficiently approximated reasonably well.

Lemma 2 *For every* $c \geq 1$*, there is a polynomial time* c*-approximate algorithm for Metric TSP-NR if and only if there is a polynomial time* c*-approximate approximation algorithm for Metric TSP-R. In particular:*

1. *If* (X, d) *is a Metric TSP input, then the cost of the optimum is the same whether or not we allow repetitions.*
2. *Every* c*-approximate algorithm for Metric TSP-NR is also a* c*-approximate algorithm for Metric TSP-R.*
3. *Every* c*-approximate algorithm for Metric TSP-R can be turned into a* c*-approximate algorithm for Metric TSP-NR after adding a linear-time post-processing step.*

PROOF: Let $opt_{TSP-R}(X, d)$ be the cost of an optimal solution for (X, d) among all solutions with or without repetitions, and $opt_{TSP-NR}(X, d)$ be the cost of an optimal solution for (X, d) among all solutions without repetitions. Then clearly in the former case we are minimizing over a larger set of possible solutions, and so

$$opt_{TSP-R}(X, d) \leq opt_{TSP-NR}(X, d)$$

Consider now any cycle, possibly with repetitions, $C = v_0 \to v_1 \to v_2 \to \cdots v_{m-1} \to v_m = v_0$. Create a new cycle C' from C by removing from C all the repeated occurrences of any vertex. (With the special case of v_0 which is repeated at the end.) For

example, the cycle $C = a \to c \to b \to a \to d \to b \to a$ becomes $a \to c \to b \to d \to a$. Because of the triangle inequality, the total length of C' is at most the total length of C, and C' is a cycle with no repetitions. If we apply the above process by taking C to be the optimal solution of (X, d) allowing repetitions, we see that

$$opt_{TSP-R}(X, d) \geq opt_{TSP-NR}(X, d)$$

and so we have proved our first claim that $opt_{TSP-R}(X, d) = opt_{TSP-NR}(X, d)$.

Regarding the second claim, suppose that we have a c-approximate algorithm for Metric TSP-NR. Then, given an input (X, d) the algorithm finds a cycle C with no repetitions such that

$$cost_d(C) \leq c \cdot opt_{TSP-NR}(X, d)$$

but C is also an admissible solution for the problem Metric TSP-R, and

$$cost_d(C) \leq c \cdot opt_{TSP-NR}(X, d) = c \cdot opt_{TSP-R}(X, d)$$

and so our algorithm is also a c-approximate algorithm for opt_{TSP-NR}.

To prove the third claim, suppose we have a c-approximate algorithm for Metric TSP-R. Then, given an input (X, d) the algorithm finds a cycle C, possibly with repetitions, such that

$$cost_d(C) \leq c \cdot opt_{TSP-R}(X, d)$$

Now, convert C to a solution C' that has no repetitions and such that $cost_d(C') \leq cost_d(C)$ as described above, and output the solution C'. We have just described a c-approximate algorithm for Metric TSP-NR, because

$$cost_d(C') \leq cost_d(C) \leq c \cdot opt_{TSP-R}(X, d) = c \cdot opt_{TSP-NR}(X, d)$$

□

Lemma 3 *For every $c \geq 1$, there is a polynomial time c-approximate algorithm for Metric TSP-NR if and only if there is a polynomial time c-approximate approximation algorithm for General TSP-R. In particular:*

1. *Every c-approximate algorithm for General TSP-R is also a c-approximate algorithm for Metric TSP-R.*

2. *Every c-approximate algorithm for Metric TSP-R can be turned into a c-approximate algorithm for General TSP-R after adding polynomial-time pre-processing and post-processing steps.*

PROOF: The first claim just follows from the fact that General and Metric TSP-R are the same problem, except that in the general problem we allow a larger set of admissible inputs.

The proof of the second claim is similar to the proof that an approximation algorithm for Metric Steiner Tree can be converted to an approximation algorithm for General Steiner Tree.

Consider the following algorithm: on input an instance (X, d) of General TSP-R, we compute the new distance function $d'(\cdot, \cdot)$ such that $d'(x, y)$ is the length of a shortest path from x to y in a weighted graph that has vertex set X and weights $d(\cdot, \cdot)$. Note that $d'(\cdot, \cdot)$ is a distance function that satisfies the triangle inequality. We also compute the shortest path between any pair x, y.

We then pass the input (X, d') to our c-approximate algorithm for Metric TSP-R, and find a cycle C' such that

$$cost_{d'}(C') \leq c \cdot opt_{TSP-R}(X, d')$$

Note that, for every pair of points (x, y), we have $d'(x, y) \leq d(x, y)$ and so this implies that

$$opt_{TSP-R}(X, d') \leq opt_{TSP-R}(X, d)$$

Finally, we construct a cycle C by replacing each transition $x \rightarrow y$ in C' by the shortest path from x to y according to $d(\cdot, \cdot)$. Because of the definition of $d'(\cdot, \cdot)$, we have

$$cost_d(C) = cost_{d'}(C')$$

and, combining the inequalities we have proved so far,

$$cost_d(C) \leq c \cdot opt_{TSP-R}(X, d)$$

meaning that the algorithm that we have described is a c-approximate algorithm for General TSP-R. □

2 A 2-approximate Algorithm

When discussing approximation algorithms for the Minimum Steiner Tree problem in the last lecture, we proved (without stating it explicitly) the following result.

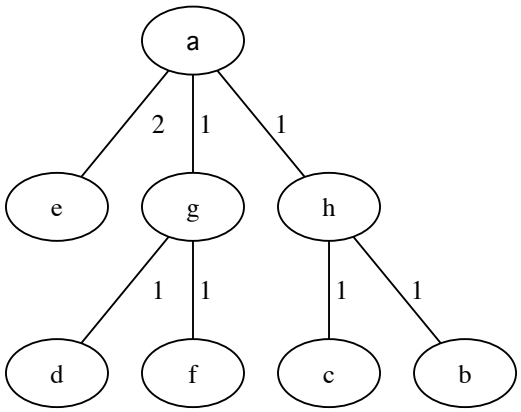
Lemma 4 *Let $T(X, E)$ be a tree over a set of vertices X, and $d : X \times X \to \mathbb{R}_{\geq 0}$ a symmetric distance function. Then there is a cycle $C = v_0 \to v_1 \to \cdots \to v_m = v_0$ that reaches every vertex at least once, and such that*

$$cost_d(C) = 2 \cdot cost_d(T)$$

where $cost_d(C) = \sum_{i=0,\ldots m-1} d(v_i, v_{i+1})$ and $cost_d(T) = \sum_{(x,y)\in E} d(x, y)$.

The proof is simply to consider a DFS visit of the tree, starting at the root; we define C to be the order in which vertices are visited, counting both the beginning of the recursive call in which they are reached, and also the time when we return at a vertex after the end of a recursive call originated at the vertex.

For example, revisiting an example from the last lecture, from the tree



We get the cycle $a \to e \to a \to g \to d \to g \to f \to g \to a \to h \to c \to b \to h \to a$, in which every point is visited at least once (indeed, a number of times equal to its degree in the tree) and every edge is traversed precisely twice.

Theorem 5 *There is a polynomial-time 2-approximate algorithm for General TSP-R. (And hence for Metric TSP-NR and Metric TSP-R.)*

PROOF: The algorithm is very simple: on input a (X, d) we find a minimum spanning tree T of the weighted graph with vertex set X and weights d, and then we find the cycle C of cost $2cost_d(T)$ as in Lemma 4.

It remains to prove that $opt_{TSP-R}(X,d) \geq opt_{MST}(X,d) = cost_d(T)$, which will then imply that we have found a solution whose cost is $\leq 2 \cdot opt_{TSP-R}(X,d)$, and that our algorithm is 2-approximate.

Let C^* be an optimal solution for (X,d), and consider the set of edges E^* which are used in the cycle, then (X,E^*) is a connected graph; take any spanning tree $T' = (X,E')$ of the graph (X,E^*), then

$$cost_d(T') \leq cost_d(C^*) = opt_{TSP-R}(X,d)$$

because T' uses a subset of the edges of C^*. On the other hand, T' is a spanning tree, and so

$$cost_d(T') \geq opt_{MST}(X,d)$$

So we have

$$opt_{TSP-R}(X,d) \geq opt_{MST}(X,d)$$

from which it follows that our algorithm achieves a 2-approximation.□

3 Eulerian Cycles

In this section we review the definition of Eulerian cycle. In the next section, we will use this notion to give a new view of the 2-approximate algorithm of the previous section, and we willnote that this new perspective suggests a potentially better algorithm, that we will analyze in the next lecture.

In this section, it will be convenient to work with *multi-graphs* instead of graphs. In an undirected multi-graph $G = (V,E)$, E is a multi-set of pairs of vertices, that is, the same pair (u,v) can appear more than once in E. Graphically, we can represent a multi-graph in which there are k edges between u and v by drawing k parallel edges between u and v. The degree of a vertex in a multigraph is the number of edges of the graph that have that vertex as an endpoint, counting multiplicities.

Definition 6 (Eulerian Cycle) *An Eulerian cycle in a multi-graph $G = (V,E)$ is a cycle $v_0 \to v_1 \to \cdots \to v_m = v_0$ such that the number of edges (u,v) in E is equal to the number of times (u,v) is used in the cycle.*

In a standard graph, a Eulerian cycle is a cycle that uses every edge of the graph exactly once.

Theorem 7 *A multi-graph $G = (V, E)$ has an Eulerian cycle if and only if every vertex has even degree and the vertices of positive degree are connected. Furthermore, there is a polynomial time algorithm that, on input a connected graph in which every vertex has even degree, outputs an Eulerian cycle.*

PROOF: If G is Eulerian, then the cycle gives a way to go from every vertex to every other vertex, except the vertices of zero degree. Furthermore, if a vertex v appears k times in the cycle, then there are $2k$ edges involving v in the cycle (because, each time v is reached, there is an edge used to reach v and one to leave from v); since the cycle contains all the edges of the graph, it follows that v has degree $2k$, thus all vertices have even degree. This shows that if a graph contains an Eulerian cycle, then every vertex has even degree and all the vertices of non-zero degree are connected.

For the other direction, we will prove by induction a slightly stronger statement, that is we will prove that if G is a graph in which every vertex has even degree, then every connected component of G with more than one vertex has a Eulerian cycle. We will proceed by induction on the number of edges.

If there are zero edges, then every connected component has only one vertex and so there is nothing to prove. This is the base case of the induction.

If we have a graph $G = (V, E)$ with a non-empty set of edges and in which every vertex has even degree, then let $V_1, \ldots, V_m$ be the connected components of G that have at least two vertices. If $m \geq 2$, then every connected component has strictly fewer vertices than G, and so we can apply the inductive hypothesis and find Eulerian cycles in each of $V_1, \ldots, V_m$.

It remains to consider the case in which the set V' of vertices of positive degree of G are all in the same connected component. Let $G' = (V', E')$ be the restriction of G to the vertices of V'. Since every vertex of G' has degree ≥ 2, there must be a cycle in G'. This is because if a connected graph with n vertices has no cycles, then it is a tree, and so it has $n - 1$ edges; but in a graph in which there are n vertices and every vertex has degree ≥ 2, the number of edges is at least $\frac{1}{2} \cdot 2 \cdot n = n$. Let C be a simple cycle (that is, a cycle with no vertices repeated) in G', and let G'' be the graph obtained from G' by removing the edges of C. Since we have removed two edges from every vertex, we have that G'' is still a graph in which every vertex has even degree. Since G'' has fewer edges than G' we can apply the induction hypothesis, and find a Eulerian cycle in each non-trivial connected component (a connected component is trivial if it contains only an isolated vertex of degree zero) of G''. We can then patch together these Eulerian cycles with C as follows: we traverse C, starting from any vertex; the first time we reach one of the non-trivial connected components of G'', we stop traversing C, and we traverse the Eulerian cycle of the component, then continue on C, until we reach for the first time one of the non-trivial connected components of G'' that we haven't traversed yet, and so on. This describes a Eulerian path into all of G'

Finally, we note that this inductive argument can be converted into a recursive algorithm. The main computation is to find the connected components of a graph, which can be done in linear time, and to find a cycle in a given graph, which can also be done in linear time using a DFS. Hence the algorithm runs in polynomial time.□

4 Eulerian Cycles and TSP Approximation

Let (X, d) be an instance of General TSP-R. Suppose that $G = (X, E)$ is a connected multi-graph with vertex set X that admits an Eulerian cycle C. Then the Eulerian cycle C is also an admissible solution for the TSP problem, and its cost is $\sum_{(u,v)\in E} d(u, v)$. Conversely, if we take any cycle which is a TSP-R solution for the input (X, d), and we let E be the multiset of edges used by the cycle (if the cycle uses the same edge more than once, we put as many copies of the edge in E as the number of times it appears in the cycle), then we obtain a graph $G = (X, E)$ which is connected and which admits an Eulerian cycle.

In other words, we can think of the General TSP-R as the following problem: given a set of points X and a symmetric distance function $d(\cdot, \cdot)$, find the multi-set of edges E such that the graph $G = (V, E)$ is connected and Eulerian, and such that $\sum_{(u,v)\in E} d(u, v)$ is minimized.

The approach that we took in our 2-approximate algorithm was to start from a spanning tree, which is guaranteed to be connected, and then *take every edge of the spanning tree twice*, which guarantees that every vertex has even degree, and hence that an Eulerian cycle exists. The reader should verify that if we take a tree, double all the edges, and then apply to the resulting multigraph the algorithm of Theorem 7, we get the same cycle as the one obtained by following the order in which the vertices of the tree are traversed in a DFS, as in the proof of Lemma 4.

From this point of view, the 2-approximate algorithm seems rather wasteful: once we have a spanning tree, our goal is to add edges so that we obtain an Eulerian graph in which every vertex has even degree. Doubling every edge certainly works, but it is a rather "brute force" approach: for example if a vertex has degree 11 in the tree, we are going to add another 11 edges incident on that vertex, while we could have "fixed" the degree of that vertex by just adding one more edge. We will see next time that there is a way to implement this intuition and to improve the factor of 2 approximation.