Chapter 63

Introduction

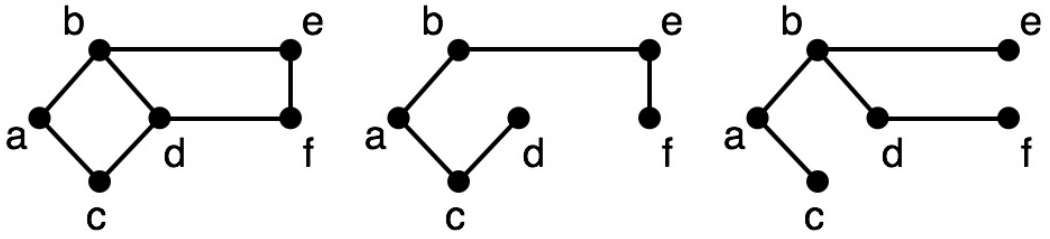
This chapter defines the Minimum Spanning Tree (MST) problem and introduces a key lemma called the "light-edge lemma" that nearly all algorithms for solving the problem utilizes.

1 Spanning Trees

Recall that we say that an undirected graph is a forest if it has no cycles and a tree if it is also connected. Given a connected, undirected graph, we might want to identify a subset of the edges that form a tree, while including all the vertices. We call such a tree a spanning tree.

Definition 63.1 (Spanning Tree)**.** For a connected undirected graph $G = (V, E)$, a spanning tree is a tree $T = (V, E')$ with $E' \subseteq E$.

Example 63.1. A graph (top), and two spanning trees for it.



Exercise 63.1. How many edges does a spanning tree have?

Solution. A graph can have many spanning trees, but all have $|V|$ vertices and $|V| - 1$ edges.

Lemma 63.1 (Spanning Trees Edge Replacement)**.** Let $G = (V, E)$ be a connected graph and let T be a spanning tree of G. Consider some edge $e = \{u, v\} \in E$ that is not in T. Let e' be any edge on the path from u to v in T and let the tree T' be $T \setminus \{e'\} \cup \{e\}$, that is a tree obtained by swapping e for e'. The tree T' is a spanning tree of G.

Proof. Consider any path in T that uses $e' = \{u', v'\}$. We can re-route this path to use $e = \{u, v\}$ instead and thus the path is a valid path in T'. Thus, T' is connected, and is acyclic. Furthermore T' has exactly the same number of nodes and edges as T and thus is a spanning tree. □

Sequential Algorithms for Spanning Trees. We can find a spanning tree of a graph by using graph search.

- A DFS-tree is a spanning tree, because it includes a path from the source to all the vertices in the graph.
- Similarly, we can construct a spanning tree based on BFS by including in the spanning tree each edge that leads to the discovery of an unvisited vertex.

DFS and BFS are work-efficient algorithms for computing spanning trees but as their span can be large. DFS in particular is sequential. The span of BFS can be as large as the diameter of the graph, which can be large.

Parallel Algorithms for Spanning Trees. We can compute a spanning tree for a graph by using graph contraction and, specifically star contraction . The idea is to use star contraction and add all the edges that are selected to define the stars to the spanning tree. Because graph contraction has poly-logarithmic span in expectation, this approach yields a good parallel algorithm, though work can be suboptimal.

Exercise 63.2. Give an algorithm for computing the spanning tree of a graph using star contraction. Prove that the algorithm is correct.

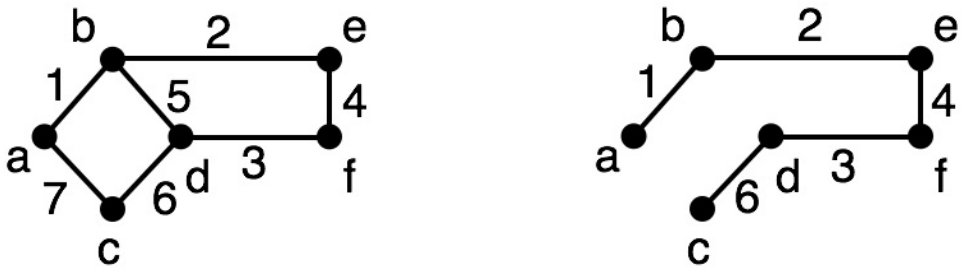
2 Minimum Spanning Trees

A graph can have many spanning trees. In some cases, such as in weighted graphs, we may be interested in finding the spanning tree with the smallest total weight (i.e., sum of the weights of its edges).

Definition 63.2 (Minimum Spanning Trees)**.** Given a connected, undirected weighted graph $G = (V, E, w)$, the minimum (weight) spanning tree (MST) problem requires finding a spanning tree of minimum weight, where the weight of a tree T is defined as

$$w(T) = \sum_{e \in E(T)} w(e).$$

Example 63.2. A graph (top) and its MST (bottom).



Example 63.3 (Network Design)**.** Minimum spanning trees have many interesting applications in network design, i.e., in the design of a network that includes vertices and connections between them. In such network design problems, it can be important to minimize some cost function, defined in terms of the connections in the network. As an example, suppose that you are wiring a building so that all the rooms are connected via bidirectional communication wires. Suppose that you can connect any two rooms at the cost of the wire connecting the rooms, which depends on the specifics of the building and the rooms but is always a positive real number. We can represent this problem as a minimization problem over a graph, where vertices represent rooms and weighted edges represent possible connections and their cost, attached to the graph as weights. To minimize the cost of the wiring, we can find a minimum spanning tree of the graph. One of the algorithms that we cover in this chapter (Boruvka's algorithm) was discovered when developing the electric network for the historical country of Moravia, which is today part of Czech Republic.

Distinct Edge-Weights Assumption. Throughout the discussion of minimum spanning trees, we assume that all edges of graphs have distinct weights. This assumption causes no loss-of-generality, because we can always break ties between edges by ordering them arbitrarily as long as the ordering is deterministic. One way to achieve this is to order edges based on their end-points or assign a unique label to each and break ties by comparing the labels. Such an ordering can be done "statically" ahead of time before running our favorite MST algorithm or "dynamically" as the algorithm runs. Another way is to tweak the edge weights to ensure uniqueness of the weights without altering the ordering of edges with distinct weights, though arguably this approach is rather clumsy from a practical point of view.

Lemma 63.2 (MST Edge Replacement)**.** Let $G = (V, E)$ be a weighted graph and let T be an MST for G. Let $e = \{u, v\} \in E$ be an edge that is not in T. Then e is heavier than any edge e' on the path between u and v in T.

Proof. By Edge-Replacement Lemma for spanning trees we know that replacing e' with e yields a spanning tree T'. If e lighter than e', T' is ligther than T and thus T cannot be an MST; a contradiction. □

Exercise 63.3. Show that for any undirected connected graph with unique edge weights, there exists one unique minimum spanning tree.

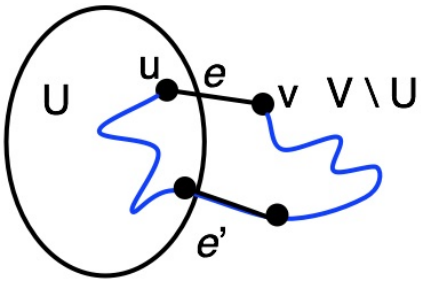
3 Light-Edge Property

There are several algorithms for computing minimum spanning trees. These algorithms are all based on the same underlying property about cuts in a graph, which we will refer to as the ***light-edge property***. Intuitively, the light-edge property states that if we partition the graph into two blocks, the minimum edge between the two blocks is in the MST. The light-edge property gives a way to identify algorithmically the edges of the MST.

Definition 63.3 (Graph Cut)**.** For a graph $G = (V, E)$, a cut is defined in terms of a non-empty proper subset $U \subsetneq V$. The set U partitions the graph into blocks induced by the vertex set U and the vertex set $V \setminus U$, which together are called the ***cut*** and written as the cut $(U, V \setminus U)$. We refer to the edges between the two parts as the ***cut edges*** written $E(U, V \setminus U)$. We sometimes say that a cut edge ***crosses*** the cut.

Example 63.4. If the subset U in the definition of graph-cuts is a single vertex v of the graph. The cut edges consist of all edges incident on v.

Lemma 63.3 (Light-Edge Property)**.** Let $G = (V, E, w)$ be a connected undirected, weighted graph with distinct edge weights. For any cut of G, the minimum weight edge that crosses the cut is contained in the minimum spanning tree of G.

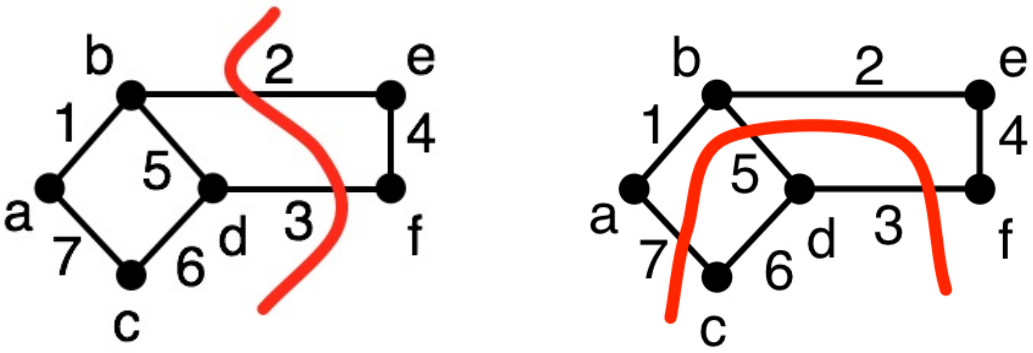


Proof.

The proof is by contradiction. Assume the lightest edge $e = \{u, v\}$ is not in an MST. Since the MST spans the graph, there is a simple path P connecting u and v in the MST (i.e., consisting of only edges in the MST). Path P crosses the cut between U and $V \setminus U$ at least once since u and v are on opposite sides of the cut.

Let e' be an edge on the path P that crosses the cut. Because e is the lightest edge crossing the cut, e' is heavier than e. But by Edge-Replacement Lemma for spanning trees, we know that we can insert e into the MST and delete e' and obtain a lighter spanning tree. This is a contradiction, and thus the lightest edge crossing the cut is in the MST. □

Example 63.5 (Cuts and Light Edges)**.** The figures below illustrates two cuts. For each cut, we can find the light edge that crosses that cut, which are the edges with weight 2 (top) and 3 (bottom) respectively.



Light-Edge Property and Algorithms. An important implication of the light-edge property as proved in Lemma 63.3 is that any lightest edge that crosses a cut can be immediately added to the MST. In fact, the algorithms that we consider in this section all take advantage of this implication. For example, Kruskal's algorithm constructs the MST by greedily adding the overall minimum edge. Prim's algorithm grows an MST incrementally by considering a cut between the current MST and the rest of graph. Boruvka's algorithm constructs a tree in parallel by considering the cut defined by each and every vertex.

Exercise 63.4. Consider any undirected, connected graph G with unique edge weights. Show that for any cycle in the graph, the heaviest edge on the cycle is not in the MST of G.

4 Approximating Metric TSP via MST

TSP and MST. There is an interesting connection between minimum spanning trees and the symmetric Traveling Salesperson Problem (TSP), an NP-hard problem: certain instances of TSP can be approximated successfully by using MST's. In this section, we present such an approximation algorithm.

Lower Bounding TSP with MST. Recall that in TSP problem, we are given a set of n cities (vertices) and are interested in finding a tour that visits all the vertices exactly once and returns to the origin. In the symmetric case of the problem, the edges are undirected (or equivalently the distance is the same in each direction). For the TSP problem, we usually consider complete graphs, where there is an edge between any two vertices. Even if a graph is not complete, we can typically complete it by inserting edges with large weights that make sure that the edge never appears in a solution. Here we also assume the edge weights are non-negative.

Since the solution to the TSP problem visits every vertex once (returning to the origin), it spans the graph. But the solution is not a tree but a cycle in which each vertex is visited once. Dropping any edge from the solution therefore would yield a spanning tree. Therefore, a solution to the TSP problem cannot have less total weight than that of a minimum spanning tree. We can thus conclude that for undirected graphs with non-negative edge

weights, a minimum spanning tree can be used to obtain a lower bound for the (symmetric) TSP problem.

Approximating TSP with MST. As we shall see in the rest of this section, minimum spanning trees can also be used to find an approximate solutions to the TSP problem, effectively finding an upper bound. This, however, requires one more condition on the TSP problem. In particular in addition to requiring that weights are non-negative we require that all distances satisfy the triangle inequality—i.e., for any three vertices a, b, and c, $w(a, c) \leq w(a, b) + w(b, c)$.

Definition 63.4 (Metric Traveling Salesperson (TSP) Problem)**.** Given a complete undirected graph $G = (V, E)$ with edge weights $W : E \to \mathbb{R}$ such that

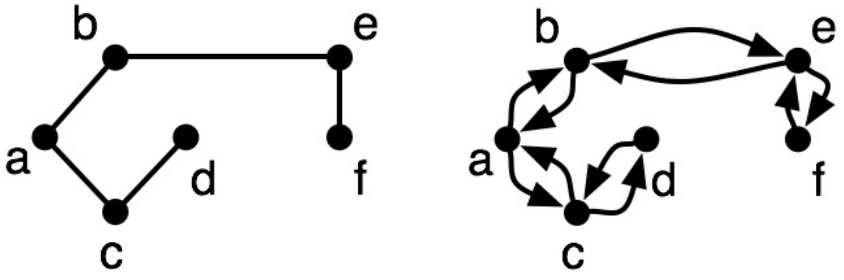
- for all $e \in E$, $W(e) \geq 0$, and
- for all $u, v, w \in E$, $W(u, v) + W(v, w) \geq W(u, w)$,

find the minimum-weight cycle that visits all the vertices.

From a TSP to an Euler Tour. We would like a way to use the MST to generate a path to take as an approximate solution to the metric TSP problem. To do this we first consider a path based on the MST that can visit a vertex multiple times, and then take shortcuts to ensure we only visit each vertex once.

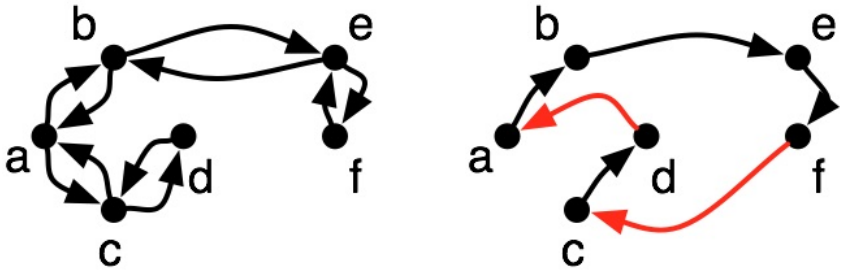
Given a minimum spanning tree T we can start at any vertex s and take a path based on the depth-first search on the tree from s. In particular whenever we visit a new vertex v from vertex u we traverse the edge from u to v and when we are done visiting everything reachable from v we then back up on this same edge, traversing it from v to u. This way every edge in our path is traversed exactly twice, and we end the path at our initial vertex. If we view each undirected edge as two directed edges, then this path is a so-called ***Euler tour*** of the tree—i.e. a cycle in a graph that visits every edge exactly once. Since T spans the graph, the Euler tour will visit every vertex at least once, but possibly multiple times.

Example 63.6 (Euler Tour)**.** The figure on the right shows an Euler tour of the tree on the left. Starting at `a`, the tour visits `a, b, e, f, e, b, a, c, d, c, a`.



Shortcuts. Recall that in the TSP problem, the underlying graph is complete and thus there is an edge between every pair of vertices. Because it is possible to take an edge from any vertex to any other, we can take shortcuts to avoid visiting vertices multiple times. More precisely what we can do is the following: when we are about to go back to a vertex that the tour has already visited, instead find the next vertex in the tour that has not been visited and go directly to it. We call this a shortcut edge.

Example 63.7 (Shortcuts)**.** The figure on the right shows a solution to TSP with shortcuts, drawn in red. Starting at `a`, we can visit `a, b, e, f, c, d, a`.



Final Bounds. We are now ready to give an upper bound on the TSP problem in terms of MST. Note that by the triangle inequality the shortcut edges are no longer than the paths that they replace. Thus by taking shortcuts, the total distance is not increased. Since the Euler tour traverses each edge in the minimum spanning tree twice (once in each direction), the total weight of the path is exactly twice the weight of the MST. With shortcuts, we obtain a solution to the TSP problem that is at most the weight of the Euler tour, and hence at most twice the weight of the MST. Because the weight of the MST is also a lower bound on the TSP, the solution we have found is within a factor of 2 of optimal. This means our approach is an approximation algorithm for TSP that approximates the solution within a factor of 2. This can be summarized as:

$$W(\text{MST}(G)) \leq W(\text{TSP}(G)) \leq 2W(\text{MST}(G)) \ .$$

Remark. It is possible to reduce the approximation factor to 1.5 using a well known algorithm developed by Nicos Christofides at CMU in 1976. The algorithm is also based on the MST problem, but is followed by finding a vertex matching on the vertices in the MST with odd-degree, adding these to the tree, finding an Euler tour of the combined graph, and again shortcutting. Christofides algorithm was one of the first approximation algorithms and it took over 40 years to improve on the result, and only very slightly.