1

Introduction

In this introductory chapter, we will formally introduce the main variants of the traveling salesman problem – symmetric and asymmetric – explain a very useful graph-theoretic view based on Euler's theorem, and describe the classical simple approximation algorithms. In this chapter, we will also introduce basic notation, in particular from graph theory, and some fundamental combinatorial optimization problems.

1.1 Problems and Algorithms

This book is about the traveling salesman problem (TSP), but this is actually more than one problem. Although one could start with the most general variant of the problem, let us begin with the most classical one.

Problem 1.1 (Symmetric TSP with Triangle Inequality)**.**

Instance: A finite set V and a distance function $c : V \times V \to \mathbb{R}_{\geq 0}$ such that $c(u, v) = c(v, u)$ for all $u, v \in V$ and $c(u, w) \leq c(u, v) + c(v, w)$ for all $u, v, w \in V$.

Task: Compute a list $v_1, v_2, \ldots, v_n$ that contains every element of V exactly once and minimizes $c(v_n, v_1) + \sum_{i=2}^{n} c(v_{i-1}, v_i)$.

The elements of V are called *cities*, and the number of cities will always be denoted by n in this book. Of course, the distance function c does not necessarily describe geometric distances, but it could also represent driving times or cost. An example with $|V| = 20$ is shown in Figure 1.1.

We will be interested in *algorithms* that accept any instance (V, c) as input and always terminate with a *feasible solution* (an order of the cities) as output. If an algorithm always finds an optimum solution, we speak of an *exact*

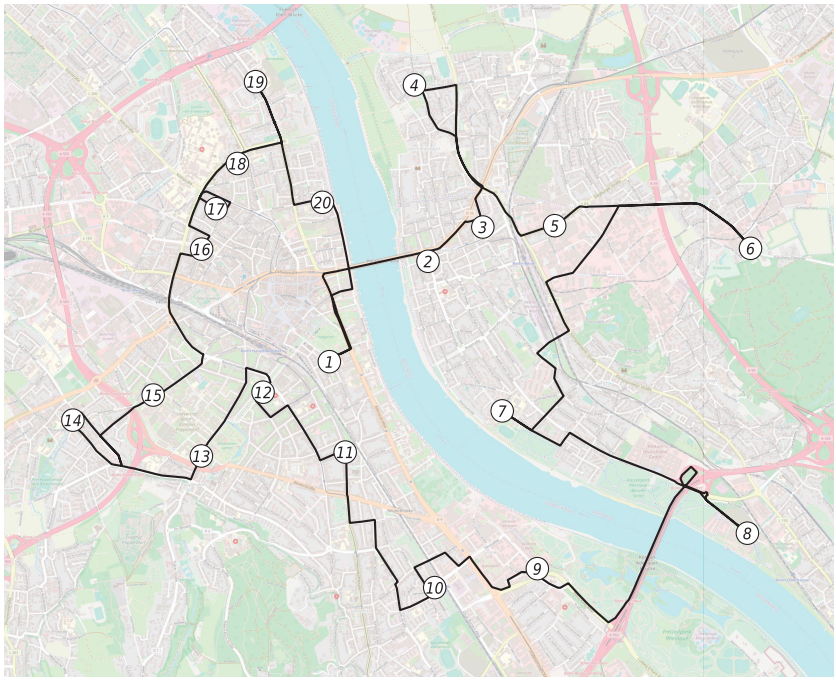

Figure 1.1 A tour visiting 20 locations in Bonn, Germany, by car. The map data is taken from OpenStreetMap (openstreetmap.org/copyright).

algorithm. One such algorithm would simply enumerate all $n!$ permutations of the cities and output the best, but this is too slow already for 20 cities (note that $20! = 2\,432\,902\,008\,176\,640\,000$), and completely hopeless for the 49-city instance that Dantzig, Fulkerson, and Johnson [1954] solved 70 years ago.

To measure the running time of an algorithm, one counts the maximum number of elementary steps that it can take. To avoid technical details, one ignores constant factors and uses the O-notation. For example, an algorithm is said to run in $O(n^3)$ time if there is a constant γ such that the number of elementary steps is never more than $\gamma \cdot n^3$. See, for example, Hougardy and Vygen [2016] for a detailed explanation.

To distinguish algorithms that are, at least asymptotically, much faster than naive enumeration, Edmonds [1965a] suggested the notion of a *polynomial-time algorithm*. For every algorithm that we present in this book, there is a constant k such that the algorithm runs in $O(n^k)$ time.

Karp [1972] showed that the Symmetric TSP with Triangle Inequality is *NP*-hard. This implies that there is no polynomial-time exact algorithm unless

$P = NP$. In fact, there is a polynomial-time exact algorithm if and only if $P = NP$. We assume that the reader is familiar with the notions of P, NP, and NP-hard; otherwise, it is sufficient to know that it is widely believed that $P \neq NP$, which would imply that there is no polynomial-time exact algorithm for any NP-hard problem.

The fastest known exact algorithm is a simple dynamic programming algorithm that computes an optimum solution in $O(n^2 2^n)$ time (Bellman [1962], Held and Karp [1962]; see Exercise 1.1). This time bound has not been improved on for more than 60 years. Since most researchers believe that $P \neq NP$, there is little hope to find a polynomial-time algorithm that always finds an optimum solution. Hence we will study approximation algorithms:

Definition 1.2 (approximation algorithm)**.** An *α-approximation algorithm* (for a minimization problem with nonnegative cost function) is a polynomial-time algorithm that always computes a feasible solution of cost at most α times the optimum.

In the context of the TSP, α can be either a constant or a function of n (the number of cities). For a constant-factor approximation algorithm, we define its *approximation ratio* to be the infimum of all α for which it is an α-approximation algorithm, or, equivalently, the supremum of $\frac{A(I)}{\mathrm{OPT}(I)}$ over all instances I, where $A(I)$ is the cost of the solution computed by the algorithm, $\mathrm{OPT}(I)$ is the cost of an optimum solution, and $\frac{0}{0} := 1$.

Probably the first proof of an approximation ratio for the TSP was due to Rosenkrantz, Stearns, and Lewis [1977]. They proposed algorithms called "nearest insertion" and "cheapest insertion" and showed that they are 2-approximation algorithms for the Symmetric TSP with Triangle Inequality. We will see a simpler 2-approximation algorithm in Proposition 1.22.

The *triangle inequality* $c(u, w) \leq c(u, v) + c(v, w)$ for all $u, v, w \in V$ naturally holds in many applications. If we have general nonnegative symmetric distances, not obeying the triangle inequality, we should allow for visiting cities more than once; we will get to this in the next section.

Distances are not always symmetric. Dropping the symmetry assumption yields the following:

Problem 1.3 (Asymmetric TSP with Triangle Inequality)**.**

Instance: A finite set V and a distance function $c : V \times V \to \mathbb{R}_{\geq 0}$ such that $c(u, w) \leq c(u, v) + c(v, w)$ for all $u, v, w \in V$.

Task: Compute a list $v_1, v_2, \ldots, v_n$ that contains every element of V exactly once and minimizes $c(v_n, v_1) + \sum_{i=2}^{n} c(v_{i-1}, v_i)$.

This problem seems to be substantially harder; while it is easy to devise a 2-approximation algorithm for the symmetric special case (see Proposition 1.22), no such algorithm is known for the asymmetric version, and no constant-factor approximation algorithm was known at all until 2017.

1.2 Graphs and Euler's Theorem

Often distances are given by a graph $G = (V, E)$ (which can, for example, represent a street network). All our *graphs* are finite; they can be undirected or directed. In both cases, they consist of a finite set V of vertices and a finite set E of edges such that each edge is associated with a pair of distinct vertices. All edge sets in this book can be multi-sets unless specified otherwise, so graphs can have parallel edges (but no loops). Graphs without parallel edges are called *simple*. Directed graphs are also called *digraphs*. For an edge $e \in E$ that goes from v to w, we write $e = \{v, w\}$ in undirected graphs and $e = (v, w)$ in digraphs, and we use this notation even if there are several edges going from v to w. Edges in digraphs are also called *arcs*. If G is a directed graph, the *underlying undirected graph* results from replacing each arc (v, w) with an undirected edge $\{v, w\}$. If H is the underlying undirected graph of a digraph G, then G is called an *orientation* of H. An edge $e = \{v, w\}$ or $e = (v, w)$ is *incident* to v and w, and if such an edge exists, v and w are *neighbors*. A vertex without neighbors is *isolated*. For a graph $G = (V, E)$, we sometimes write $V(G) := V$ and $E(G) := E$.

For a vertex set $U \subseteq V$, we denote by $\delta(U)$ the (multi)set of edges with exactly one endpoint in U. In directed graphs, $\delta^-(U)$ and $\delta^+(U)$ contain the entering and the leaving edges, respectively (so $|\delta(U)| = |\delta^-(U)| + |\delta^+(U)|$). For a single vertex $v \in V$, we write $\delta(v) := \delta(\{v\})$, $\delta^-(v) := \delta^-(\{v\})$, and $\delta^+(v) := \delta^+(\{v\})$. We call $|\delta(v)|$ (the number of edges incident to v) the *degree* of v, and in digraphs, $|\delta^-(v)|$ and $|\delta^+(v)|$ are the *in-degree* and *out-degree* of v, respectively. We add a subscript and, for example, write $\delta_G(U)$ or $\delta_E(U)$ if the graph $G = (V, E)$ is not clear from the context.

Lemma 1.4 (handshake lemma)**.** *In any graph, the number of odd-degree vertices is even.*

Proof. For any graph (V, E), we have $\sum_{v \in V} |\delta(v)| = 2|E|$; hence there is an even number of odd summands on the left-hand side. □

A *walk* (from v_0 to v_k of *length* k) in G is a sequence $v_0, e_1, v_1, e_2, \ldots, v_k$ such that e_i is an edge from vertex v_{i-1} to vertex v_i for all $i = 1, \ldots, k$. If

$v_0 = v_k$, we speak of a *closed walk*. Note that $k = 0$ is possible. The *footprint* of a walk in $G = (V, E)$ is the multi-subset of E that contains r copies of any edge that the walk traverses r times. For a multi-subset F of E and a cost function $c : E \to \mathbb{R}$, we define the *cost* of F by $c(F) := \sum_{e \in F} c(e)$, where the sum counts edges according to their multiplicity in F. Then the cost of a walk $v_0, e_1, v_1, e_2, \ldots, v_k$ with footprint F can be expressed as $c(F) = \sum_{e \in F} c(e) = \sum_{i=1}^{k} c(e_i)$. Sometimes we say *weight* instead of cost.

A walk cannot visit all vertices unless the graph is connected. An undirected graph is *connected* if it contains a walk from v to w for all $v, w \in V$. A directed graph is *connected* if the underlying undirected graph is connected. A directed graph is *strongly connected* if it contains a walk from v to w for all $v, w \in V$.

A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$. We also say that G *contains* G' or that G' is *in* G. For a graph $G = (V, E)$ and $\emptyset \neq W \subseteq V$, the graph with vertex set W that contains all edges of E with both endpoints in W is called the subgraph of G *induced by* W and is denoted by $G[W]$; its edge set is denoted by $E[W]$. The maximal connected subgraphs of a graph G are its *connected components*; they are induced subgraphs. A *multi-subgraph* results from a subgraph by possibly adding copies of edges. Sometimes we obtain subgraphs by deleting an edge e, a vertex v (and its incident edges), or a set of vertices X and write $G - e$, $G - v$, and $G - X = G[V(G) \setminus X]$.

Contracting a vertex set W in a graph G means deleting all vertices and edges in $G[W]$, adding a new vertex v_W, and for every edge in $\delta(W)$ replacing the endpoint in W by v_W. We call the result G/W. Contracting an edge means contracting the (two-element) set of its endpoints.

Given a graph G (directed or undirected), we will be looking for a closed walk in G that contains every vertex at least once. Euler [1736] observed that the footprint of such a walk has a simple property:

Definition 1.5 (Eulerian)**.** An undirected graph $G = (V, E)$ (and its edge set E) is called *Eulerian* if every vertex has even degree. A directed graph $G = (V, E)$ (and its edge set E) is called *Eulerian* if for every vertex the in-degree equals the out-degree.

The following characterization is known as Euler's theorem:

Theorem 1.6 (Euler [1736], Hierholzer [1873])**.** *Let $G = (V, E)$ be a connected graph (directed or undirected). Then G is Eulerian if and only if it contains a closed walk that traverses each edge exactly once. Such a walk can be computed in $O(|E|)$ time.*

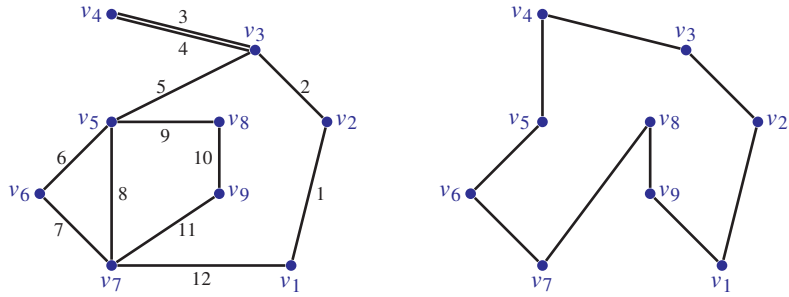

Figure 1.2 The left picture shows a connected Eulerian graph. By Theorem 1.6, this graph contains a closed walk that traverses every edge exactly once. The numbers next to the edges show the order in which the edges appear in one such Eulerian walk. The vertices $v_1, \ldots, v_n$ are numbered in the order of their first appearance in this walk. The right picture shows the solution to the Symmetric TSP with Triangle Inequality that results from taking shortcuts as in the proof of Lemma 1.7.

Proof. Let $G = (V, E)$. If G contains a closed walk $v_0, e_1, v_1, \ldots, e_k, v_k$ that traverses each edge exactly once, then E is the footprint of this walk. If this walk visits a vertex r times (where v_0 and v_k count as one visit), then it enters that vertex r times and leaves it r times; so its degree is $2r$. Hence G is Eulerian.

We prove the converse by induction on $|E|$, the case $E = \emptyset$ being trivial. So let $G = (V, E)$ be Eulerian, and let F be the footprint of a walk $v_0, e_1, v_1 \ldots, e_k, v_k$ in G that contains every edge at most once and is as long as possible. Since E is nonempty, so is F. Moreover, $v_k = v_0$, for otherwise there is an unused edge leaving v_k that we could append to the walk. So we have a closed walk, and by the first part, its footprint F is Eulerian. Hence also $E \setminus F$ is Eulerian. By induction, $E \setminus F$ is the union of footprints of closed walks (one in each connected component), and each of them must contain a vertex of $v_1, \ldots, v_k$ because G is connected. So we can insert the other walks at these positions into the first walk.

This proof easily implies a linear-time algorithm, by greedily extending a walk as long as possible and recursively applying the algorithm to the remainder. □

We use Euler's theorem as follows (cf. Figure 1.2):

Lemma 1.7. *Let (V, c) be an instance of Symmetric TSP with Triangle Inequality, and let (V, F) be a connected Eulerian undirected graph. Then there exists a solution $v_1, \ldots, v_n$ of cost at most $\sum_{e=\{v,w\} \in F} c(v, w)$, and such a solution can be found in $O(|F|)$ time.*

Proof. By Theorem 1.6, one can construct, in $O(|F|)$ time, a closed walk in (V, F) that traverses each edge exactly once. Let $v_1, \ldots, v_n$ be the elements of V in the order in which they appear in that walk for the first time. Then $c(v_n, v_1) + \sum_{i=2}^{n} c(v_{i-1}, v_i) \leq \sum_{e=\{v,w\} \in F} c(v, w)$ because, by the triangle inequality, $c(v_{i-1}, v_i)$ is at most the total cost of the edges in the subwalk from the first appearance of v_{i-1} to the first appearance of v_i (for $i = 1, \ldots, n$, where $v_0 := v_n$). □

The same proof works for the directed version:

Lemma 1.8. *Let (V, c) be an instance of Asymmetric TSP with Triangle Inequality, and let (V, F) be a connected Eulerian directed graph. Then there exists a solution $v_1, \ldots, v_n$ of cost at most $\sum_{e=(v,w) \in F} c(v, w)$, and such a solution can be found in $O(|F|)$ time.* □

This motivates the following definition, which plays a central role in this book:

Definition 1.9 (tour)**.** A *tour* in a graph $G = (V, E)$ (directed or undirected) is a multi-subset F of E such that (V, F) is connected and Eulerian.

By Theorem 1.6, an edge set F is a tour if and only if it is the footprint of a closed walk in G that visits every vertex at least once. Using this, we can formulate the TSP in graph-theoretical terms:

Problem 1.10 (Symmetric TSP)**.**

Instance: A simple connected undirected graph $G = (V, E)$ and a cost function $c : E \to \mathbb{R}_{\geq 0}$.

Task: Compute a tour in G with minimum cost.

This has sometimes been called the graphical TSP (see, e.g., Cornuéjols, Fonlupt, and Naddef [1985]). In the asymmetric setting, we can use the same terminology:

Problem 1.11 (Asymmetric TSP)**.**

Instance: A simple strongly connected directed graph $G = (V, E)$ and a cost function $c : E \to \mathbb{R}_{\geq 0}$.

Task: Compute a tour in G with minimum cost.

Now we want to argue that these graph-theoretic versions of the TSP are equivalent to the ones given in the previous section. We say that problem P_1 *reduces to* P_2 if there is a polynomial-time algorithm that computes, for any given instance I_1 of P_1, an instance I_2 of P_2 with the same optimum cost, and

for any feasible solution S_2 of I_2, a solution S_1 of I_1 with no larger cost. If P_1 reduces to P_2 and P_2 reduces to P_1, we say that P_1 and P_2 are *equivalent*.

If $v_0, e_1, v_1, e_2, \ldots, v_k$ is a walk without any repetitions of vertices, then the graph with vertex set $\{v_0, \ldots, v_k\}$ and edge set $\{e_1, \ldots, e_k\}$ is called a *path*. If $v_0 = v_k$ but there are no other repetitions, this graph is called a *circuit* (or *cycle*). (A path or circuit can be directed or undirected.) A path or circuit in a graph G is called *Hamiltonian* if it contains all vertices of G. A solution to an instance of the Symmetric TSP with Triangle Inequality with $n \geq 3$ can be interpreted as a Hamiltonian circuit in the *complete graph* on vertex set V (whose edge set is $\binom{V}{2}$). We write $c(e) := c(v, w)$ for an edge $e = \{v, w\}$ of this graph. Similarly, a solution to an instance of the Asymmetric TSP with Triangle Inequality with $n \geq 2$ can be interpreted as a Hamiltonian circuit in the *complete directed graph* on vertex set V (whose edge set is $\{(v, w) \in V \times V : v \neq w\}$). Again we write $c(e) := c(v, w)$ for an edge $e = (v, w)$ of this digraph.

For an instance (G, c) of the Symmetric TSP or the Asymmetric TSP and two vertices v and w of G, the *distance* from v to w is the minimum total cost of the edges of a walk from v to w. We often denote it by $\mathrm{dist}_{(G,c)}(v, w)$. By *Dijkstra's algorithm*, such a walk can be computed in polynomial time (see Theorem 1.14).

Proposition 1.12. *Symmetric TSP with Triangle Inequality (Problem 1.1) and Symmetric TSP (Problem 1.10) are equivalent. Asymmetric TSP with Triangle Inequality (Problem 1.3) and Asymmetric TSP (Problem 1.11) are equivalent.*

Proof. We first reduce Problem 1.10 to Problem 1.1. Let $I_1 = (G, c)$ be an instance of Problem 1.10 with $G = (V, E)$. For $v, w \in V$, let $\bar{c}(v, w)$ be the distance from v to w in (G, c), and $I_2 := (V, \bar{c})$. By Lemma 1.7, from any tour F in G, we can construct a solution $v_1, \ldots, v_n$ to I_2 with $\bar{c}(v_n, v_1) + \sum_{i=2}^{n} \bar{c}(v_{i-1}, v_i) \leq \bar{c}(F) \leq c(F)$.

Conversely, for every solution $v_1, \ldots, v_n$ to I_2, we can compute a minimum-cost walk in (G, c) from v_n to v_1 and from v_{i-1} to v_i for $i = 2, \ldots, n$ and append all these walks to obtain a closed walk visiting all vertices. Its footprint is a tour with cost $\bar{c}(v_n, v_1) + \sum_{i=2}^{n} \bar{c}(v_{i-1}, v_i)$. In particular, I_1 and I_2 have the same optimum cost.

The reduction from Problem 1.11 to Problem 1.3 is identical.

To reduce Problem 1.1 to Problem 1.10, let $I_1 = (V, c)$ be an instance of Problem 1.1 and define $I_2 = (G, c)$, where $G = (V, E)$, $E = \binom{V}{2}$, and $c(e) = c(v, w)$ for all $e = \{v, w\} \in E$. Every solution to I_1 corresponds to a Hamiltonian circuit in G with the same cost and vice versa. For every tour F in G, we can construct a Hamiltonian circuit of at most the same cost by Lemma 1.7.

To reduce Problem 1.3 to Problem 1.11, let $I_1 = (V, c)$ be an instance of Problem 1.3 and define $I_2 = (G, c)$, where $G = (V, E)$ has edge set $E = \{(v, w) : v, w \in V,\ v \neq w\}$, and $c(e) = c(v, w)$ for all $e = (v, w) \in E$, and proceed the same way. □

If problem P_1 reduces to problem P_2 and P_2 has an α-approximation algorithm, then so has P_1. We will often work with the graph versions of the TSP (Symmetric TSP and Asymmetric TSP), but sometimes the versions with the triangle inequality are more useful. Proposition 1.12 allows us to switch between the versions and use whichever is more convenient.

If we do not require the triangle inequality but still want to visit every city exactly once, the problem is hopeless, as Sahni and Gonzalez [1976] observed: Any approximation algorithm would imply $P = NP$. This is because any α-approximation algorithm, for any function α, would allow us to decide in polynomial time whether a given graph contains a Hamiltonian circuit (see Problem 1.20): Just define the distance of two cities to be 0 if they are joined by an edge in the graph and 1 otherwise; then any α-approximation algorithm outputs a solution of cost 0 if and only if the given graph contains a Hamiltonian circuit.

1.3 Some Basic Combinatorial Optimization Problems

In this section, we cite three classical combinatorial optimization results without proofs. Proofs can be found in every book on combinatorial optimization, such as Schrijver [2003] or Korte and Vygen [2018].

We already used the fact that a walk from s to t whose footprint has minimum cost can be computed in polynomial time. We may assume that such a walk does not visit any vertex more than once, for otherwise we can omit cycles and obtain a walk with fewer edges that does not cost more. For an instance (G, c) of the Symmetric TSP or the Asymmetric TSP and a subgraph H of G with edge set F, we call $c(F)$ the *cost* of H. So we can formulate the problem of finding a walk from s to t whose footprint has minimum cost as follows:

Problem 1.13 (Shortest Path)**.**

Instance: A simple graph $G = (V, E)$ (directed or undirected), a cost function $c : E \to \mathbb{R}_{\geq 0}$, and two vertices $s, t \in V$.

Task: Compute a path P from s to t in G with minimum cost, or decide that there is no such path.

The classical algorithm of Dijkstra [1959] (see Exercise 1.4) solves this problem efficiently. As before, n denotes the number of vertices in the given graph.

Theorem 1.14 (Dijkstra [1959])**.** *There is an $O(n^2)$-time algorithm for* Shortest Path.

A faster running time can be obtained for sparse graphs (with $o(n^2)$ edges), but this is not important for the purpose of this book. Note that it is essential that the cost function is nonnegative: The shortest path problem with general weights is *NP*-hard. Using Dijkstra's algorithm, we can compute the distance from s to t for all vertices $s, t \in V$. Since the algorithm in fact computes shortest paths from one vertex s to all vertices $t \in V$, only n applications of Dijkstra's algorithm suffice. The *metric closure* of a pair (G, c), where $G = (V, E)$ is a directed or undirected graph and $c : E \to \mathbb{R}_{\geq 0}$, is the pair $(\bar{G}, \bar{c})$ where $\bar{G}$ has the same vertex set and contains an edge $e = (v, w)$ or $e = \{v, w\}$, respectively, whenever G contains a path from v to w, and $\bar{c}(e)$ is the distance from v to w.

Theorem 1.15. *There is an $O(n^3)$-algorithm that, given a simple graph $G = (V, E)$ and a cost function $c : E \to \mathbb{R}_{\geq 0}$, computes the metric closure of (G, c).*

Another basic problem asks to connect all vertices at minimum cost. A *tree* is a minimal connected graph – that is, the deletion of any edge would destroy connectivity. A subgraph of a graph G is called *spanning* if it contains all vertices of G.

Problem 1.16 (Minimum Spanning Tree)**.**

Instance: A simple undirected graph $G = (V, E)$ and a cost function $c : E \to \mathbb{R}_{\geq 0}$.

Task: Compute a spanning tree (V, S) in G with minimum cost, or decide that G is not connected.

This problem was solved quite early by Borůvka [1926]:

Theorem 1.17 (Borůvka [1926], Jarník [1930], Prim [1957])**.** *There is an $O(n^2)$-time algorithm for* Minimum Spanning Tree.

In fact, it is well known that the Minimum Spanning Tree problem can be solved by a simple greedy algorithm (see Exercise 1.5 or the proof of Theorem 2.14).

The third problem we cite here is much more difficult to solve. A *perfect matching* in G is a set M of edges such that every vertex of G is incident to exactly one of these edges.

Problem 1.18 (Weighted Matching)**.**

Instance: A simple undirected graph $G = (V, E)$ with $|V|$ even and a cost function $c : E \to \mathbb{R}_{\geq 0}$.

Task: Compute a perfect matching in G with minimum cost, or decide that no perfect matching exists.

This problem was solved by Edmonds [1965b]:

Theorem 1.19 (Edmonds [1965b], Gabow [1973])**.** *There is an $O(n^3)$-time algorithm for Weighted Matching.*

For Minimum Spanning Tree and Weighted Matching, one could also allow edges with negative cost. Since all spanning trees have $n - 1$ edges and all perfect matchings have $\frac{n}{2}$ edges, adding a constant to all edges costs does not change the set of optimum solutions. This trick does not work for Shortest Path. One can still solve Shortest Path in polynomial time for *conservative weights* (i.e., when there is no circuit of negative total weight), but this is more complicated (see Exercise 1.11).

In contrast to the above three problems, many others have a polynomial-time algorithm only if $P = NP$. We mention one famous example of such an NP-hard problem:

Problem 1.20 (Hamiltonian Circuit)**.**

Instance: A undirected graph $G = (V, E)$.

Task: Decide whether G has a Hamiltonian circuit.

Theorem 1.21 (Karp [1972])**.** *Hamiltonian Circuit has a polynomial-time algorithm if and only if $P = NP$.*

A graph with a Hamiltonian circuit is called *Hamiltonian*. Theorem 1.21 easily implies that the shortest path problem with general weights has no polynomial-time algorithm unless $P = NP$ (cf. Exercise 1.7).

1.4 Christofides' Algorithm

Given an instance I of one of our TSP variants, we will denote by $\mathrm{OPT}(I)$ the cost of an optimum solution. For two edge sets A and B, we denote by $A \dot{\cup} B$ (the *disjoint union* of A and B) the multi-set that contains two copies of each edge in $A \cap B$ and one copy of each edge in $(A \cup B) \setminus (A \cap B)$. Our very first approximation algorithm is now almost trivial:

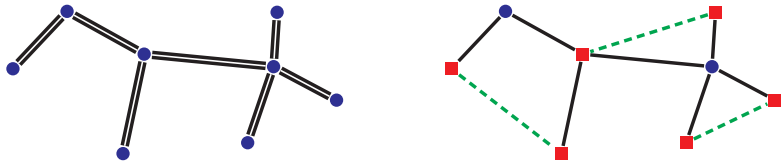

Figure 1.3 Illustrating the double tree algorithm (left) and Christofides' algorithm (right). Both start with a minimum-cost spanning tree. The former then doubles all edges, while the latter adds a minimum-cost perfect matching (green, dashed) among the odd-degree vertices (red squares).

Proposition 1.22 (Rosenkrantz, Stearns, and Lewis [1977])**.** *There is a 2-approximation algorithm for* Symmetric TSP.

Proof. Given an instance (G, c) with $G = (V, E)$, compute a minimum-cost spanning tree (V, S) in (G, c) (cf. Theorem 1.17) and output the tour $S \dot\cup S$, which results from S by doubling all edges. Since any tour is connected and thus contains a spanning tree, we have $c(S) \le \mathrm{OPT}(G, c)$, so the tour $S \dot\cup S$ that we compute costs at most $2\,\mathrm{OPT}(G, c)$. □

Rosenkrantz, Stearns, and Lewis [1977] actually proved (already in 1974) that two different algorithms ("nearest insertion" and "cheapest insertion") are 2-approximation algorithms, but the above folklore proof is much simpler. The algorithm in the proof of Proposition 1.22 has often been called the *double tree algorithm*.

Christofides [1976], and independently Serdyukov [1978], showed how to improve on this. (See van Bevern and Slugina [2020] for a historical note.) Christofides' algorithm also begins by computing a minimum-cost spanning tree (V, S). Then, instead of doubling all edges, it finds a potentially cheaper way to make S Eulerian. For a graph (V, F), let $\mathrm{odd}(F) = \{v \in V : |F \cap \delta(v)| \text{ odd}\}$ denote the set of odd-degree vertices. We formulate Christofides' algorithm first for the Symmetric TSP with Triangle Inequality. See Algorithm 1.23, and see Figure 1.3 for an illustration.

Theorem 1.24 (Christofides [1976], Serdyukov [1978])**.** *Christofides' algorithm (Algorithm 1.23) is a* $\frac{3}{2}$*-approximation algorithm for* Symmetric TSP with Triangle Inequality.

Proof. An optimum solution corresponds to a Hamiltonian circuit H in G of cost $\mathrm{OPT}(V, c)$. We have $c(S) \le c(H)$ because deleting one edge from H results in a spanning tree. Let $w_1, \dots, w_k$ be the vertices of W in the order in which they appear in a traversal of H, and let $w_0 := w_k$. Note that

Algorithm 1.23: Christofides' Algorithm

Input: an instance (V, c) of Symmetric TSP with Triangle Inequality

Output: a solution $v_1, \ldots, v_n$

(1) Let $G = \left(V, \binom{V}{2}\right)$ be the complete graph on V, and for $e = \{v, w\} \in \binom{V}{2}$, let $c(e) = c(v, w)$.
(2) Compute a minimum-cost spanning tree (V, S) in (G, c).
(3) Let $W = \text{odd}(S)$.
(4) Compute a minimum-cost perfect matching M in $(G[W], c)$.
(5) Apply Lemma 1.7 to the tour $(V, S \dot\cup M)$ and output the resulting solution.

$k = |W|$ is even by the handshake lemma (Lemma 1.4). Then by the triangle inequality, $\sum_{i=1}^{k} c(w_{i-1}, w_i) \leq c(H)$. Hence we have two perfect matchings $M_1 = \{\{w_{i-1}, w_i\} : i \text{ even}\}$ and $M_2 = \{\{w_{i-1}, w_i\} : i \text{ odd}\}$ in $G[W]$ with total cost at most $c(H)$. So

$$c(M) \;\leq\; \min\left\{c(M_1), c(M_2)\right\} \;\leq\; \frac{1}{2}\left(c(M_1) + c(M_2)\right) \;\leq\; \frac{1}{2}c(H).$$

By Lemma 1.7, our output has cost $c(S \dot\cup M) = c(S) + c(M) \leq \frac{3}{2}c(H) = \frac{3}{2}\,\text{OPT}(V, c)$. The algorithm can be implemented to run in polynomial time by Theorems 1.17 and 1.19. □

We will now reformulate Christofides' algorithm for the Symmetric TSP. The following notion will be used very often in this book:

Definition 1.25 (T-join)**.** Let V be a finite set and $T \subseteq V$ with $|T|$ even. A *T-join* is a multi-subset J of $\binom{V}{2}$ with $T = \text{odd}(J)$. If $G = (V, E)$ is a graph and $J \subseteq E$, then we say that J is a T-join in G.

We start with a few basic properties. For two sets A and B, their *symmetric difference* $A \triangle B = (A \setminus B) \cup (B \setminus A)$ contains all elements that are in the union of A and B, but not in their intersection.

Proposition 1.26. *Let $G = (V, E)$ be an undirected graph and $T, T' \subseteq V$ with $|T|, |T'|$ even. Let J be a T-join and J' a T'-join. Then $J \triangle J'$ is a $(T \triangle T')$-join.*

Proof. A vertex v has odd degree in $(V, J \triangle J')$ if and only if it has odd degree in $(V, J \dot\cup J')$, and this is the case if and only if it has odd degree in exactly one of (V, J) and (V, J'). □

Proposition 1.27. *Let $G = (V, E)$ be an undirected graph and $T \subseteq V$ with $|T|$ even. Then G contains a T-join if and only if every connected component of G contains an even number of elements of T.*

Proof. Necessity follows from Lemma 1.4. For sufficiency, let $T = \{t_1, \dots, t_k\}$ such that t_{i-1} and t_i are in the same connected component of G for $i = 2, 4, \dots, k$. Then take any path P_i from t_{i-1} to t_i for $i = 2, 4, \dots, k$. The symmetric difference of the edge sets of these paths is a T-join by Proposition 1.26. □

Lemma 1.28. *Let $G = (V, E)$ be an undirected graph and $T \subseteq V$ with $|T|$ even. Let J be a T-join in G. Then there exists a numbering $T = \{t_1, \dots, t_k\}$ and a path from t_{i-1} to t_i in (V, J) for $i = 2, 4, \dots, k$ such that these paths are pairwise edge-disjoint.*

Proof. We use induction on $k = |T|$, the case $k = 0$ being trivial. By Proposition 1.27, there are two vertices $t_{k-1}, t_k \in T$ in the same connected component of (V, J), so let P be the edge set of a path from t_{k-1} to t_k in (V, J), and apply the induction hypothesis to $T' = T \setminus \{t_{k-1}, t_k\}$ and the T'-join $J \setminus P$. □

Using an algorithm for Weighted Matching, we can compute minimum-cost T-joins in polynomial time:

Theorem 1.29 (Edmonds and Johnson [1973])**.** *Given a simple undirected graph $G = (V, E)$, $c : E \to \mathbb{R}_{\geq 0}$, and $T \subseteq V$ with $|T|$ even, one can compute a minimum-cost T-join in (G, c) or decide that none exists in $O(n^3)$ time.*

Proof. The existence of a T-join can be decided with Proposition 1.27.

Let $H = \left(T, \binom{T}{2}\right)$ be the complete undirected graph on T, and for $v, w \in T$, let $\bar{c}(\{v, w\})$ be the distance from v to w in (G, c). Note that $\bar{c}$ can be computed in $O(n^3)$ time by Theorem 1.15.

Now compute a minimum-cost perfect matching M in $(H, \bar{c})$, using Theorem 1.19. For $\{v, w\} \in M$, compute a shortest path from v to w in (G, c), and let J be the symmetric difference of these $|M|$ paths. We prove that J is a minimum-cost T-join in (G, c).

Proposition 1.26 implies that J is indeed a T-join. To show that J has minimum cost, let J^* be a minimum-cost T-join. By Lemma 1.28, there exists a numbering $T = \{t_1, \dots, t_k\}$ and a path P_i from t_{i-1} to t_i in (V, J^*) for $i = 2, 4, \dots, k$ such that these paths are pairwise edge-disjoint. We conclude

$$c(J) \leq \bar{c}(M) \leq \sum_{i=2,4,\dots,k} \bar{c}(\{t_{i-1}, t_i\}) \leq \sum_{i=2,4,\dots,k} c(P_i) \leq c(J^*). \quad \square$$

The problem can actually be solved for general weights too. We do not need this for now but note it for later use (in Chapter 12):

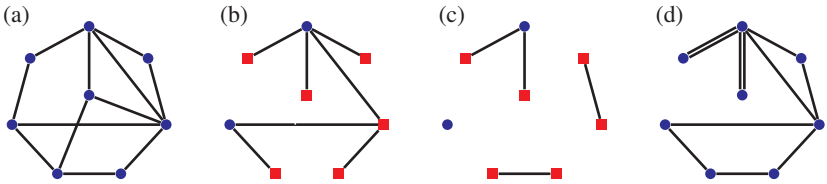

Figure 1.4 Christofides' algorithm illustrated for an unweighted graph instance. (a) An unweighted graph G (i.e., $c(e) = 1$ for all edges e), (b) a spanning tree (V, S) whose odd-degree vertices (elements of $\mathrm{odd}(S)$) are shown as red squares, (c) a minimum $\mathrm{odd}(S)$-join J, and (d) the resulting tour $S \dot\cup J$.

Corollary 1.30. *Given a simple undirected graph $G = (V, E)$, $c : E \to \mathbb{R}$, and $T \subseteq V$ with $|T|$ even, one can compute a minimum-cost T-join in (G, c) or decide that none exists in $O(n^3)$ time.*

Proof. Let $E^- = \{e \in E : c(e) < 0\}$ and $c'(e) := |c(e)|$ for all $e \in E$. Then $c'(K \triangle E^-) = c(K) - c(E^-)$ for all $K \subseteq E$. Let $T' := T \triangle \mathrm{odd}(E^-)$. Then J' is a minimum c'-cost T'-join if and only if $J' \triangle E^-$ is a minimum c-cost T-join. Hence the problem reduces to Theorem 1.29. □

With the notion of T-joins, we have an elegant reformulation of Christofides' algorithm: see Algorithm 1.31. Figure 1.4 provides an example.

Algorithm 1.31: Christofides' Algorithm

Input: an instance (G, c) of Symmetric TSP
Output: a tour F

(1) Compute a minimum-cost spanning tree (V, S) in (G, c).
(2) Let $W = \mathrm{odd}(S)$, and let J be a minimum-cost W-join in (G, c).
(3) Output the tour $S \dot\cup J$.

The running time of Christofides' algorithm is also $O(n^3)$, dominated by the subroutine to find a minimum-cost $\mathrm{odd}(S)$-join (cf. Theorem 1.29). Let us now prove the approximation guarantee again for this version:

Theorem 1.32 (Christofides [1976], Serdyukov [1978])**.** *Christofides' algorithm (Algorithm 1.31) is a $\frac{3}{2}$-approximation algorithm for Symmetric TSP.*

Proof. We have $c(S) \le \mathrm{OPT}(G, c)$, like in the proof of Proposition 1.22. Any optimum tour F^* contains a W-join J_1 by Proposition 1.27. Let $J_2 := F^* \setminus J_1$. By Proposition 1.26, $\mathrm{odd}(J_2) = \mathrm{odd}(F^*) \triangle \mathrm{odd}(J_1) = \emptyset \triangle W = W$. After deleting

Table 1.1 *Approximation ratios for* Symmetric TSP *in the order of their discovery. (R) means randomized; this algorithm computes a random tour, and the approximation ratio compares its expected cost to* OPT.

Approximation Ratio	Year	Reference	Chapter
2	1974	Rosenkrantz, Stearns, and Lewis [1977]	–
$\frac{3}{2}$	1976	Christofides [1976]	1.4
$\frac{3}{2}$	1976	Serdyukov [1978]	1.4
$\frac{3}{2} - 10^{-36}$ (R)	2020	Karlin, Klein, and Oveis Gharan [2021]	10–11
$\frac{3}{2} - 10^{-36}$	2022	Karlin, Klein, and Oveis Gharan [2023]	11.6

pairs of parallel edges in J_1 and J_2, we get two W-joins J_1' and J_2' in G with $c(J_1') + c(J_2') \le c(J_1) + c(J_2) = c(F^*) = \mathrm{OPT}(G, c)$. Hence

$$c(J) \;\le\; \min\left\{c(J_1'), c(J_2')\right\} \;\le\; \tfrac{1}{2}\left(c(J_1') + c(J_2')\right) \;\le\; \tfrac{1}{2}\,\mathrm{OPT}(G, c).$$

We conclude $c(S \mathbin{\dot\cup} J) = c(S) + c(J) \le \frac{3}{2}\,\mathrm{OPT}(G, c)$. □

Adding a matching M in Algorithm 1.23 or a W-join J in Algorithm 1.31 is called *parity correction* because it corrects the parity of every vertex degree (renders it even). Bounding the cost of parity correction will be a central topic in several chapters of this book.

Of course, Theorems 1.24 and 1.32 are equivalent. This bound on the approximation ratio of Christofides' algorithm is tight even for unweighted graph instances: For a complete graph with an even number of vertices, take a spanning tree whose vertices all have odd degree, then we end up with $\frac{3}{2}n - 1$ edges. The special case of Symmetric TSP where $c(e) = 1$ for all $e \in E$ is known as Graph TSP. Today we know a slightly better approximation algorithm for Symmetric TSP (see Table 1.1 and Chapters 10 and 11) and much better approximation algorithms for Graph TSP (see Chapters 12 and 13). However, the following question is still open:

Open Problem 1.33. Find an α-approximation algorithm for Symmetric TSP for some $\alpha \ll \frac{3}{2}$ (say $\alpha \le 1.49$).

Chekuri and Quanrud [2018] found a $(\frac{3}{2} + \varepsilon)$-approximation algorithm that is faster than Christofides' algorithm, for any $\varepsilon > 0$.

1.5 Cycle Cover Algorithm

For Asymmetric TSP, a constant-factor approximation algorithm is much more difficult to obtain, and indeed such an algorithm was not known until 2017. It is trivial to give an n-approximation algorithm: Given an instance (G, c) with $G = (V, E)$, order the cities arbitrarily (say $V = \{v_1, \dots, v_n\}$), and take a shortest v_n-v_1-path and a shortest v_{i-1}-v_i-path for $i = 2, \dots, n$ in (G, c); output the disjoint union of all these paths. Since every tour contains a path from v_{i-1} to v_i for any two cities v_{i-1} and v_i, this algorithm produces a tour at most n times longer than optimal (and this bound is essentially tight; see Exercise 1.12).

The first nontrivial approximation algorithm was found by Frieze, Galbiati, and Maffioli [1982]. It is based on the following concept: A *cycle cover* of a graph $G = (V, E)$ (directed or undirected) is a subset $F \subseteq E$ of edges such that every vertex has degree 2 in (V, F), and in-degree 1 and out-degree 1 in the directed case. In particular, the edge set of a Hamiltonian circuit is a cycle cover, and every cycle cover is Eulerian, but a cycle cover is not necessarily connected.

Lemma 1.34. *Given a simple directed graph $G = (V, E)$ and $c : E \to \mathbb{R}_{\geq 0}$, one can compute a minimum-cost cycle cover in (G, c) or decide that none exists in $O(n^3)$ time.*

Proof. Let G^{12} be the undirected graph that contains two vertices v^1 and v^2 for each $v \in V$ and an edge $\{v^1, w^2\}$ for each $(v, w) \in E$ (with the same cost). There is a one-to-one correspondence between the cycle covers in G and the perfect matchings in G^{12}. Hence the problem reduces to finding a minimum-cost perfect matching in G^{12}, which can be done in $O(n^3)$ time by Theorem 1.19. □

We remark that the graph G^{12} constructed in this proof is *bipartite* (every edge has exactly one endpoint in $\{v^1 : v \in V\}$), and Weighted Matching is easier in bipartite graphs, but this is not important here.

The cycle cover algorithm by Frieze, Galbiati, and Maffioli [1982] is best described for the Asymmetric TSP with Triangle Inequality (see Algorithm 1.35 and Figure 1.5).

Theorem 1.36 (Frieze, Galbiati, and Maffioli [1982])**.** *The cycle cover algorithm (Algorithm 1.35) is a $(\log_2 n)$-approximation algorithm for Asymmetric TSP with Triangle Inequality.*

Proof. At any stage, F is Eulerian, and the algorithm leaves the while-loop only when F is a tour. The number of connected components decreases by at least a factor of 2 in each iteration of the while-loop. Hence there are at most $\lfloor \log_2 n \rfloor$ iterations, and thus by Lemma 1.34, the algorithm runs in $O(n^3 \log n)$ time.

Algorithm 1.35: Cycle Cover Algorithm

Input: an instance (V, c) of Asymmetric TSP with Triangle Inequality

Output: a solution $v_1, \ldots, v_n$

(1) Let $G = (V, \{(v, w) \in V \times V : v \neq w\})$ be the complete directed graph on V. Let $F := \emptyset$.
(2) **while** (V, F) is not connected **do**
(3) Choose one vertex from each connected component of (V, F); let W be the set of these vertices.
(4) Let F_W be a minimum-cost cycle cover in $(G[W], c)$. Set $F := F \dot{\cup} F_W$.
(5) Apply Lemma 1.8 to the tour F and output the resulting solution.

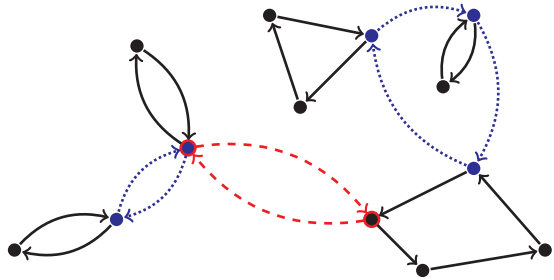

Figure 1.5 Illustrating the cycle cover algorithm. The first iteration chooses a minimum-cost cycle cover (black, solid). The second iteration chooses a representative vertex of each connected component and adds a minimum-cost cycle cover on these (blue, dotted). After two more edges are added in the third iteration (red, dashed), the digraph is connected, and the algorithm terminates.

For any set $W \subseteq V$, the minimum cost of a cycle cover in $G[W]$ is at most $\mathrm{OPT}(V, c)$ because, due to the triangle inequality, we can take shortcuts in any Hamiltonian circuit in G to obtain a Hamiltonian circuit in $G[W]$ without increasing the cost. We conclude that $c(F_W) \leq \mathrm{OPT}(V, c)$ in each iteration, and hence our output has cost at most $\lfloor \log_2 n \rfloor \mathrm{OPT}(V, c)$. □

This bound for the cycle cover algorithm is tight (see Exercise 1.14).

More than 20 years later, the upper bound on the approximation ratio for Asymmetric TSP was improved by a constant factor by Bläser [2008] to $0.99 \log_2 n$, by Kaplan et al. [2005] to $0.842 \log_2 n$, and by Feige and Singh [2007] to $\frac{2}{3} \log_2 n$. We will not present these algorithms here; they are refinements

Table 1.2 *Approximation ratios for* Asymmetric TSP *in the order of their discovery. (R) means randomized; this algorithm computes a random tour, and the approximation ratio compares its expected cost to* OPT. *Moreover,* ε *stands for an arbitrarily small positive constant.*

Approximation Ratio	Year	Reference	Chapter
$\log_2 n$	1980	Frieze, Galbiati, and Maffioli [1982]	1.5
$0.99 \log_2 n$	2002	Bläser [2008]	–
$0.842 \log_2 n$	2003	Kaplan et al. [2005]	–
$\frac{2}{3} \log_2 n$	2006	Feige and Singh [2007]	–
$O(\frac{\log n}{\log \log n})$ (R)	2009	Asadpour et al. [2017]	5
506	2017	Svensson, Tarnawski, and Végh [2020]	6–8
$22 + \varepsilon$	2019	Traub and Vygen [2022]	6–8
$17 + \varepsilon$	2021	this book	6–8

of the cycle cover algorithm. The first sublogarithmic approximation factor was obtained by Asadpour et al. [2017] and will be presented in Chapter 5. Finally, a constant-factor approximation algorithm was discovered by Svensson, Tarnawski, and Végh [2020]. In Chapters 6–8, we will present an improved version of this algorithm. Table 1.2 summarizes the history.

Exercises

1.1 Show that Asymmetric TSP with Triangle Inequality can be solved exactly in $O(n^2 2^n)$ time.
Hint: Choose an arbitrary vertex s. For every set X with $\{s\} \subsetneq X \subseteq V$ and every vertex $t \in X \setminus \{s\}$, compute a list $s = v_1, v_2, \ldots, v_k = t$ that contains every element of X exactly once and minimizes $\sum_{i=2}^{k} c(v_{i-1}, v_i)$.
Note: This technique is called dynamic programming. No faster algorithm is known, even for Symmetric TSP with Triangle Inequality.
(Bellman [1962], Held and Karp [1962])

1.2 Prove that a connected undirected graph contains a walk that traverses each edge exactly once if and only if it has at most two odd-degree vertices.

1.3 Call a tour in a graph *minimal* if no proper subset is a tour in that graph. Prove that a minimal tour in an undirected graph does not contain three parallel edges, and prove that a minimal tour in a directed graph does not

contain $n-1$ parallel edges. Show that these bounds are tight: 2 or $n-2$ parallel edges are possible.

1.4 Consider Dijkstra's algorithm to compute the distance from a vertex s to all other vertices in a digraph $G=(V,E)$ with nonnegative weights $c: E \to \mathbb{R}_{\geq 0}$: Initialize $R := \emptyset$, $d(s) := 0$, and $d(v) := \infty$ for all $v \in V \setminus \{s\}$. Then, while $R \neq V$, select $v \in V \setminus R$ with $d(v)$ minimum, add v to R, and set $d(w) := \min\{d(w), d(v)+c(e)\}$ for all $e=(v,w) \in \delta^+(v)$. Prove that this algorithm is correct.
(Dijkstra [1959])

1.5 Consider the following algorithm. Given a connected undirected graph $G=(V,E)$ and edge costs $c: E \to \mathbb{R}$, initialize $F := \emptyset$. As long as there exists an edge $e \in E \setminus F$ such that $(V, F \cup \{e\})$ contains no circuit, choose such an edge with minimum cost and add it to F.
Prove that this algorithm computes an optimum solution to the Minimum Spanning Tree problem.
Hint: Among all optimum spanning trees, consider one that has as many edges as possible in common with the output of the algorithm.
(Kruskal [1956])

1.6 Let $G=(V,E)$ be a graph and $X \subsetneq V$ such that $G[V \setminus X]$ has more than $|X|$ connected components with an odd number of vertices. Show that then G has no perfect matching. (The converse also holds and is known as Tutte's theorem.)

1.7 Deduce from Theorem 1.21 that the shortest path problem with general weights has no polynomial-time algorithm unless $P = NP$.

1.8 Show that for every even $n \geq 4$, there is a Hamiltonian graph G on n vertices in which every vertex has degree 3 and Christofides' algorithm – run on G with unit weights – may compute a tour with $\frac{3}{2}n-1$ edges.

1.9 The Euclidean TSP is a special case of the Symmetric TSP with Triangle Inequality: Here $V \subsetneq \mathbb{R}^2$, and c is given by the Euclidean distance. Prove that even for Euclidean TSP, Christofides' algorithm is not an α-approximation algorithm for any $\alpha < \frac{3}{2}$.

1.10 In the Rural Postman Problem, we are given a connected undirected graph $G=(V,E)$ with weights $c: E \to \mathbb{R}_{\geq 0}$ and a subset $\bar{E}$ of edges. We ask for a connected (not necessarily spanning) Eulerian multi-subgraph of G that contains at least one copy of every element of $\bar{E}$. Devise a $\frac{3}{2}$-approximation algorithm for the Rural Postman Problem.
(This was first mentioned by Frederickson [1979].)

1.11 Conclude from Corollary 1.30 that there is a polynomial-time algorithm for the Shortest Path problem when the graph G is undirected and

the weights c are conservative (i.e., there is no circuit of negative total weight).

1.12 Prove that the trivial algorithm mentioned at the beginning of Section 1.5 is an $(n-1)$-approximation algorithm for Asymmetric TSP, and prove that this bound is tight.

1.13 Show that a minimum-cost cycle cover in an undirected graph can be computed in polynomial time. Note that it is not allowed to take any edge twice.

Hint: Find a reduction to Weighted Matching by first replacing every edge by a path of three edges and then duplicating every original vertex.

1.14 Show that whenever n is a power of 2, there are instances with n cities for which the cycle cover algorithm (Algorithm 1.35) can produce a solution that is no better than $\log_2 n$ times the optimum.

1.15 Consider the variant of the cycle cover algorithm (Algorithm 1.35) in which F_W in Step (4) is chosen as the edge set of a cycle C in $G[W]$ with minimum mean weight $\frac{c(E(C))}{|E(C)|}$. Karp [1978] showed that a minimum-mean-weight cycle can be computed in $O(n^3)$ time. Prove (by induction on n) that this variant is a $2(1+\frac{1}{2}+\cdots+\frac{1}{n})$-approximation algorithm. (Bläser [2008] attributed this to Kleinberg and Williamson; see also Williamson and Shmoys [2011].)