Chapter 15

Approximation scheme for the traveling salesman problem

In this chapter we present a framework for obtaining approximation schemes. Our primary example application is the traveling-salesman problem, but we introduce the framework using a simpler problem.

15.1 Cutting into small pieces

We start by considering a simple problem for which Baker's methodology does not suffice.

CUT INTO SMALL PIECES

- *Input:* a graph G with edge weights
- *Output:* a set S of edges such that each component of $G - S$ has at most three vertices
- *Goal:* minimize the weight of S

For a reason that will become apparent, we will focus on the unit-weight case, where the goal is to minimize the cardinality of S. This problem (with "three" replaced by a parameter) has been studied because of its application to network epidemiology, under the name "deleting edges to restrict the size of an epidemic." Figure 15.1 shows an example.

If "three" is replaced by "two", the problem is equivalent to finding a maximum matching, which is polynomial-time solvable. However, the case of three (or any greater number) is NP-hard, even for planar graphs. Is there an approximation scheme?

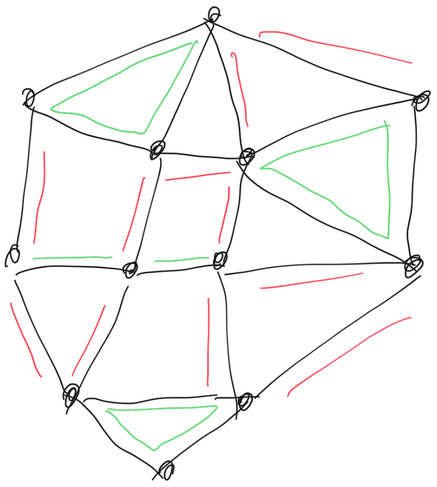


Figure 15.1: The edges to be deleted (the edges of S) are indicated with dashed lines, and the edges to remain are indicated with solid lines. Note that each component of solid edges has at most three vertices.

15.1.1 First attempt

Let's try to apply the methodology of Section 14.7. According to that methodology, the key is finding a decomposition of the graph into parts so that (1) each part has small branchwidth, (2) a solution for the whole induces a solution for each part (*whole-to-parts*), and (3) the union of solutions for the parts is a solution for the whole (*parts-to-whole*).

The good news is that the whole-to-parts property holds for Cut Into Small Pieces. The bad news is that a decomposition of a graph does not ensure the parts-to-whole property.

Consider a decomposition such as that used in Section 14.7.5:

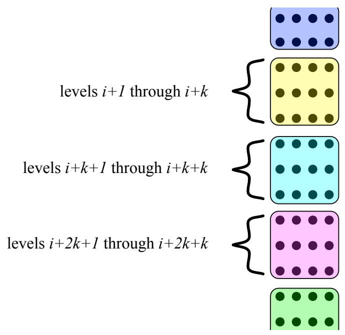
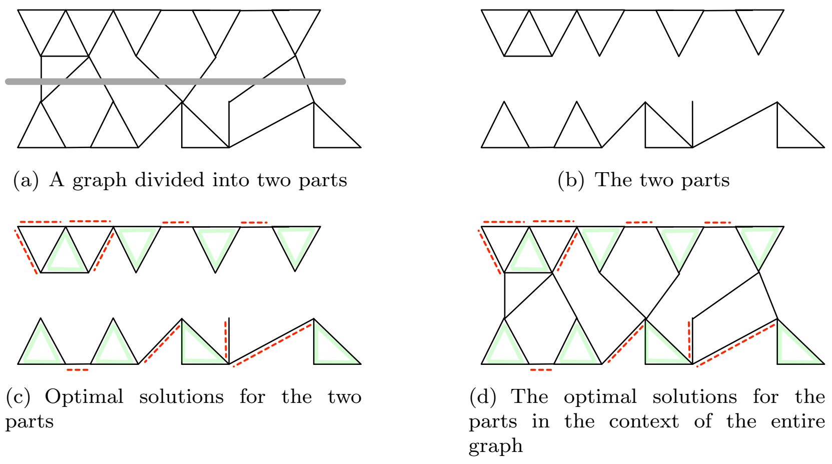

levels $i+1$ through $i+k$
levels $i+k+1$ through $i+k+k$
levels $i+2k+1$ through $i+2k+k$

Figure 15.2(a) shows a graph decomposed in this way, and Figure 15.2(b)


(a) A graph divided into two parts

(b) The two parts

(c) Optimal solutions for the two parts

(d) The optimal solutions for the parts in the context of the entire graph

Figure 15.2: Failure of the parts-to-whole property

shows the resulting parts. Figure 15.2(c) shows an optimal solution for each part. Note that when the solution edges are removed, each resulting component has at most three vertices. However, as shown in Figure 15.2(d), when we restore the edges removed to split the graph into parts, components are merged to form new components of size greater than three. Thus the union of solutions for the parts is not a feasible solution for the whole: the parts-to-whole property does not hold.[1] Recall that in every approximation scheme we have seen so far, the solution for the whole is obtained by taking the union of the solutions for the parts. Because this approach does not work for CUT INTO SMALL PIECES, we conclude that the approximation-scheme methodology of Chapter 14 does not suffice for this problem.

15.1.2 Deletion Decomposition

When we divide the graph into parts, we must also take into account the edges *between* the parts. Consider the following approach:

1. Decompose the input graph into vertex-disjoint subgraphs.
2. Find optimal solutions in each of the subgraphs.
3. Return the union of these solutions together with some of the edges between the subgraphs.

According to Baker's methodology, the algorithm would consider some k different decompositions, each obtained by removing a different set E_i of edges

[1]In other forms of decomposition, the two parts share vertices and maybe even edges—but the same issue would arise.

between the subgraphs. Because $E_0, \ldots, E_k$ are disjoint, for at least one value of $i \in \{0, \ldots, k-1\}$, the weight of $E_i \cap \text{OPT}$ is at most $1/k$ times the weight of OPT.

Unfortunately, this fact is irrelevant here. Even if somehow the algorithm knew which edges of E_i belonged to OPT, including only those edges in the solution in Step 3 would likely not lead to a feasible solution because the solutions obtained in Step 2 for each of the subgraphs are not likely to match up with the edges of OPT $\cap E_i$ to form a feasible solution for G. For example, in Figure 15.2(d), unless all the edges of E_i are removed, a component of size greater than three will result.

The analysis must allow for the possibility that *all* the edges of E_i must be included in the algorithm's output. Using the same averaging argument as was used in analyzing Baker's algorithm, we can ensure that there is one choice of i for which the weight of E_i is at most $1/k$ times the weight of the entire graph. Let us say that the algorithm selects this choice. Because of the whole-to-parts property, the weight of the union of solutions to parts (as found in Step 2 is at most the weight of OPT. This analysis therefore shows that the weight of the algorithm's solution is

$$\text{weight(OPT)} + \frac{1}{k}(\text{weight of input graph})$$

For constant k, this does not suffice to ensure that the algorithm is an approximation scheme; the weight of the input graph could greatly exceed the weight of OPT.

Consider therefore the special case in which all edge weights are one, and that furthermore the input graph is simple.

Lemma 15.1.1. *For any connected strict graph with more than three vertices, there is a partition of the edges into sets such that, for any solution S to* Cut Into Small Pieces*, S contains at least one-third of the edges in each set.*

Corollary 15.1.2. *For any connected strict graph G with more than three vertices, for any set S of edges whose removal results in components with at most three vertices, $|S| \geq |E(G)|/3$.*

The corollary states that (in the case of a simple graph with unit weights) the weight of the graph is at most three times the weight of OPT. Therefore we can use $k = \lceil 3/\epsilon \rceil$ in the algorithm described above, and be guaranteed that the weight of E_i be at most ϵ times the weight of OPT, and that therefore the algorithm's output solution is at most $1 + \epsilon$ times the weight of OPT.

To facilitate stating the algorithm more formally, we first abstract the procedure for decomposing the input graph. Because the decomposition yields vertex-disjoint subgraphs each having bounded branchwidth, the union of these subgraphs has bounded branchwidth.

Lemma 15.1.3 (Deletion-Decomposition Lemma)**.** *There is a linear-time algorithm that, for any positive integer k and planar graph G, outputs a k-part* *subpartition* $E_0 \cup \cdots \cup E_{k-1}$ *of* $E(G)$ *such that, for* $i = 0, \ldots, k-1$, $G - E_i$ *has branchwidth at most* $2k$.

We can now state more formally the approximation scheme for unit-weight Cut Into Small Pieces in simple planar graphs. Assume the input graph G_0 has $|V(G_0)| > 3$, for otherwise the empty set is the optimal solution.

1. For $k = \lceil 3\epsilon^{-1} \rceil$, use the algorithm of the Deletion-Decomposition Lemma to compute the k-part subpartition $E_0 \cup \cdots \cup E_{k-1}$ of $E(G_1)$. Let $q = \text{minarg }_i |E_i|$. Let $G_1 = G_0 - E_q$.

2. Find the optimal solution for G_1.

3. Return the union of this solution and the edges of $E_{\hat{i}}$.

Because $E_0, \ldots, E_{k-1}$ are disjoint, $\sum_i |E_i| \leq |E(G_0)|$. By averaging, there is some integer q for which $|E_q| \leq (1/k)|E(G_1)|$. Corollary 15.1.2 implies that $|\text{OPT}| \geq |E(G_0)|/3$. Because $k = \lceil 3\epsilon^{-1} \rceil$, we infer that $|E_q| \leq \epsilon|\text{OPT}|$.

By the whole-to-parts property, the solution found in Step 2 has cardinality at most $|\text{OPT}|$. Therefore the solution returned in Step 3 has cardinality at most $(1+\epsilon)|\text{OPT}|$. This proves that the algorithm is an approximation scheme.

Problem 15.1. *Prove Lemma 15.1.1.*

Problem 15.2. *Show that, for some constant* c*, there is a* $c^w n$ *algorithm for* Cut Into Small Pieces *in graphs with branchwidth* w*.*

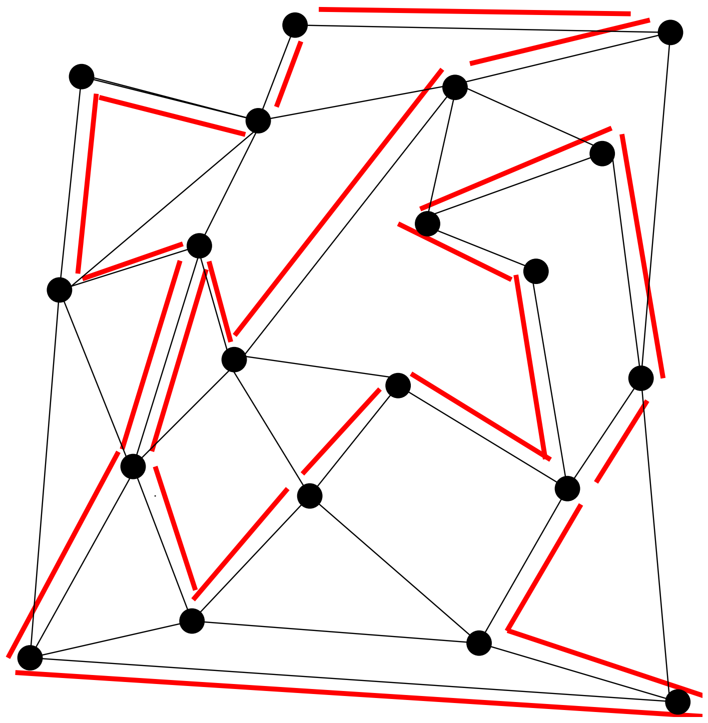
It follows from the Deletion-Decomposition Lemma and Problem 15.2 that the approximation scheme takes linear time.

It is crucial that weights are unit; the approximation analysis depends on Corollary 15.1.2, which does not hold for arbitrary weights.

15.2 The traveling salesman problem

Next we study the most famous hard optimization problem in graphs, the *traveling salesman problem* (TSP).

An *undirected traveling-salesman tour* in a graph is a closed walk of darts that visits every vertex at least once:



Given a graph with edge-weights, the goal of the TSP is to find a traveling salesman tour of minimum weight. For now, we do not consider the corresponding *directed* problem.

Historically, the TSP has been the focus of much research in combinatorial optimization. Often when a new technique is introduced in the literature, it is introduced in application to the TSP. Because planar graphs are useful in modeling road networks, algorithmic insights about TSP in planar graphs can be relevant to problems in mobility. We shall return to this point in the next chapter.

We will see that the problem can be reformulated so as to remove the ordering aspect of a solution. Let G be the input graph, and let W be a traveling-salesman tour. We can almost interpret W as a subgraph of G_0. We say "almost" because an edge of G could occur multiple times in W. A *multisubgraph* of G is a graph M obtained from G by replacing each edge of G by some number of copies of that edge (with the same endpoints). That is, each edge of G can belong to M with any multiplicity. The *degree* of a vertex of G with respect to M is the sum of multiplicities in M of edges incident to the vertex.

Lemma 15.2.1 (Euler Lemma)**.**

1. *For any graph G and closed walk W in G, let M be the multisubgraph in which an edge's multiplicity is its multiplicity in W. Then the edges of (G, M) are connected and every vertex has even degree.*

2. *There is a linear-time algorithm that, for any multisubgraph M such that the edges are connected and every vertex has even degree, finds a walk W in which each edge's multiplicity is the same as in M.*

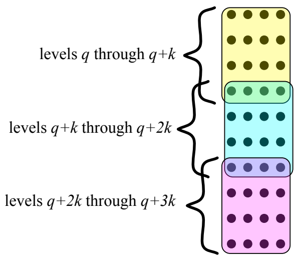
We say that a multisubgraph M of G is *Eulerian* if it is connected and every vertex of G has even degree with respect to M.

It follows from the Euler Lemma that the problem of finding a minimum-weight tour visiting a given set S of vertices is equivalent to the problem of finding a minimum-weight Eulerian multisubgraph M incident to every vertex in S.

Moreover, consider any Eulerian multisubgraph M. If the multiplicity in M of some edge e is k where k is greater than two, changing the multiplicity to $k \bmod 2$ also results in an Eulerian multisubgraph incident to the same set of vertices. Thus in addressing the minimum-weight Eulerian problem, we can restrict attention to multisubgraphs in which the maximum multiplicity is two. We call such a multisubgraph a *bisubgraph.* Thus the TSP is equivalent to the problem of finding a minimum-weight Eulerian bisubgraph. We refer to this problem as EULERIAN BISUBGRAPH.

15.3 Approximating unit-weight Eulerian Bisubgraph

One might hope that EULERIAN BISUBGRAPH is amenable to the approximation-scheme methodology described in Chapter 14. In particular, consider the decomposition used in Section 14.7.3 for VERTEX COVER:



The problem EULERIAN BISUBGRAPH satisfies the parts-to-whole property with respect to this decomposition. Unfortunately, it does not satisfy the whole-to-parts property. The union of minimum-weight Eulerian bisubgraphs of the parts could be far more expensive than a minimum-weight Eulerian bisubgraph of the whole. This means that *none* of the decompositions we considered in Chapter 14 will work; the methodology of that chapter is not applicable to EULERIAN BISUBGRAPH.

Instead, recall our approximation algorithm for the unit-weight version of CUT INTO SMALL PIECES. It uses a method we call *approximation through deletion decomposition*, which we reformulate as follows. Given the input graph G,

- Obtain a subpartition of the edges into k parts so that deleting any part results in a graph of branchwidth at most $2k$. Let S be the part of smallest weight.
- Find an optimal solution in $G - S$.
- Add a multisubset of the edges of S to the solution for $G - S$ to obtain a solution for G.

The goal of Cut into Small Pieces is to select edges to disconnect the input graph, which is why deletion of edges makes sense. The goal of Eulerian Bisubgraph is, loosely speaking, to select edges to connect the graph, which suggests that edge *contraction* might be more suitable. Consider the following approach, which is called *approximation through contraction decomposition.* Let G be the (planar) input graph.

- ***Reduce width:*** Obtain a subpartition of the edges of G into k parts so that contracting any part results in a graph of branchwidth at most $2k$. Let S be the part of smallest weight.
- ***Solve:*** Find an optimal solution in G/S.
- ***Lift:*** Add a multisubset of the edges of S to the solution for G/S to obtain a solution for G.

Because edge contraction is the dual of edge deletion, applying *approximation through contraction decomposition* to a graph G is essentially the same as applying *approximation through deletion decomposition* to the dual of G. For the *Reduce width* step, we use the analogue of the Deletion-Decomposition Lemma (Lemma 15.1.3):

Lemma 15.3.1 (Contraction-Decomposition Lemma)**.** *There is a linear-time algorithm that, for any positive integer k and planar graph G, outputs a k-part subpartition $E_0 \cup \cdots \cup E_{k-1}$ of $E(G)$ such that, for $i = 0, \ldots, k-1$, G/E_i has branchwidth at most $2k + 1$ (and E_i contains no cycles).*

The proof is discussed in Section **??**.

The other steps of *approximation through contraction decomposition* are specific to the optimization problem being addressed. In addressing TSP, the *Solve* step (discussed in Section **??**) consists in finding a minimum-weight Eulerian bisubgraph of a graph of small branchwidth. The *Lift* step (discussed in Section 15.8) consists in considering each compressed edge in turn, and incorporate zero, one, or two copies of it into the solution, whichever is needed to keep the solution Eulerian.

Consider the running time. The *Reduce width* step takes linear time regardless of the optimization problem addressed. In addressing TSP, the *Lift* step takes linear time, and the *Solve* step takes at most $c^w n$ time for some constant c where w is the branchwidth. Thus for any $\epsilon > 0$ the overall algorithm takes linear time if the width w can be bounded by a constant depending on ϵ.

Consider the approximation analysis. We follow the pattern used in analyzing the algorithm for Cut into Small Pieces. In the *Reduce width* step, the part S of smallest weight has weight at most $1/k$ times the total weight of the graph. For now, let us restrict attention to the unit-weight problem. We can assume without loss of generality that the input graph G is strict. By sparsity, therefore, $|E(G)| \leq 3|V(G)|$. We choose $k = \lceil 6\epsilon^{-1} \rceil$, so $|S| \leq \frac{1}{2}\epsilon|V(G)|$. Therefore the weight of the multisubset of edges added in the *Lift* step is at most $\epsilon|V(G)|$. On the other hand, in any Eulerian bisubgraph each vertex has degree at least two, so the size of any Eulerian bisubgraph of G is at least $|V(G)|$. Therefore the weight of edges added in the *Lift* step is at most ϵ times the optimal value. The optimal value for G/S is no more than the optimal value for G, so the weight of the solution found in the *Solve* step is at most the optimal value. Thus the algorithm outputs a solution whose weight is at most $1 + \epsilon$ times the optimal value. The algorithm is an approximation scheme.

Returning to the running time, the width of G/S is at most $2k + 1$, which is $2\lceil 6/\epsilon \rceil + 1$. Thus the *Solve* step takes at most $c^{2\lceil 6/\epsilon \rceil + 1} n$ time, so the overall algorithm takes linear time when ϵ is considered a constant.

15.4 Beyond unit-weight graphs: a sparsifier

The algorithm of Section 15.3 is an approximation scheme only for unit-weight planar graphs. The approximation analysis of that algorithm requires that k be a constant such that a $1/k$ fraction of the weight of the input graph is a lower bound on the weight of the optimal solution. For arbitrary-weight graphs, there is no such constant k.

Fortunately, there is an algorithm we can use to thin out the input graph before applying the algorithm of Section 15.3.

Let P be an optimization problem, e.g. TSP, and let $\mathrm{OPT}(G, w(\cdot))$ be the optimal value for an edge-weight graph G. An *(α, β)-sparsifier for P* is an algorithm that, given G, outputs a subgraph H such that

- $\mathrm{OPT}(H) \leq \alpha\, \mathrm{OPT}(G)$ and
- the weight of H is at most $\beta\, \mathrm{OPT}(G)$.

In Section 15.9, we show that, for any $\epsilon > 0$, there is a linear-time $(1+\epsilon, 1+2/\epsilon)$-sparsifier for TSP. We therefore obtain a linear-time approximation scheme for TSP in planar graphs with edge weights.

The same approach can be used to address a variety of optimization problems involving selecting a subgraph or bisubgraph to achieve connectivity. For example, consider the Steiner tree problem. The input is a graph with edge-weights and a subset S of vertices; the output is a subgraph in which S is connected; the goal is to minimize the weight of the subgraph. The approach outlined for TSP can be adapted to achieve an $O(n \log n)$ approximation scheme for Steiner Tree in planar graphs.

In Chapter 16, we show how the approach can be adapted to a variant of TSP, Steiner TSP, in which the input specifies a set S of vertices that must be visited by the tour (the tour can travel through other vertices as well). In Chapter 17, we show how to address PRIZE-COLLECTING STEINER TSP, in which the input additionally includes a function assigning a cost to each vertex in S; the output is a tour visiting a subset of the vertices in S (and some other vertices); and the goal is to to minimize the weight of the tour plus the cost of the vertices not visited.

In each case, the algorithm has the following form:

- *Sparsify:* Select a subgraph G_1 of the input graph G_0 such that $\text{OPT}(G_1) \leq (1+\epsilon)\text{OPT}(G_0)$ and the weight of G_1 is at most β times $\text{OPT}(G_0)$, where β is a constant.

- *Compress:* Obtain a subpartition of the edges of G_1 into k parts such that compressing any part results in a graph of branchwidth at most $2k$. Let S be the part of smallest weight.

- *Solve:* Find an optimal solution in G_1/S.

- *Lift:* Transform the solution for G_1/S into a solution for G_0.

The *Compress* step is an application of the Compression-Decomposition Lemma. The *Solve* step involves a dynamic program of the kind we have by now seen many times. This step obviously depends on the problem being addressed but is generally straightforward. Similarly the *Lift* step is generally straightforward. The *Sparsify* step, on the other hand, has often required the development of new techniques. Chapters 16 and 17 outline two such techniques.

For problems that involve selecting edges to *remove* in order to *reduce* connectivity, the approach does not apply directly:

- Deletion of edges in the *Sparsify* step only reduces the OPT, so sparsification does not make sense;

- Compression of edges in the *Compress* step can greatly increase OPT, so there is no way that the solution found in the *Solve* step can be lifted to obtain an approximately optimal solution for the input graph.

However, consider applying the approach to the dual of the graph, or, equivalently, swapping edge deletion with edge compression:

- *Dual Sparsify:* Obtain G_1 from the input graph G_0 by contracting a set of edges chosen so that $\text{OPT}(G_1) \leq (1+\epsilon)\text{OPT}(G_0)$ and the weight of G_1 is at most $\beta(n)$ times $\text{OPT}(G_0)$.

- *Delete* Obtain a subpartition of the edges of G_1 into k parts such that deleting any part results in a graph of branchwidth at most $2k$. Let S be the part of smallest weight.

- *Solve:* Find an optimal solution in $G_1 - S$.

- *Lift:* Transform the solution for $G_1 - S$ into a solution for G_0.

This approach has been used, e.g. to obtain a kind of approximation scheme for Graph Bisection; in the *Dual Sparsify* step, the weight of G_1 is $O(\log n)$ times $\mathrm{OPT}(G_0)$ where n is the size of G_0. The *Delete*, *Solve*, and *Lift* steps correspond to the steps of our approximation scheme for unit-weight Cut into Small Pieces. For that problem, no dual sparsification step is known, so the approach does not yield an approximation scheme for Cut into Small Pieces with arbitrary weights.

****

Finally, putting the pieces together, the algorithm is as follows:

1. Find a spanner G for TSP in the input graph G_0.
2. Find disjoint sets $E_1, \ldots, E_k$ such that, for any i, G/E_i has branchwidth at most $2k$.
3. Let $i^* = \text{minarg }_i \text{weight}(E_i)$.
4. For each connected component of G/E_{i^*}, find a minimum-weight tour.
5. Lift the union of the tours to G by incorporating at most two copies of each edge of E_{i^*}.

Define the parameter k by $k = 2g(\epsilon)\epsilon^{-1}$ where c is the constant in Inequality **??**.

Then the weight of the solution obtained in Step 5 is at most

$$\begin{aligned}
\mathrm{OPT}(G/E_{i^*}) + 2\,\text{weight}(E_{i^*}) &\leq \mathrm{OPT}(G) + 2\,\frac{1}{k}\,\text{weight}(G) \\
&\leq (1+\epsilon)\mathrm{OPT}(G_0) + 2\,\frac{1}{2c\epsilon^{-1}}c\,\mathrm{OPT}(G_0) \\
&\leq (1+\epsilon)\mathrm{OPT}(G_0) + \epsilon\,\mathrm{OPT}(G_0) \\
&\leq (1+2\epsilon)\mathrm{OPT}(G_0)
\end{aligned}$$

15.5 Compression decomposition and contraction decomposition

We consider how to find the sets of Step 1.

Lemma 15.5.1 (Compression-Decomposition). *For any positive integer k and planar graph G, there is a k-part subpartition $E_1 \cup \cdots \cup E_k$ of $E(G)$ such that, for $i = 1, \ldots, k$, the graph obtained from G by compressing E_i has branchwidth at most $2k$.*

Proof. We adapt the procedure used for multiterminal cut in Section **??**, appying it to the dual G^* of G.

Execute breadth-first search on G^*. For $i = 0, 1 \ldots, k$, let E_i be the set of edges of G^* whose levels are congruent mod k to i.

The procedure above separates G^* into connected components each having the form

$$G^*[V^*_{\ell+1} \cup V^*_{\ell+2} \cup \cdots \cup V^*_{\ell+k}]$$

where V^*_ℓ is the set of vertices of Gj of level ℓ. By the BFS-Branchwidth Lemma (Lemma 14.7.1), each such connected component has branchwidth at most $2k$.

Deleting edges from the dual graph G^* corresponds to compressing these edges in the primal graph G. This shows that G/E_i has branchwidth at most $2k$. □

Recall that compression differs from contraction on self-loops (cut-edges) in the dual. On such edges, compression can disconnect a graph. Since this is sometimes undesirable, we show that such compressions can be avoided.

Lemma 15.5.2 (Contraction-Decomposition Lemma)**.** *For any positive integer k and planar graph G, there is a k-part subpartition $A_1 \cup \cdots \cup A_k$ of $E(G)$ such that, for $i = 1, \ldots, k$, the graph obtained from G by contracting A_i has branchwidth at most $2k + 1$.*

Proof. Let $E_1 \cup \cdots \cup E_k$ be the subpartition of $E(G)$ given by the Compression-Decomposition Lemma. Fix a value of i. Initialize $G' := G$. For each edge e of E_i (in arbitrary order), if e is not currently a self-loop of G' then contract it (which in this case is the same as compressing it). Let A_i be the set of edges contracted, and let B_i be the set of edges of E_i that remain in G'.

Consider an edge $e \in B_i$ (so e is a self-loop). For edges $e_1, e_2 \in E(G')$, if e_1 is enclosed by e and e_2 is not enclosed, every e_1-to-e_2 path includes the common endpoint of e, which shows that e_1 and e_2 are not in the same biconnected component of G'. Therefore, in view of the discussion in Section 4.6.1, compressing e exactly preserves all biconnected components of G' except that it removes the singleton biconnected component consisting of e. By Lemma 15.5.1, each biconnected component of G/E_i has branchwidth at most $2k$, so the same holds for each biconnected component of G/A_i. By Lemma 14.5.3, therefore, G/A_i has branchwidth at most $2k + 1$. □

15.6 The framework

The approach described in Section **??** can be used for a variety of optimization problems. In this section, we outline a general framework.

Let G_0 be the input graph, and let weight($\cdot$) be an assignment of weights to the edges. For any graph G such that $E(G) \subset E(G_0)$, we denote by OPT(G) the optimum value of the optimization problem on graph G.

An algorithm in this framework can take two forms that are exactly dual to each other. We start by presenting the form that is applicable to the traveling-salesman problem (TSP). This is applicable to minimization problems (such as TSP) that satisfy the following condition:

Contraction-Monotonicity: Contracting edges cannot increase the optimum value.

Let $\epsilon > 0$ be an error parameter.

Spanner step: Let G_1 be a graph obtained from G_0 by edge-deletions such that

1. $\text{weight}(G_1) \leq g(\epsilon)\text{OPT}(G_{in})$, and
2. $\text{OPT}(G_1) \leq (1+\epsilon)\text{OPT}(G_{in})$

where $g(\epsilon)$ is some function of ϵ. [5]

Thinning step: Apply the Contraction-Decomposition Lemma to G_1 to obtain a subpartition $A_1 \cup \cdots \cup A_k$ of the edges, where $k = \epsilon\, g(\epsilon)$. Let A_q be the part of smallest weight. Let $G_2 = G_1/A_q$.

Branchwidth step: Find the optimal solution in G_2, using the fact that G_2 has branchwidth at most $2k+1$.

Lifting step: Convert the optimal solution found in the previous step to a solution for G_1 by incorporating some of the edges of A_q, increasing the weight by at most $c\,\text{weight}(A_q)$, where c is a constant.

Assume for now that these steps can be carried out for a particular optimization problem. We assume that the Lifting step increases the weight by at most $c\,\text{weight}(A_q)$, so

$$\begin{aligned}
&\text{weight of the output solution}\\
&\leq \text{OPT}(G_2) + c\text{weight}(A_q)\\
&\leq \text{OPT}(G_1) + c\text{weight}(A_q) \text{ by Contraction-Monotonicity}\\
&\leq (1+\epsilon)\text{OPT}(G_0) + c\,\text{weight}(A_q) \text{ by Property 2 of the Spanner step}\\
&\leq (1+\epsilon)\text{OPT}(G_0) + c\cdot\frac{1}{k}\text{weight}(G_1)\text{by an averaging argument in the Thinning step}\\
&\leq (1+\epsilon)\text{OPT}(G_0) + c\cdot\frac{1}{k}g(\epsilon)\text{OPT}(G_0) \text{ by Property 1 of the Spanner step}\\
&\leq (1+\epsilon)\text{OPT}(G_0) + c\,\epsilon\text{OPT}(G_0) \text{ by choice of } k\\
&\leq (1+(c+1)\epsilon)\text{OPT}(G_0)
\end{aligned}$$

Thus the output solution is approximately optimal.

As described in Section **??**, for TSP in unit-length graphs, the Spanner step need only delete parallel edges and self-loops, for then $\text{weight}(G_1) \leq 3\text{OPT}(G_0)$ and $\text{OPT}(G_1) = \text{OPT}(G_0)$. For this problem, we set $k = 3\epsilon^{-1}$. The constant c in the Lifting step is 2, so the algorithm returns a tour of length $1 + 2\epsilon$ times optimum.

For TSP with arbitrary nonnegative lengths, we show in Section 15.9 how to carry out the Spanner step with $g(\epsilon) = 2\epsilon^{-1} + 1$, so we set $k = 2\epsilon^{-2} + \epsilon^{-1}$.

Again the constant c is 2, so the algorithm returns a tour of length $1+3\epsilon$ times optimal.

The running time is dominated by the Branchwidth step. A straightforward algorithm for TSP in an n-vertex graph of branchwidth w takes time $2^{O(w \log w)}n$.

15.6.1 Dual framework

Compare the approach proposed for TSP to that proposed for multiterminal cut. They are really just duals of each other. The analogue of Contraction-Monotonicity that applies to multiterminal cut is:

> **Deletion-Monotonicity:** Deleting edges cannot increase the optimum value.

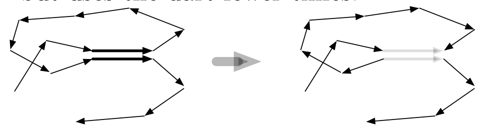
In the dual form of the framework, the spanner step *contracts* edges, whereas the thinning step uses the Deletion-Decomposition Lemma to *delete* edges.

15.7 Properties of tours

In this section, we outline some properties of tours that help us in the Branchwidth step and the Lifting step.

Lemma 15.7.1. *For any walk W in a graph, there is a walk W' that visits the same vertices as W, such that every edge used by W' is used by W, and occurs at most twice in W'.*

Proof. Let W be a walk in G, and suppose some dart d occurs at least twice in W. Write $W = W_1\ d\ W_2\ d$. Then W_1 rev(W_2) is a walk of G that visits the same vertices as W but uses the dart fewer times.



Repeating this step yields the lemma. □

Lemma 15.7.1 shows that, in seeking the minimum-length closed walk visiting a given set of vertices, we can restrict ourselves to considering walks in which each edge occurs at most twice.

Proposition 15.7.2. *For any tour in a planar graph, there exists a tour that visits the same vertices and comprises the same darts in the same multiplicities, and does not cross itself.*

Proof. Suppose $\hat{W} = W_1\ a\ W\ b\ W_2\ c\ W\ d$ is a closed walk where $c\ W\ d$ forms a crossing configuration with $a\ W\ b$. Then $d\ W_1\ a$ rev$(c\ W_2\ b)$ rev$(W)\ W$ is a closed walk visiting the same nodes and comprising the same darts in the same multiplicities, and with one fewer crossing configurations. □

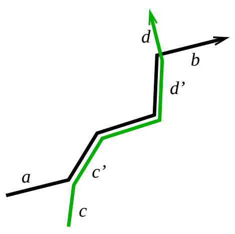


Figure 15.3: The light walk forms a crossing configuration with the bold walk.

Proposition 15.7.2 shows that we can restrict our attention to non-self-crossing walks.

15.8 Lifting for TSP

The Lifting step is supposed to start with a solution to TSP for the contracted graph G_1/A_q, and produce a solution for the uncontracted graph G_1. We reformulate this, based on Section15.7, as starting with an Eulerian bi-subgraph for G_1/A_q and producing an Eulerian bi-subgraph for G_1. The procedure, LiftMany simply considers each contracted edge in turn, uncontracts it, and incorporates one or two copies of it into the solution.

Each iteration is carried out by a procedure LiftOne(G, e, B) that, given an Eulerian bisubgraph B of $G/\{e\}$, returns an Eulerian bisubgraph B' of G such that $B' - \{e\} = B$. (See Figure 15.4.) We use the notation $B \cup \{e\} \cup \{e\}$ to indicate the multiset obtained from B by adding two copies of e.

```
def LiftOne($G, e, B$):
  if the endpoints of $e$ in $G$ have odd degree in $B$
    return $B \cup \{e\}$
  return $B \cup \{e\} \cup \{e\}$

def LiftMany($G, S, B$):
  if $S = \emptyset$
  return $B$
  let $e$ be an edge of $S$
    return LiftOne($G, e,$ LiftMany($G/\{e\}, S - \{e\}, B$))
```

Lemma 15.8.1 (Correctness of LiftOne)**.** *If B is an Eulerian bisubgraph of $G/\{e\}$ and e is not a self-loop then* LiftOne(G, e, B) *returns an Eulerian bisubgraph of G.*

Proof. Since B is an Eulerian bisubgraph of $G/\{e\}$, it must contain at least one edge incident to at least one endpoint of e in G. Therefore $B \cup \{e\}$ connects each connected component of G. Since in $G/\{e\}$ the degree in B of each vertex is even, in G every vertex has even degree in B except possibly the endpoints of e. If even the endpoints have even degree then $B \cup \{e\}$ is an Eulerian bisubgraph of G. If not, then both endpoints must have odd degree in B since the sum of their degrees is the degree of the vertex resulting from contracting e. Hence $B \cup \{e\} \cup \{e\}$ is an Eulerian bisubgraph. □

15.9 Spanner

Theorem 15.9.1. *There is a linear-time algorithm that, given a planar graph G_0 with edge-weights and any $\epsilon > 0$, outputs an edge subgraph G such that*

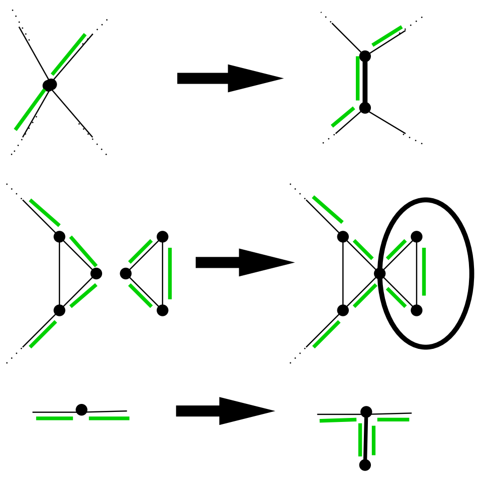


Figure 15.4: Examples of lifting. The Eulerian bisubgraph is indicated with light lines. In the top diagram, only one copy of the formerly compressed edge (the bold line) must be incorporated. In the second diagram, no copies are needed. In the third diagram, two copies are needed.

A1: *weight$(G) \leq (1+2\epsilon^{-1})MST(G_0)$, where $MST(G_0)$ is the weight of the minimum spanning tree of G_0, and*

A2: *for every pair of vertices u and v,*

$$\begin{aligned} &\text{minimum weight of a } u\text{-to-}v \text{ path in } G \\ \leq\ & (1+\epsilon)\cdot \text{minimum weight of a } u\text{-to-}v \text{ path in } G_0 \end{aligned} \tag{15.1}$$

Lemma 15.9.2. *Properties A1 and A2 imply Properties S1 and S2 of Section* **??** *with $\rho_\epsilon = 1+2\epsilon^{-1}$.*

Proof. Because a tour includes a spanning tree, $MST(G) \leq \text{OPT}(G)$. Hence Property A1 implies that Property S1 of Section **??** is achieved with $\rho_\epsilon = 1+2\epsilon^{-1}$.

Now we show that Property A2 implies Property S2, i.e. that $\text{OPT}(G) \leq (1+\epsilon_0)\text{OPT}(G_0)$. (This argument was used in [**?**].) Let T_0 be an optimal tour of G_0. For each edge uv of T_0 that is not in G, there is a u-to-v path in G of weight at most $(1+\epsilon)$weight(uv); replace uv in T_0 with that path. The result of all the replacements is a tour T_1 whose weight is at most $1+\epsilon$ times that of T_0. This shows $\text{OPT}(G) \leq (1+\epsilon)\ \text{OPT}(G_0)$. □

The algorithm of Theorem 15.9.1 is as follows.

```
define SPANNER($G_0, \epsilon$):
    let $x[\cdot]$ be an array of numbers, indexed by edges
    find a minimum spanning tree $T$ of $G_0$
    assign $x[e]$ := weight($e$) for each edge $e$ of $T$
    initialize $S$ := {edges of $T$}
    let $T^*$ be the dual tree, rooted at the infinite face
    for each edge $e$ of $T^*$, in order from leaves to root
      let $f_e$ be the face of $G_0$ whose parent edge in $T^*$ is $e$
      let $e$=$e_0, e_1, \ldots, e_s$ be the sequence of edges comprising $f_e$
      $x_{\text{omit}} := \sum_{i=1}^{s} x[e_i]$
      if $x_{\text{omit}} > (1+\epsilon)$weight($e$)
        then add $e$ to $S$ and assign $x[e]$ := weight($e$)
        else assign $x[e] := x_{\text{omit}}$
    return $S$
```

As we saw in Chapter 4, the minimum spanning tree of G_0 can be found in linear time.

Now we address correctness of the procedure. Say an edge e is *accepted* when e is assigned to S, and *rejected* if e is considered but not assigned to S.

Lemma 15.9.3. *In the for-loop iteration in which e is considered, for every other edge e_i of f_e, $x[e_i]$ has been assigned a number.*

Proof. The face f_e has only one parent edge in T^*, and it is e. For every other edge e_i of f_e, either e_i belongs to T or e_i is a child edge of f_e in T^*. □

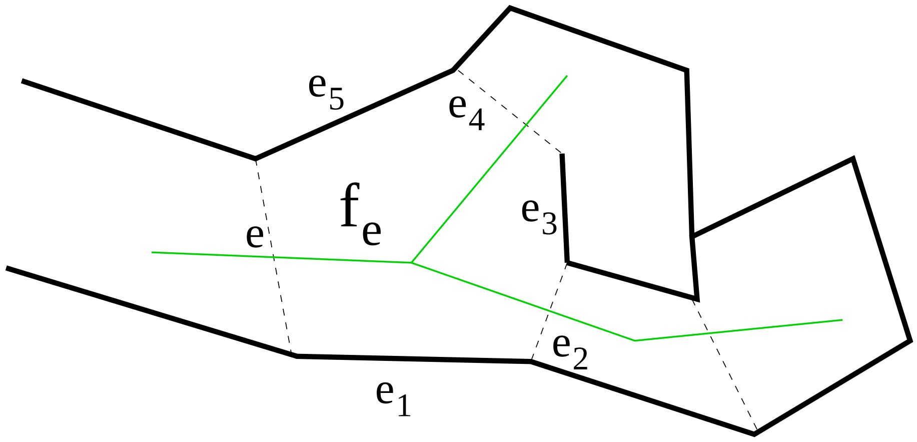


Figure 15.5: Diagram showing part of dual tree (in light edges) and primal tree (in dark edges) and primal nontree edges (dashed): e_2 and e_4 are child edges of e in the dual tree. The face f_e is indicated.

For any edge e of G_0 not in T,

- let $\hat{G}_e$ denote the subgraph of G_0 consisting of accepted edges together with e,
- let $\hat{f}_e$ denote the face of $\hat{G}_e$ that contains e and encloses f_e,
- let $\hat{W}_e$ denote the walk formed by the sequence of edges comprising $\hat{f}_e$ not including e itself, and
- let $P_e = \begin{cases} e & \text{if } e \text{ is accepted} \\ \hat{W}_e & \text{otherwise} \end{cases}$

Note that each of $\hat{W}_e$ and P_e has the same endpoints as e. For an edge e of T, define $P_e = e$. The basic argument of the following lemma comes from [**?**].

Lemma 15.9.4. *For any edge e of G_0, not in T,*

1. *every edge of $\hat{f}_e$ is either in T or is a descendant of e in T^*, and*
2. *$\hat{W}_e = P_{e_1} \ \cdots \ P_{e_s}$, where $e_1 \ \ldots \ e_s$ is the walk consisting of the edges comprising f_e other than e.*

Proof. by induction. Consider the case in which e is a leaf-edge of T^*. Let f be the corresponding leaf-node in G_0^*. Because f is a leaf, the only incident edge that is in T^* is e itself, so $e_1, \ldots, e_s$ belong to T. All these edges are accepted, proving Part 1. To prove Part 2, note that $W_e = e_1 \ \cdots \ e_s$ and that $P_{e_i} = e_i$ for $i = 1, \ldots, s$. Thus the lemma holds for e.

Consider the case where e is not a leaf. Let $\hat{G}_{e+}$ be the subgraph of G_0 consisting of accepted edges together with $e, e_1, \ldots, e_s$. For each e_i, recall that $\hat{f}_{e_i}$ is the face of $\hat{G}_{e_i}$ that contains e_i and encloses f_{e_i}. We claim that $\hat{f}_{e_i}$ is also a face of $\hat{G}_{e+}$. To prove the claim, note that $\hat{G}_{e_i}$ can be obtained from $\hat{G}_{e+}$ by deleting a subset of $\{e, e_1, \ldots, e_s\} - \{e_i\}$. None of these edges are edges of T or descendants in T^* of e_i, so, by Part 1 of the inductive hypothesis, none belongs to $\hat{f}_{e_i}$.

Note that $\hat{G}_e$ can be obtained from $\hat{G}_{e+}$ by deleting those edges among $e_1, \ldots, e_s$ that are rejected. By the claim, each such deletion replaces a rejected edge e_i in f_e with the walk $\hat{W}_{e_i}$. This together with the definition of P_{e_i} proves Part 2. By Part 1 of the inductive hypothesis, every edge in each $\hat{W}_{e_i}$ is an edge of T or a descendant of e_i in T and hence a descendant of e as well. This proves Part 1. □

Lemma 15.9.5. *In the for-loop iteration that considers e,*

- *the value assigned to x_{omit} is weight$(\hat{W}_e)$, and*
- *the value assigned to $x[e]$ is weight(P_e).*

Proof. The proof is by induction. By Lemma 15.9.3, the edges $e_1, \ldots, e_s$ are considered before e. By the inductive hypothesis, $x[e_i] = \text{weight}(P_e)$. By Lemma 15.9.4, $\text{weight}(\hat{W}_e) = \sum_{i=1}^{s} x[e_i]$, which proves the first statement. The second statement follows by definition of P_e. □

Corollary 15.9.6. *For each edge e, weight$(P_e) \leq (1+\epsilon)$weight(e).*

Proof. If e is accepted, $P_e = e$ so the statement holds trivially. Suppose e is rejected. By the conditional in the algorithm, in the iteration considering e, the value assigned to x_{omit} was at most $(1+\epsilon)$weight(e). By the first part of Lemma 15.9.5, weight$(\hat{W}_e)$ and therefore weight(P_e) are at most $(1+\epsilon)$weight(e). □

Corollary 15.9.7. *The graph of accepted edges satisfies Property A2.*

Proof. For any pair of vertices u and v, let P be the shortest u-to-v path in G_0. For each edge e of P,there is a walk P_e consisting of accepted edges between the endpoints of e. By Corollary 15.9.6, weight$(P_e) \leq (1+\epsilon)$weight(e). Replacing each edge e of P with P_e therefore yields a walk of weight at most $\sum_{e \in P}(1+\epsilon)$weight$(e)$, which is at most $(1+\epsilon)$weight(P). □

Lemma 15.9.8. *At any time during the algorithm's execution, the weight of the infinite face in the graph consisting of accepted edges is at most*

$$2 \cdot MST(G_0) - \epsilon \cdot \textit{weight}(\textit{accepted edges not in } T)$$

Proof. The proof is by induction. Before the for-loop commences, the graph of accepted edges is T, the minimum spanning tree of G_0. Hence the weight of the infinite face is exactly $2 \cdot MST(G_0)$, so the lemma's statement holds for this time. Consider a for-loop iteration, and let e be the edge being considered. If e is not accepted, there is no change to the set of accepted edges, so the lemma's statement continues to hold.

Suppose e is accepted. Let G_{after} be the subgraph consisting of edges accepted so far, and let $G_{\text{before}} = G_{\text{after}} - \{e\}$. Note that G_{after} can be obtained from $\hat{G}_e$ by deleting edges that will be accepted in the future. By the leaves-to-root ordering, none of the deleted edges are descendants of e in T^*. By Part 1 of Lemma 15.9.4, therefore, $\hat{f}_e$ is a face of G_{after}. Let g be the other face of G_{after} that contains e.

We claim that g is the infinite face of G_{after}. To prove the claim, note that G_{after} can be obtained from G_0 by deleting edges that have already been rejected and edges not yet considered. By the leaves-to-root ordering, e's proper ancestors in T^* have not yet been considered, so they are among the edges deleted. These deletions are contractions in the dual. The root of T^* is the infinite face, so the contractions result in g being the infinite face.

Note that G_{before} can be obtained from G_{after} by deleting e. This deletion replaces e in the face g with $\hat{W}_e$. This shows that

$$\begin{aligned}
&\text{weight of infinite face in } G_{\text{before}} - \text{weight of infinite face in } G_{\text{after}} \\
&= \text{weight}(\hat{W}_e) - \text{weight}(e) \\
&> (1+\epsilon)\text{weight}(e) - \text{weight}(e) \text{ because } e \text{ was accepted} \\
&= \epsilon \cdot \text{weight}(e)
\end{aligned}$$

which shows that the lemma's statement continues to hold. □

Corollary 15.9.9. *The graph G of accepted edges satisfies Property A1.*

Proof. By Lemma 15.9.8, the weight of the infinite face in the graph consisting of all accepted edges is at most

$$2 \cdot MST(G_0) - \epsilon \cdot \text{weight(accepted edges not in } T)$$

so weight(accepted edges not in T) $\leq 2\epsilon^{-1} \cdot MST(G_0)$. Since weight($T$) $= MST(G_0)$, it follows that the weight of all accepted edges is at most $(1 + 2\epsilon^{-1})MST(G_0)$. □

This completes the proof of Theorem 15.9.1.

15.10 TSP on bounded-branchwidth planar graphs

To be written.