Chapter 15

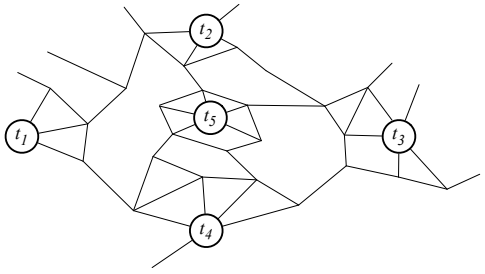
Approximation scheme for the traveling salesman problem

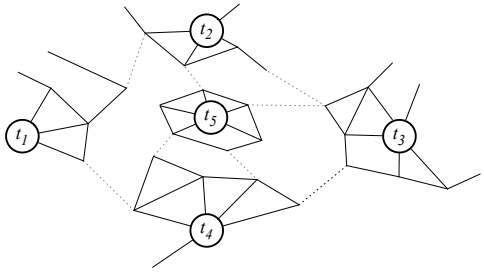
15.1 Multiterminal cut

In this chapter, we describe a methodology that can be used for some problems for which the previous chapter's methodology fails. To start with, we consider

> the *multiterminal cut problem*: given a graph G with edge-costs and given a set S of vertices (the *terminals*), find a minimum-cost set E of edges such that $G - E$ contains no S-to-S paths.

Here is an example. The top figure shows part of an instance and the bottom figure illustrates the corresponding part of the solution.



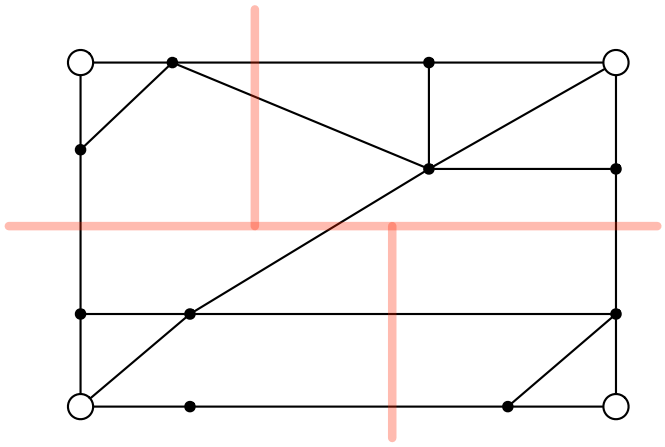


The problem is a natural generalization of minimum st-cut in undirected graphs. It is NP-hard, even in planar graphs. Is there an approximation scheme?

15.1.1 First attempt

Let's try to apply the methodology of Section 14.7. The good news is that the whole-to-parts property holds of multiterminal cut. The bad news is that a decomposition of a graph into slightly overlapping subgraphs does not ensure the parts-to-whole property.

In this diagram, the terminals are located at the four corners.



The graph is decomposed into four parts. Since each part contains only one terminal, the optimal multiterminal cut for that part is empty. The union of these empty solutions, of course, is not a multiterminal cut for the whole graph since it does not separate terminals that belong to distinct parts.

The failure of the parts-to-whole property is catastrophic for the approach to approximation schemes used in Chapter 14. We will need a somewhat different approach.

15.1.2 Deletion Decomposition

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

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

The approximation error for an algorithm of this form arises from the addition of edges between the subgraphs, so we want the total weight of these edges to be at most ϵ times the optimal value.

Drawing on our experience with approximation schemes in Chapter 14, we consider using the decomposition design of Lemma 14.7.5:

> Let G be a plane graph, let r be a vertex, let k be a positive integer, and let E_i be the set of edges of G whose levels are congruent mod k to i. Then $G - E_i$ has branchwidth at most $2k$.

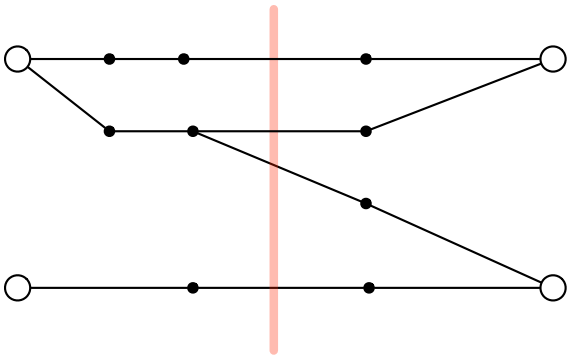
We abstract this procedure as follows.

Lemma 15.1.1 (Deletion-Decomposition Lemma)**.** *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 $i = 1, \ldots, k$, $G - E_i$ has branchwidth at most $2k$.*

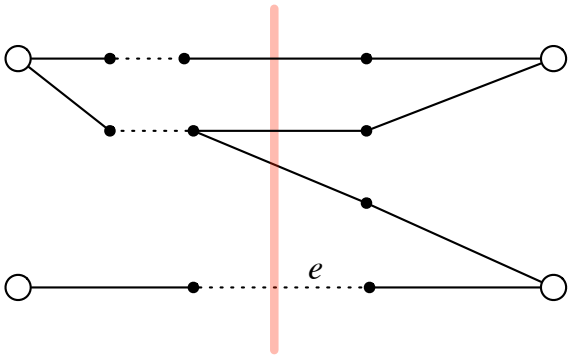
9

Which i should the multiterminal-cut algorithm use? We might think to choose i so as to minimize the weight of $OPT \cap S_i$, because this choice gives weight$(OPT \cap S_i) \leq \frac{1}{k}$ weight(OPT). However, this won't work: even if somehow the algorithm knew which edges of S_i belonged to OPT, including those edges in the solution in Step 3 would likely not lead to a multiterminal cut because the multiterminal cuts obtained in Step 2 for each of the subgraphs are not likely to match up with the edges of $OPT \cap S_i$ to form a multiterminal cut for G.

Consider the graph G separated into two subgraphs by removal of the set S of edges intersected by the vertical line:



In each of the two subgraphs, the optimal multiterminal cut is the empty set. However, suppose the optimal solution consists of the dotted lines in the following diagram:



Only the edge labeled e is in $OPT \cap S$, but adding e to the (empty) solutions to the subgraph instances does not yield a solution to the original instance.

It seems that, to be safe, *all* the edges in S must be added in Step 3, regardless of OPT. Returning to the decomposition design, we should choose i to minimize weight(S_i). This ensures only that weight$(S_i) \leq \frac{1}{k}$weight$(E(G))$. Thus the approximation error introduced in Step 3 is a small fraction of weight(OPT) only if the weight of the entire input graph is not much greater than weight(OPT).

Suppose therefore that weight$(E(G)) \leq g(\epsilon)$ weight(OPT) for some function $g(\cdot)$ of ϵ. By choosing $k = g(\epsilon)\epsilon^{-1}$, we can ensure that the approximation error introduced in Step 3 is at most ϵ weight(OPT).

15.1.3 Reducing the weight

How can we justify the assumption that weight$(E(G)) \leq g(\epsilon)$ weight(OPT)? For some unit-weight problems, as we will see, this inequality is a consequence of sparsity. For most problems, and for arbitrary weights, we cannot assume this is true of the input graph—we must make it true by reducing the weight of the input graph.

Let G_0 denote the input graph. One way of deriving a low-weight graph G from G_0 is by *deleting* a set A of heavy edges. However, for multiterminal cut this is not a useful strategy, for a multiterminal cut in $G_0 - A$ would not be a multiterminal cut in G_0. Instead, consider *contracting* a set A of heavy edges. This has the advantage that a multiterminal cut in G_0/A is a multiterminal cut in G_0.

Of course, contracting edges can have the effect of increasing the minimum weight of a multiterminal cut because it rules out some solutions. For this approach to work, we would need to ensure that the weight does not increase too much. Thus we need an algorithm to find a set A with two properties:

$$\text{weight}(G_0/A) \leq g(\epsilon)\,\text{weight}(\text{OPT}(G_0)) \tag{15.1}$$

$$\text{weight}(\text{OPT}(G_0/A)) \leq (1+\epsilon)\text{weight}(\text{OPT}(G_0)) \tag{15.2}$$

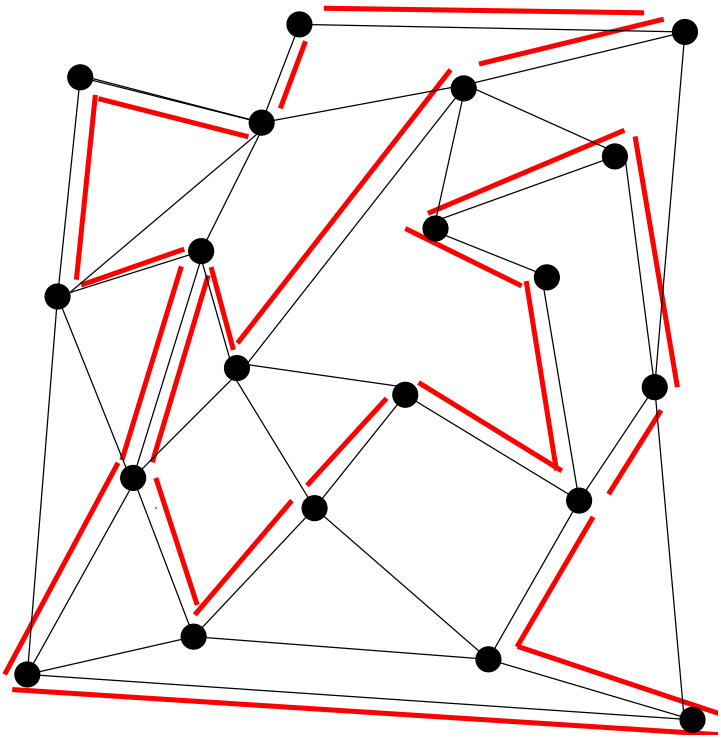
where OPT(G) denotes a minimum-weight multiterminal cut in G. Does such a set A of edges even exist? Yes; take A to be the set of edges not in the minimum-weight multiterminal cut. Then the two inequalities are trivially satisfied.

We refer to the graph G_0/A as a *spanner* for the multiterminal cut instance. Can it be found in polynomial time? The answer is yes. Though the details are beyond the scope of this text, the algorithm builds on techniques we will describe later.

15.2 The traveling salesman problem

Now we turn to the *traveling salesman problem* (TSP), for which we give an approximation scheme that builds on the technique described in the previous section and adds a twist on the technique.

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.

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.

Lemma 15.2.1 (Euler's Lemma)**.** *1. For any graph G and tour W, 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.*

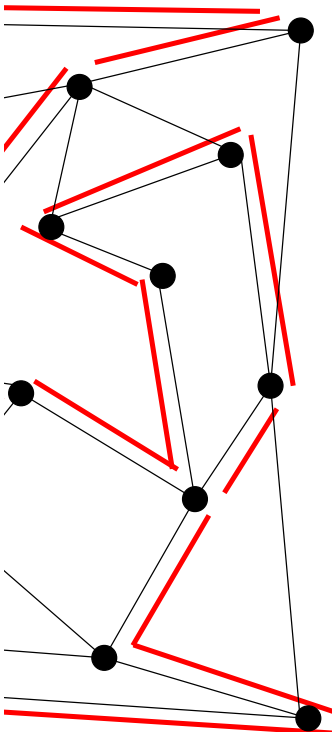
provide proof? NoneResponsible NoStatus

It follows from Euler's 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 multisubgraph M with the following properties:

- every vertex of S is the endpoint of at least one edge in M,
- the edges of M are connected, and
- every degree is even.

15.3 Approximating TSP

The approximation-scheme methodology described in Chapter 14 fails on TSP because traveling salesman tours do not satisfy the whole-to parts property. As shown below, the part of a tour that lies in a subgraph is not itself a tour:



Suppose we were to decompose the input graph into subgraphs and find the

shortest tour in each subgraph. The sum of weights of those tours could be far more than the weight of the shortest tour in the input graph. Indeed, some subgraph might be disconnected, in which case it would have *no* tour.

A subgraph of G is essentially a graph obtained from G by deletions. An alternative that seems more useful for TSP is to obtain a graph from G by *contractions*. Note that TSP satisfies the following modified version of whole-to-parts:

> *Contraction whole-to-parts property:* Let G be a graph and let H be a contraction of G. For any solution S for G, the subset of S that belongs to H is a solution for H.

This suggests an approach to finding an approximately minimum-weight tour in a planar graph G:

1. Find disjoint sets $E_1, \ldots, E_k$ such that, for any i, G/E_i has branchwidth at most $2k$.
2. For each i, find an optimal solution S_i in G/E_i.
3. **Now what?**

We show in Section 15.4 that Step 1 can be carried out in linear time. Since the resulting graphs $G/E_1, \ldots, G/E_k$ have bounded branchwidth, an optimal solution can be found in each of these graphs in linear time. What about Step 3?

A solution S_i for G/E_i is not a solution for G. The problem is that TSP does not have the parts-to-whole property. We faced a similar problem in considering multiterminal cut; in that case, we combined the solutions for the pieces and also included in the solution the edges E_i that had been removed. In the case of TSP, we will show that a solution can be obtained by augmenting S_i with edges chosen from among E_i. We show that the augmentation uses each edge of E_i at most twice. As a consequence, the increase in weight due to Step 3 is at most twice the weight of E_i.

We refer to the augmentation as *lifting* since it involves transforming a solution for the contracted graph into one for the uncontracted graph. We discuss lifting in Section 15.7.

How can we ensure that the increase in weight due to lifting is a small fraction of the optimal value? As in Section 15.1.2, the algorithm should choose i from among $1, \ldots, k$ to minimize weight(E_i). As a consequence,

$$\text{weight}(E_i) \leq \frac{1}{k}(\mathit{weight}(E_1) + \cdots + \text{weight}(E_k)) \leq \frac{1}{k}\,\text{weight}(E(G))$$

Thus the increase in weight due to lifting is bounded by $\frac{2}{k}\,\text{weight}(E(G))$. To ensure that the increase is a small fraction of the weight of the optimal tour, we take an approach similar to that Section 15.1.3. Starting with the input graph G_0, the algorithm needs to get rid of enough edges that the weight of the

remaining set A is at most c times the weight of the optimal tour, where c is a constant.

How to get rid of edges? We want to ensure that a tour for the trimmed-down graph is also a tour for the original input graph G_0, so we get rid of edges by deleting them. At the same time, we want to ensure that the optimal tour in the trimmed-down graph is not much heavier than that in the original input graph. Summarizing the requirements, we seek a set A of edges to delete such that

$$\begin{aligned} \text{weight}(G_0 - A) &\leq g(\epsilon)\,\text{weight}(\text{OPT}(G_0)) && (15.3)\\ \text{weight}(\text{OPT}(G_0 - A)) &\leq (1+\epsilon)\text{weight}(\text{OPT}(G_0)) && (15.4) \end{aligned}$$

where $\text{OPT}(G)$ denotes a minimum-weight tour in G. We refer to $G_0 - A$ as a *spanner* for the given instance of TSP. In Section 15.8 we describe a linear-time algorithm for constructing a spanner.

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 15.3.

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

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

15.4 Compression decomposition and contraction decomposition

We consider how to find the sets of Step 1.

Lemma 15.4.1 (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 15.1, 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 15.4.2 (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 15.4.1, 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 14.5.5, therefore, G/A_i has branchwidth at most $2k + 1$. □

15.5 The framework

The approach described in Section 15.3 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. [10]

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 15.3, 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 15.8 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 $2^{O(w \log w)}n$.

15.5.1 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.

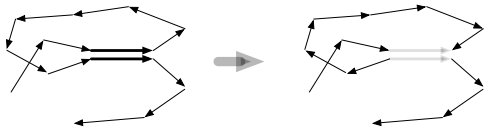
In the dual form of the framework, the spanner step *contracts* edges, whereas the thinning step uses the Deletion-Decomposition Lemma to *delete* edges.

15.6 Properties of tours

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

Lemma 15.6.1. *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 $W_1\ \mathrm{rev}(W_2)$ 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 15.6.1 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 15.6.2. *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.*

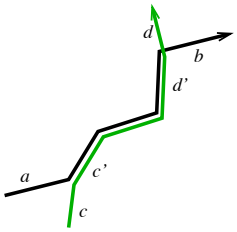


Figure 15.1: The light walk forms a crossing configuration with the bold walk.

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 $d\ W_1\ a\ \text{rev}(c\ W_2\ b)\ \text{rev}(W)\ W$ is a closed walk visiting the same nodes and comprising the same darts in the same multiplicities, and with one fewer crossing configurations. □

Proposition 15.6.2 shows that we can restrict our attention to non-self-crossing walks.

15.7 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 Section15.6, 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 15.2.) 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 ∪ {e}
  return B ∪ {e} ∪ {e}

def LiftMany(G, S, B):
  if S = ∅
  return B
  let e be an edge of S
    return LiftOne(G, e, LiftMany(G/{e}, S − {e}, B))
```

Lemma 15.7.1 (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. □

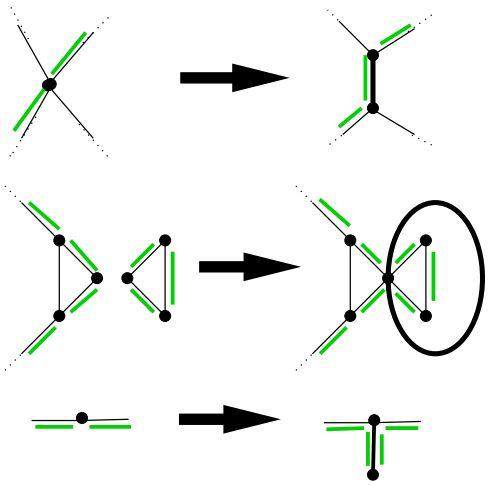


Figure 15.2: Examples of lifting. The Eulerian bisubgraph is indicated with light lines. In the top diagram, only one copy of the formerly compressed edge (the bold line) must be incorporated. In the second diagram, no copies are needed. In the third diagram, two copies are needed.

15.8 Spanner

Theorem 15.8.1. *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*

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,*

$$\begin{aligned} &\textit{minimum weight of a } u\textit{-to-}v \textit{ path in } G \\ \leq\ &(1+\epsilon) \cdot \textit{minimum weight of a } u\textit{-to-}v \textit{ path in } G_0 \end{aligned} \tag{15.5}$$

Lemma 15.8.2. *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 $(1 + \epsilon)\text{weight}(uv)$; 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 15.8.3 is as follows.

```
define SPANNER(G_0, ε):
    let x[·] 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, ..., e_s be the sequence of edges comprising f_e
      x_omit := Σ_{i=1}^{s} x[e_i]
      if x_omit > (1 + ε)weight(e)
        then add e to S and assign x[e] := weight(e)
        else assign x[e] := x_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.

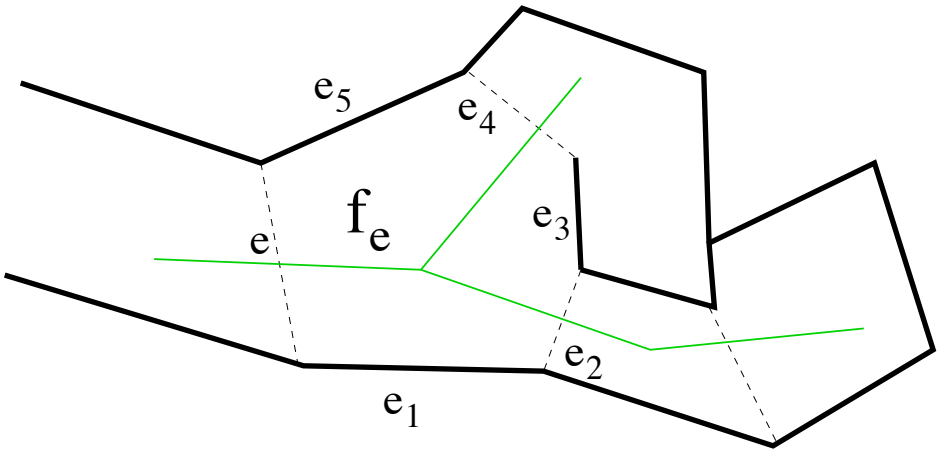


Figure 15.3: Diagram showing part of dual tree (in light edges) and primal tree (in dark edges) and primal nontree edges (dashed): e_2 and e_4 are child edges of e in the dual tree. The face f_e is indicated.

Lemma 15.8.3. *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^*. □

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 15.8.4. *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 15.8.5. *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 15.8.4, the edges $e_1, \ldots, e_s$ are considered before e. By the inductive hypothesis, $x[e_i] = \text{weight}(P_e)$. By Lemma 15.8.5, $\text{weight}(\hat{W}_e) = \sum_{i=1}^{s} x[e_i]$, which proves the first statement. The second statement follows by definition of P_e. □

Corollary 15.8.6. *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 $(1+\epsilon)\text{weight}(e)$. By the first part of Lemma 15.8.6, $\text{weight}(\hat{W}_e)$ and therefore $\text{weight}(P_e)$ are at most $(1+\epsilon)\text{weight}(e)$. □

Corollary 15.8.7. *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 15.8.7, $\text{weight}(P_e) \leq (1+\epsilon)\text{weight}(e)$. Replacing each edge e of P with P_e therefore yields a walk of weight at most $\sum_{e \in P}(1+\epsilon)\text{weight}(e)$, which is at most $(1+\epsilon)\text{weight}(P)$. □

Lemma 15.8.8. *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 15.8.5, 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 15.8.9. *The graph G of accepted edges satisfies Property A1.*

Proof. By Lemma 15.8.9, the weight of the infinite face in the graph consisting of all accepted edges is at most

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

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 15.8.3.

15.9 TSP on bounded-branchwidth planar graphs

To be written.