Chapter 14

Branchwidth and local approximation schemes

14.1 Dynamic programming on a rooted tree

Some problems are easy to solve on trees. Our example problem will be Vertex Cover, which is NP-hard on general graphs and even on planar graphs. (Soon we'll see how to approximate the problem on planar graphs.)

Vertex Cover *on Rooted Trees:*

- input: a rooted tree with a function $c(\cdot)$ assigning weights to vertices
- output: a minimum-weight set S of vertices such that each edge is incident to at least one vertex in S.

Imagine you're posting guards at intersections so that each city block is handled by a guard at one of the intersections. (Perhaps the different weights reflect differences in popularity of the intersections, e.g. one intersection might be near a café.

Next we give a linear-time algorithm that, for a tree, returns the minimum weight of a vertex cover. It is easy to modify the algorithm so that it also returns a minimum-weight vertex cover.

```
def VertexCover(T, r):
    let (x, y) := helper(r)
    return x

def helper(v):
    let v_1, ..., v_k be the children of v in T
    for i = 1, ..., k
        let (x_i, y_i) := helper(v_i).
    y := weight(v) + sum_{i=1}^{k} x_i
```

$x := \min\{y, \sum_{i=1}^{k} y_i\}$
return (x, y).

Lemma 14.1.1. *VertexCover(T) returns the minimum weight of a vertex cover of the tree T.*

Proof. We will show by induction that helper(v) takes a node v and returns the pair (x, y) where x is the minimum weight of a vertex cover of the subtree of T rooted at v, and y is the minimum weight of a vertex cover of the same subtree, subject to the constraint that v belong to the vertex cover. Certainly this is true when v is a leaf. Suppose it is true for the children $v_1, \ldots, v_k$ of node v. Let T_v be the subtree of T rooted at v. Let C_y be the vertex cover of T_v forced to include v. The arcs $vv_1, \ldots, vv_k$ are covered by v and the arcs of $T_{v_1}, \ldots, T_{v_k}$ are covered by the optimal vertex covers for these subgraphs whose values are $x_1, \ldots, x_k$. So weight(C_y) = weight$(v) + \sum_{i=1}^{k} x_i$. The optimal vertex cover for T_v either contains v (and so has value y) or doesn't. If it doesn't, it must contain all of $v_1, \ldots, v_k$ to cover the arcs $vv_1, \ldots, vv_k$, in which case, the cover has weight $\sum_{i=1}^{k} y_i$. □

The above algorithm is an application of dynamic programming. It is so slick we are tempted to try to apply the idea to graphs that are not trees.

14.2 Carvings

For a ground set S, a *carving* of S is a maximal noncrossing family $\mathcal{C}$ of subsets of S. By *maximal*, we mean that you cannot add any other subset of S to $\mathcal{C}$ while preserving the noncrossing property. We refer to the sets in $\mathcal{C}$ as *clusters*.

Recall from Section 5.6 that a noncrossing family of sets forms a rooted forest under the subset relation. The forest corresponding to a carving $\mathcal{C}$ is a rooted binary tree, i.e. it has a single root and each node has at most two children. To see that it is binary, suppose that some node X has children X_1, X_2, and X_3. The set $X_1 \cup X_2$ does not cross any descendant of X_1 or X_2 (it includes each of these sets) or X_3 (it is disjoint from this set) or any ancestor of X (it is a subset of each of these sets) or any node that is neither an ancestor nor a descendant of X (is disjoint from these sets). Therefore $\mathcal{C} \cup \{X_1 \cup X_2\}$ is also noncrossing, contradicting the maximality of $\mathcal{C}$.

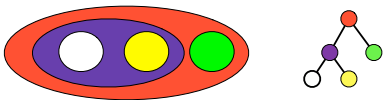


Figure 14.1: Copy of Figure 5.3. The diagram on the left shows a Venn diagram of some noncrossing sets, and the diagram on the right shows the corresponding rooted forest (a tree in this case).

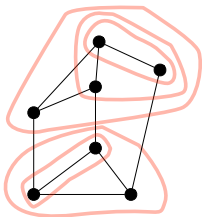


Figure 14.2: This figure shows a carving-decomposition of a graph. Clusters other than the trivial ones (singletons and whole graph) are indicated. The width is four.

14.3 Carving-decomposition: carving of a vertex set

Let G be a graph. A carving $\mathcal{C}$ of $V(G)$ is called a *carving-decomposition*. The *width* of a carving-decomposition is $\max\{|\delta_G(X)| \ : \ X \in \mathcal{C}\}$. Recall that $\delta_G(X)$ is the cut in G corresponding to X, i.e. the set of edges of G having exactly one endpoint in X. The *carvingwidth* of a graph is the minimum width over all carving-decompositions.

For each vertex v of G, the carving contains the singleton set $\{v\}$. The corresponding cut $\delta_G(\{v\})$ includes all the edges incident to v (not including self-loops). Thus the width is at least the maximum degree of the graph. This somewhat limits the usefulness of carvingwidth.

The following lemma is obvious but useful.

Lemma 14.3.1. *Deleting edges or vertices does not increase the carvingwidth of a graph.*

14.3.1 Solving edge dominating set on a graph with a carving-decomposition of small width

An *edge dominating set* of a graph is a set S of edges such that each edge in the graph shares a vertex with some edge in S.

Theorem 14.3.2. *There is an algorithm that, given a graph G with edge-weights and a carving-decomposition $\mathcal{C}$ of G, finds the minimum-weight edge dominating set of G in time $O(2^w n)$ where w is the width of $\mathcal{C}$.*

The algorithm uses dynamic programming. For each cluster $X \in \mathcal{C}$, the algorithm constructs a table $M_X[\cdot]$ indexed by the subsets of $\delta_G(X)$. For such a subset A, $M_X[A]$ is the minimum-weight of a subset of edges that covers every edge in $G[X - \partial_G(A)]$, i.e. every edge whose endpoints are in X and are not endpoints of edges of A. Recall that the clusters of $\mathcal{C}$ form a tree according to set inclusion. The dynamic program uses rootward computation on this tree. At the root, the table $M_{V(G)}[\cdot]$ is computed, and $M_{V(G)}[\emptyset]$ is the minimum weight of

an edge dominating set. Below we give the dynamic program to compute these tables and thus the minimum weight of a dominating set; the dynamic program can be instrumented to facilitate subsequent computation of the dominating set itself.

```
def EdgeDomTable(X):
  if X is a leaf,
    M_X[A] := 0 for each A ⊆ δ(X)
  else
    let X_1, X_2 be the children of X
    for i = 1, 2, M_{X_i} := EdgeDomTable(X_i)
    for every subset A of δ(X):
      for every subset A' of edges between X_1 and X_2,
        if every edge between X_1 and X_2 shares an endpoint with an edge in A ∪ A',
          M_X[A] := min{M_X[A], weight(A') + Σ_{i=1}^{2} M_{X_i}[(A ∪ A') ∩ δ(X_i)]}
  return M_X[·]
```

14.4 Carving-decomposition of a planar graph

The following bound on carvingwidth uses the same idea as Fundamental Cycle Separator (Lemma 5.3.1 in Section 5.3).

Lemma 14.4.1. *Let G be a planar embedded graph of degree at most Δ. Let T^* be a spanning tree of the dual G^*, and suppose that every simple path in T^* consists of at most k edges. Then G has a carving-decomposition of width at most $k + \Delta - 1$.*

Proof. Let T be the set of edges not in T^*. Then T is a spanning tree of G. Let r be a vertex with one incident edge of T. Consider T as rooted at r. We define a carving of $V(G)$.

For each vertex v of G, let $C(v)$ denote the set of descendants of v. First consider the case $v = r$. Note that $C(r)$ consists of all the vertices of G, so $\delta_G(C(r)) = \emptyset$. Now suppose $v \neq r$, and let d be the parent dart of v. In this case, $\delta_G(C(v))$ is the fundamental cut of d with respect to T, and is therefore the fundamental cycle of d with respect to T^*. The fundamental cycle of d consists of d itself together with a simple path in T^*, and therefore consists of at most $k + 1$ edges.

The family $\{C(v) : v \in V(G)\}$ is not a carving because it is not maximal, so we must add some additional clusters.

Let v be a vertex, and let its children be $v_1, \ldots, v_s$. For $i = 0, 1, 2, \ldots, s-1$, define $C_i(v) = \{v\} \cup \bigcup_{j=1}^{i} C(v_j)$. The proof that $|\delta_G(C_i(v))| \leq k + \Delta - 1$ is similar to the proof that $|\delta_G(C(v)| \leq k + 1$ and is illustrated in Figure 14.3.

To finish the proof, the following properties can be shown.

1. The family of clusters $\mathcal{C} = \{C(v) : v \text{ with at least one child}\} \cup \{C_i(v) : v \in V(G), i \leq \text{number of children of } v\}$ is noncrossing.

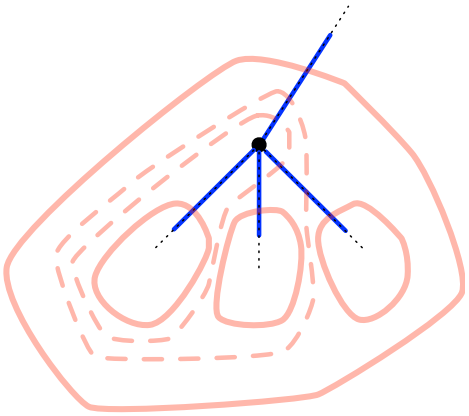


Figure 14.3: This figure illustrates the construction of a carving-decomposition for a planar graph with a spanning tree. The solid closed curves are fundamental cycles in the dual. Since each simple path in the dual spanning tree has size at most k, the fundamental cycles have size at most $k+1$. For a complete carving-decomposition, the dashed closed curves are needed. Each of these consists of a simple path in the dual spanning tree, plus at most $\Delta - 1$ additional edges that in the primal are incident to the vertex v.

2. The number of clusters in $\mathcal{C}$ shows that it is maximal.

□

The width bound in Lemma 14.4.1 can be improved for a special case.

Lemma 14.4.2. *Let G be a planar embedded graph of degree at most four. Let T^* be a spanning tree of the dual G^*, and let f be a vertex of G^*. Suppose that every simple from-f path in T^* has at most r edges, and that G^* is bipartite. Then G has a carving decomposition of width at most $2r$.*

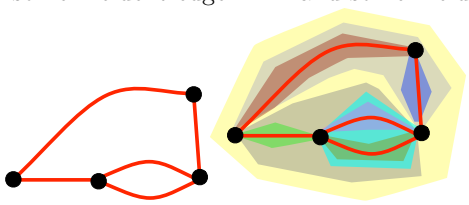
Problem 14.1. *Prove Lemma 14.4.2.*

The disadvantage of carvingwidth as a measure is that it cannot be very small if the graph's maximum degree is large. The next measure we study can be small even if the degree is large.

14.5 Branch decomposition: Carving of an edge-set

Let G be a graph. A carving $\mathcal{C}$ of $E(G)$ is called a *branch-decomposition*. The *width* of a branch-decomposition is $\max\{|\partial_G(X)| \ : \ X \in \mathcal{C}\}$, where $\partial_G(X)$ is the

set of vertices with some incident edge in X and some incident edge not in X.



The following lemma is analogous to Lemma 14.3.1 but stronger.

Lemma 14.5.1. *Deleting or contracting edges does not increase the branchwidth of a graph.*

14.5.1 Solving vertex cover on a graph with a branch decomposition of small width

Theorem 14.5.2. *There is an algorithm that, given a graph G with vertex-weights and a branch-decomposition $\mathcal{C}$ of G, finds the minimum-weight vertex cover of G in time $O(4^w n)$ where w is the width of $\mathcal{C}$.*

The algorithm resembles that given for edge dominating set in a graph with a carving-decomposition. For each edge-subset $X \in \mathcal{C}$, the algorithm constructs a table $M_X[\cdot]$ indexed by the subsets of $\partial_G(X)$. For such a subset A, $M_X[A]$ is the minimum-weight of a subset of vertices that covers all edges in X that are not incident to vertices in A. Since $\partial_G(E(G)) = \emptyset$, $M_{E(G)}[\emptyset]$ is the minimum-weight of a vertex cover.

Problem 14.2. *Prove Theorem 14.5.2 by giving the algorithm for minimum-weight vertex cover.*

So far, we have seen dynamic-programming algorithms only for minimization problems. Now we consider a maximization problem, *maximum-weight independent set*. An *independent set* in a graph G is a set S of vertices such that $G[S]$ has no edges. Given a graph with vertex-weights,

Problem 14.3. *Give an algorithm that, given a graph with vertex-weights and a branch-decomposition of width w, finds a maximum-weight independent set. The running time of the algorithm should be $2^{O(w)}n$.*

14.5.2 Biconnectivity and the block-cut tree

Recall from Section 4.10 that a graph is biconnected if every pair of edges belong to some simple cycle. A *biconnected component* (or *block*) of a graph is a maximal biconnected subgraph.

Two biconnected components may share a vertex, in which case the vertex is called an *articulation point* (or a *cutpoint*). Note that two biconnected components cannot share two vertices.

The *block-cutpoint tree* of a connected graph G is a bipartite graph. It has a vertex for each block of G and a vertex for each cutpoint; for each block and

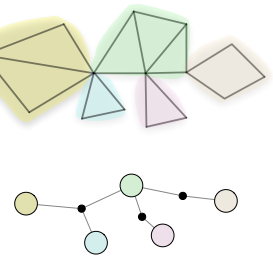


Figure 14.4: The top diagram shows a graph with its biconnected components. The bottom diagram shows the block-cutpoint tree.

cutpoint in that block, there is an edge between the corresponding vertices. A cycle in this graph would imply the existence of a simple cycle containing edges from distinct blocks, contradicting the definition of block.

14.5.3 Biconnected components and branchwidth

Lemma 14.5.3. *Let G be a graph. Suppose that every biconnected component of G has branchwidth at most w. Then G has branchwidth at most $w + 1$.*

Problem 14.4. *Use the block-cut tree of G to prove Lemma 14.5.3.*

14.6 A branchwidth bound for planar graphs

For a graph and a vertex r, we define the *radius* of G with respect to r to be the maximum vertex depth in a breadth-first-search tree rooted at r. That is, the radius is the maximum over vertices v of the minimum size of an r-to-v path.

Lemma 14.6.1. *There is a linear-time algorithm that, given a planar embedded graph G, returns a branch-decomposition whose width is at most*

$$2\min\{\textit{radius of } G, \textit{radius of } G^*\}$$

where the radii are with respect to given vertices.

Lemma 14.6.1 follows from a stronger result. To prove the stronger result, we derive two other plane graphs from G, the *face-vertex incidence graph* $FV(G)$ and its dual, the *medial graph* $M(G)$. We show how to obtain a carving-decomposition of the medial graph, and then show how to transform it to a branch-decomposition of G.

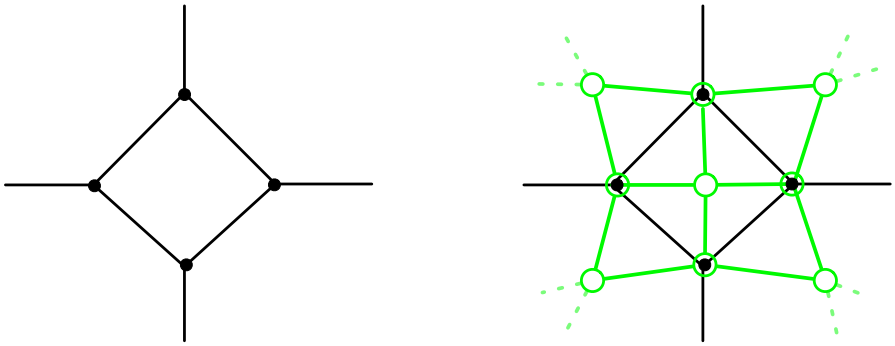


Figure 14.5: On the left is a fragment of an embedded graph. On the right is the same fragment, with part of the face-incidence graph superimposed.

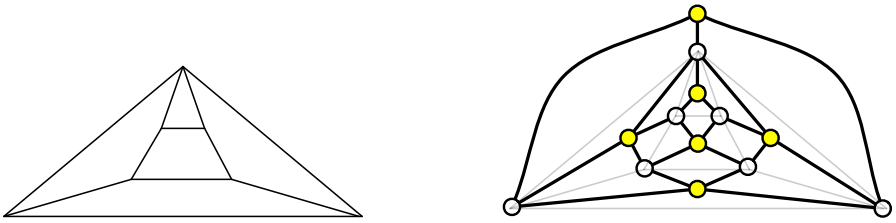


Figure 14.6: On the left is a planar embedded graph. On the right is the same graph with the face-incidence graph superimposed. The *face* vertices are filled in.

14.6.1 The face-vertex incidence graph

Let G be an embedded graph. A graph is a *face-vertex incidence graph* of G if its vertex set corresponds one-to-one with the union of the vertex set of G and the face set of G, and if, for a vertex v and a face f of G, the vertices corresponding to v and f are adjacent in $FV(G)$ if v occurs on the boundary of f.

Informally, we will refer to *the* face-vertex incidence graph of G since up to isomorphism there is just one.

Theorem 14.6.2 (Radius-Branchwidth Theorem)**.** *There is a linear-time algorithm that, given a plane graph G that is not a tree and a vertex f of the face-incidence graph of G, finds a branch-decomposition of G whose width is at most the radius of the face-vertex incidence graph with respect to f.*

A path in G or a path in G^* corresponds to a path in the face-incidence graph at most twice the size. This shows that the Radius-Branchwidth Theorem implies Lemma 14.6.1.

In proving the Radius-Branchwidth-Theorem, we assume G is connected,

else the radius of the face-vertex incidence graph is infinite. Since G is not a tree, it has at least two faces.

14.6.2 The embedded face-vertex incidence graph

For brevity, we refer to the face-vertex incidence graph as the FV graph. From examining a diagram of the FV graph, the following properties are intuitively obvious:

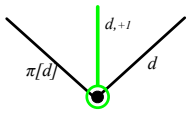
FV Property 1 The FV graph inherits a planar embedding from the original graph,

FV Property 2 Each face of the FV graph has size four.

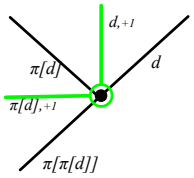
FV Property 3 The faces of the FV graph correspond one-to-one with edges of the original graph (each original edge is embedded in one face of the face-vertex incidence graph).

We will formally define the face-vertex graph of G as an embedded graph $FV(G)$. This will enable us to prove the FV properties. Moreover, the formal embedding will be useful to anyone who wishes to implement the construction of the face-vertex incidence graph. The embedding for $FV(G)$ will be denoted $\hat{\pi}$.

The figures suggest that, for each vertex v of G, the edges incident to v in $FV(G)$ should be interspersed between the edges incident to v in G itself. That is, for each consecutive pair $d, \pi[d]$ of outgoing darts of v in G, we shall place a dart of $FV(G)$.



We shall designate this dart as $(d, +1)$. The next dart in the embedding cycle is between $\pi[d]$ and $\pi[\pi[d]]$, so it must be $(\pi[d], +1)$



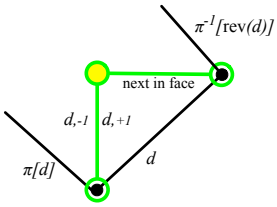
and so we shall define

$$\hat{\pi}[(d, +1)] := (\pi[d], +1) \tag{14.1}$$

The dart $(d, +1)$ belongs to some face of $FV(G)$, and the next dart in the face is

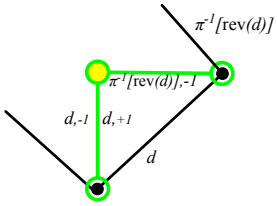
$$\begin{aligned} (\hat{\pi})^*[(d, +1)] &= \hat{\pi}[\mathrm{rev}((d, +1))] \\ &= \hat{\pi}[(d, -1)] \end{aligned}$$

How then should we define $\hat{\pi}[(d,-1)]$? The figure



indicates that the next dart in the face is the reverse of the dart between $\pi^{-1}[\text{rev}(d)]$ and $\text{rev}(d)$. According to the convention we have established, the dart of $FV(G)$ between $\pi^{-1}[\text{rev}(d)]$ and $\text{rev}(d)$ is $(\pi^{-1}[\text{rev}(d)],+1)$. We must therefore define

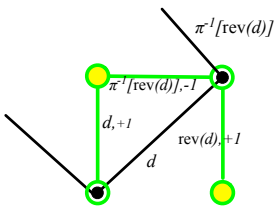
$$\hat{\pi}[(d,-1)] := (\pi^{-1}[\text{rev}(d)],-1) \tag{14.2}$$



We now have defined the image under $\hat{\pi}$ of darts of the form $(d,+1)$ and of darts of the form $(d,-1)$, so we have completely defined $\hat{\pi}$, and therefore completely defined the embedded graph $FV(G)$.

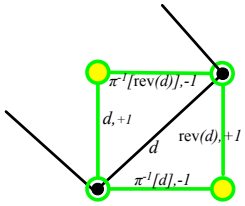
Let's continue to explore the face of $FV(G)$ containing $(d,+1)$. For convenience, let d' denote $\pi^{-1}[\text{rev}(d)]$. The next dart in the face after $(d',-1)$ is $\hat{\pi}[(d',+1)]$, which according to 14.1, is

$$\begin{aligned}(\pi[d'],+1) &= (\pi[\pi^{-1}[\text{rev}(d)]],+1)\\ &= (\text{rev}(d),+1)\end{aligned}$$



The next dart in the face is

$$\begin{aligned}\hat{\pi}^*[(\text{rev}(d),+1)] &= \hat{\pi}[(\text{rev}(d),-1)]\\ &= (\pi^{-1}[\text{rev}(\text{rev}(d))],-1) \text{ by 14.2}\\ &= (\pi^{-1}[d],-1)\end{aligned}$$



To find the next dart in the face, we again reverse this dart, obtaining $(\pi^{-1}[d], +1)$, and apply $\hat{\pi}$ using 14.1, obtaining

$$\begin{aligned} \hat{\pi}[(\pi^{-1}[d], +1)] &= (\pi[\pi^{-1}[d]], +1) \\ &= (d, +1) \end{aligned}$$

so we are back where we started, having traversed the whole face. The face consists of four darts, $(d, +1), (\pi^{-1}[\text{rev}(d)], -1)$, $(\text{rev}(d), +1)$, and $(\pi^{-1}[d], -1)$, so we have proved FV Property 2. Next, define the function $g(\cdot)$ on edges of G by mapping e to this cycle of darts:

$$g(e) = \{(d, +1)\ (\pi^{-1}[\text{rev}(d)], -1)\ (\text{rev}(d), +1)\ (\pi^{-1}[d], -1)\} \text{ where } d = (e, +1) \tag{14.3}$$

Note that the darts of e occur only in pairs with $+1$. This shows that distinct edges e, e' map to disjoint faces of $FV(G)$. The number of such faces is $|E(G)|$, so the number of darts of $FV(G)$ in all such faces is $4|E(G)|$, which is precisely the number of darts of $FV(G)$. This shows that every dart of $FV(G)$ is in one such face, so these are the *only* faces of $FV(G)$. We have proved FV Property 3.

Finally, we prove FV Property 1, that the embedding is planar. Assume for simplicity that G is connected. Let n, m, ϕ be the number of vertices, edges, and faces of G. Since G is planar, $n - m + \phi = 2$. Let n', m', ϕ' be the number of vertices, edges, and faces of $FV(G)$. By construction, $n = n + \phi$ and $m' = 2m$. We have just seen that $\phi' = m$. Therefore

$$\begin{aligned} n' - m' + \phi' &= n + \pi - 2m + m \\ &= n - m + \pi \\ &= 2 \end{aligned}$$

so $FV(G)$ is planar. We have proved FV Property 1.

14.6.3 The dual of the face-incidence graph

The dual of the face-incidence graph is called the *medial graph* of G, and is denoted $M(G)$. Each vertex of $M(G)$ is a face f of $FV(G)$, and $g^{-1}(f)$ is an edge of G, where $g(\cdot)$ is the bijection defined in 14.6.2.

When we draw $M(G)$ on top of a drawing of G, we place each vertex x of $M(G)$ in the middle of the corresponding edge $g^{-1}(x)$ G, as shown in Figure 14.7. For a set X of vertices of $M(G)$, let us use $g^{-1}(X)$ to denote $\{g^{-1}(x) : x \in X\}$, i.e. the corresponding set of edges of G.

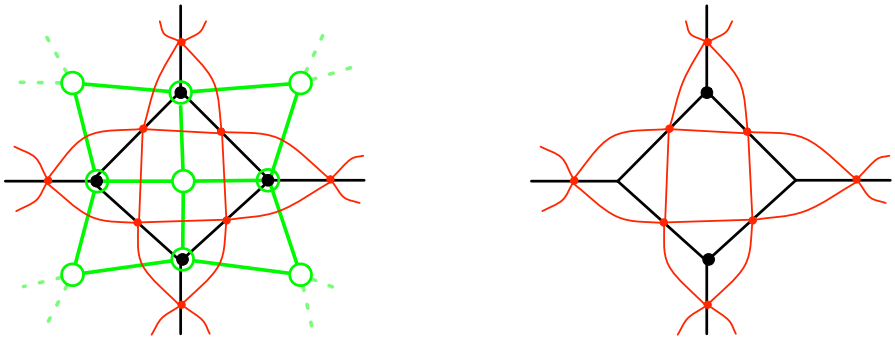


Figure 14.7: The drawing on the left shows the medial graph superimposed on the face-incidence graph and the original graph. The figure on the right shows the medial graph with just the original graph.

14.6.4 From a carving-decomposition of $M(G)$ to a branch-decomposition of G

Since $FV(G)$ is a bipartite planar graph of degree at most four, Lemma 14.4.2 proves that it has a carving-decomposition $\mathcal{C}$ of width at most twice the radius of $FV(G)$. We will show that $\mathcal{C}$ can be transformed into a *branch*-decomposition $\mathcal{C}'$ of G having at most half the width of $\mathcal{C}$.

Each cluster X of $\mathcal{C}$ is a set of vertices of $M(G)$, so $g^{-1}(X)$ is a set of *edges* of G. Define

$$\mathcal{C}' = \{g^{-1}(X) \,:\, X \in \mathcal{C}\} \tag{14.4}$$

The fact that $\mathcal{C}$ is a carving of $V(M(G))$ implies that $\mathcal{C}'$ is a carving of $E(G)$.

Lemma 14.6.3. *The width of $\mathcal{C}'$ is at most half the width of $\mathcal{C}$.*

Proof. The proof is illustrated in Figure 14.8. Let X be a cluster in $\mathcal{C}$, and let $Y = g^{-1}(X)$ be the corresponding cluster in $\mathcal{C}'$. We must show that $|\partial_G(Y)| \leq \frac{1}{2}|\delta_{M(G)}(X)|$. To do this, we show that each vertex $v \in \partial_G(Y)$ corresponds to at least two distinct edges $e_1, e_2 \in \delta_{M(G)}(X)$ such that e_1 and e_2 are darts of v. The latter condition ensures that there is no double-counting; for distinct vertices $u, v \in \partial_G(Y)$, the two edges corresponding to u are distinct from the two edges corresponding to v.

Since $v \in \partial_G(Y)$, $\delta_G(v)$ contains at least one edge in Y and at least one edge not in Y. Let $v = (d_0\ d_1\ d_2\ \ldots\ d_{k-1})$, and let $d_i, d_{i+1}, \ldots, d_j$ be a maximal consecutive subsequence of $d_0\ d_1\ d_2\ \ldots\ d_{k-1}$ consisting of darts of edges in Y. Then d_{i-1} and d_{j+1} (interpreting - and + as mod k) are darts of edges not in Y. (Possibly $d_{i-1} = d_{j+1}$.)

See Figure 14.9.

Let e_j and e_{j+1} be the edges of d_j and d_{j+1}, respectively. It follows from the definition of $g(\cdot)$ that $g(e_{j+1})$ contains $(d_{j+1}, -1)$ and that $g(e_j)$ contains

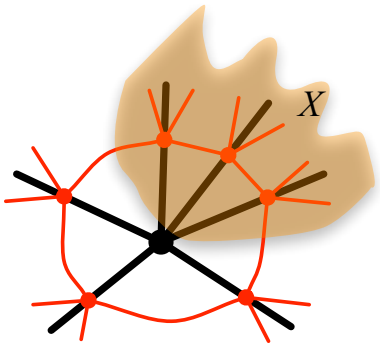


Figure 14.8: The thick dark edges belong to G. The thin light edges belong to $M(G)$. The shaded region represents a set X of vertices of $M(G)$, which corresponds to a set Y of edges of G. The dark circle is a vertex v of G. It is on the boundary of Y, which means that a nonempty proper subset of the edges incident to v belong to Y. The figure illustrates that, for each vertex v on the boundary of Y, at least two edges of $M(G)$ belong to $\delta_{M(G)}(X)$. Moreover, these two edges are both darts of v, so no edge is counted for two different vertices.

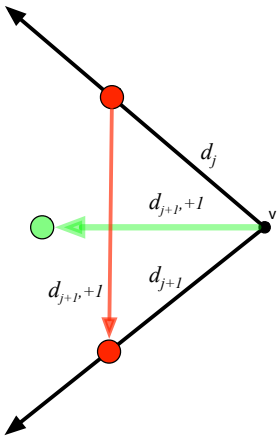


Figure 14.9: The black lines represent darts of G. The green line is a dart of $FV(G)$ pointing from a vertex of G to a face of G. The red line is a dart of $M(G)$.

$(\pi[d_j], +1)$, which is $(d_{j+1}, +1)$. This shows that $g(e_j)$ and $g(e_{j+1})$ are the tail and head, respectively, of $(d_{j+1}, +1)$ in $M(G)$, which shows that $d_{j+1} \in \delta_{M(G)}(X)$. The proof that $d_i \in \delta_{M(G)}(f(X))$ is similar. □

14.6.5 Proof of the Radius-Branchwidth Theorem

Now we can prove the Radius-Branchwidth Theorem (Theorem 14.6.2): there is a linear-time algorithm that, for a plane graph G and a vertex r of $FV(G)$, finds a branch-decomposition of G whose width is at most one plus the radius of $FV(G)$ with respect to r. The algorithm constructs $FV(G)$ and computes an r-rooted breadth-first-search tree T^* of $FV(G)$. By applying Lemma 14.4.2, the algorithm finds a carving-decomposition $\mathcal{C}$ of $M(G)$. The width of $\mathcal{C}$ is at most $k + \Delta - 4$ where k is the maximum number of edges in any simple path in T^* and Δ is the degree of $M(G)$. Let ρ be the radius of $FV(G)$ with respect to r. Since $k \leq 2\rho$ and $\Delta = 4$, the width of $\mathcal{C}$ is at most 2ρ. The algorithm then constructs a branch-decomposition $\mathcal{C}'$ of G as defined in 14.4. By Lemma 14.6.3, the width of $\mathcal{C}'$ is at most ρ.

14.7 Approximation schemes

Can we find a minimum-weight vertex cover in a planar graph? This problem turns out to be NP-hard. However, the problem does admit an *approximation scheme*. That is, for each $\epsilon > 0$, there is a polynomial-time algorithm that finds a solution whose weight is at most $1 + \epsilon$ times optimal. How can that be? The running time of the algorithm is $2^{O(1/\epsilon)}n$. For every fixed ϵ, therefore, the algorithm takes $O(n)$ time. For this reason, we consider it a *linear-time* approximation scheme.

In this section, we describe a methodology for deriving linear-time approximation schemes for a variety of optimization problems in planar graphs. The problems amenable to this methodology include minimum-weight vertex cover, minimum-weight edge dominating set, and maximum-weight independent set.

The methodology can be used for these problems in part because of the following property:

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

For example, if S is any vertex cover for G then the subset of S belonging to H is a vertex cover for H.

[3]

14.7.1 The subgraph induced by k BFS levels has branchwidth at most $2k$

The basis for these approximation schemes (and for others discussed in subsequent chapters) is breaking a planar graph into subgraphs each consisting of a

small sequence of breadth-first search levels. The following lemma shows that such a subgraph has small branchwidth.

Let G be a plane graph. Fix a root vertex r, and consider the breadth-first-search levels of vertices and edges of G with respect to r. Let $V_\ell = \{v \in V(G) : \text{level}(v) = \ell\}$.

Lemma 14.7.1 (BFS-Branchwidth Lemma)**.** *$G[V_{\ell+1} \cup V_{\ell+2} \cup \cdots \cup V_{\ell+k}]$ and its dual have branchwidth at most $2k$.*

Proof. Let G' be the graph obtained from G by deleting all vertices at levels greater than $\ell + k$ and contracting edges of the breadth-first-search tree whose endpoints have levels less than $\ell + 1$. The contractions result in a vertex r'. The remaining breadth-first-search-tree edges form an r'-rooted tree of depth at most k. By Lemma 14.6.1, therefore, G' and its dual G'^* have branchwidth at most $2k$. By Lemma 14.5.1, $G' - r'$ and its dual still have branchwidth at most $2k$, which proves the lemma. □

14.7.2

4

14.7.3 An approximation scheme for Vertex Cover

To find an approximately optimal solution to an optimization problem in a planar graph, we decompose the input graph into subgraphs to which the BFS-Branchwidth Lemma applies, solve the problem exactly in each subgraph, and combine these solutions to obtain a solution for the input graph. We repeat this process for several slightly different decompositions, and output the best solution thereby obtained.

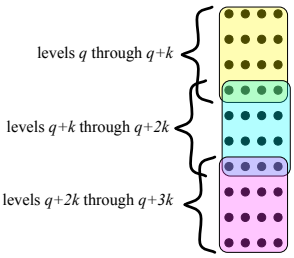
The form of the decomposition depends on whether the optimization problem is a maximization problem or a minimization problem, and whether the problem involves selection of vertices or selection of edges.

To illustrate the methodology, we will describe the approximation scheme for *minimum-weight vertex cover*. The input consists of an error parameter $\epsilon > 0$ and a plane graph G with an assignment weight$(\cdot)$ of weights to vertices.

The algorithm is as follows.

- Arbitrarily choose a vertex r of G. Execute breadth-first search on G, rooted at r.
- Let $k = \lceil 1/\epsilon \rceil$. For $i = 0, 1, \ldots, k-1$ and $j = -1, 0, 1, 2, \ldots$, let G_{ij} denote the subgraph of G induced by the vertices at levels $jk + i$ through $(j + 1)k + i$, and let S_{ij} be the minimum-weight vertex cover in G_{ij}.
- For $i = 0, 1, 2, \ldots, k-1$, let $S_i = \bigcup_j S_{ij}$, and return whichever of $S_0, \ldots, S_{k-1}$ has the least weight.

Now we show that each set S_i (and in particular the set returned by the algorithm) is a vertex cover. Fix $i = q$. The graphs G_{q0}, G_{q1}, G_{q2} form a decomposition of G. Each subgraph G_{qj} consists of a consecutive sequence of $k+1$ breadth-first-search levels, as shown below:



Two consecutive subgraphs in the decomposition overlap at a single level of vertices.

The overlap ensures that every edge e in G is at least one of $G_{q0}, G_{q1}, G_{q2}, \ldots$. Suppose e is in G_{qj} Then S_{qj} must contain one of e's endpoints, so e is covered by $\bigcup_j S_{qj}$. We have proved the following property:

> *Parts-to-whole property:* For each choice of i, the union of solutions for the subgraphs forming the decomposition is a solution for the whole graph.

This property implies that the algorithm's output is indeed a vertex-cover.

How fast is the algorithm? By the BFS-Branchwidth Lemma (Lemma 14.7.1), each subgraph G_{ij} has branchwidth at most $2(k+1)$, so, by Theorem 14.5.2, the minimum-weight vertex cover S_{ij} can be found in time $O(4^{2(k+1)}|V(G_{ij})|)$. Summing over all i and j, the total time is $O(k4^{2(k+1)}n)$.

Performance analysis

Now we prove the algorithm returns a solution whose weight is at most $1+\epsilon$ times optimal. The analysis is based on the whole-to-parts property given in Section 14.7:

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

Let OPT denote a minimum-weight vertex cover. Since

$$\begin{aligned} OPT &\cap \{\text{vertices whose levels are congruent to 0 mod } k\} \\ OPT &\cap \{\text{vertices whose levels are congruent to 1 mod } k\} \\ &\vdots \\ OPT &\cap \{\text{vertices whose level are congruent to } k-1 \text{ mod } k\} \end{aligned}$$

are disjoint,

$$\sum_i \text{weight}(OPT \cap \{\text{vertices whose levels are congruent to } i \text{ mod } k\}) \leq \text{weight}(OPT)$$

This shows that the average weight is at most $(1/k)$weight(OPT), so at least one of these sets has weight at most $(1/k)$weight(OPT). (This argument is called *averaging*.)

Let q be the integer whose congruence class has the least weight. Consider the decomposition

$$G_{q0}, G_{q1}, G_{q2}, \ldots$$

Every vertex of G occurs in exactly one of these subgraphs except for the vertices whose levels are congruent to q, and these vertices occur in exactly two. In particular, every vertex in OPT occurs exactly once in

$$OPT \cap V(G_{q0}), OPT \cap V(G_{q1}), OPT \cap V(G_{q2}), \ldots$$

except for the vertices in OPT whose levels are congruent to q, which occur twice. The sum of weights

$$\text{weight}(OPT \cap V(G_{q0})) + \text{weight}(OPT \cap V(G_{q1})) + \text{weight}(OPT \cap V(G_{q2}) + \cdots$$

is therefore

$$\text{weight}(OPT) + \text{weight}(OPT \cap \{\text{vertices whose level are congruent to } q \bmod k\})$$

which is at most $(1 + (1/k))$weight(OPT).

For $j = 0, 1, 2, \ldots$, by the whole-to-parts property, the minimum weight of a vertex cover of G_{qj} is at most weight$(OPT \cap V(G_{q0}))$ so

$$\begin{aligned} S_q &= \sum_j \text{minimum weight of a vertex cover of } G_{qj} \\ &\leq (1 + (1/k))\text{weight}(OPT) \end{aligned}$$

The algorithm does not know OPT, so does not know q. However, the algorithm returns whichever of $S_0, \ldots, S_{k-1}$ has minimum weight, so the weight of the solution returned is at most that of S_q. This shows that the solution returned is at most $(1 + \epsilon)$weight(OPT).

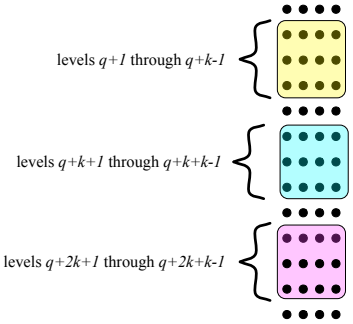
Remark 14.7.2. *The approximation error in this algorithm results from the design of the decomposition into parts, and that design in turn is chosen to satisfy the* parts-to-whole *property.*

Remark 14.7.3. *Trying all the congruence classes is critical to this methodology. In the next chapter, we will see a different methodology in which the algorithm can explicitly select the best congruence class.*

14.7.4 An approximation scheme for maximum-weight independent set

Now we illustrate how the methodology can be used to obtain an approximation algorithm for a maximization problem, namely INDEPENDENT SET. The approximation scheme must use a decomposition that satisfies the *parts-to-whole*

property. In order to ensure that the union of independent sets of parts is an independent set of the whole, we must ensure that vertices in distinct parts are not adjacent. We therefore use a decomposition of the following form.



As before, each subgraph in the decomposition consists of a consecutive sequence of breadth-first-search levels, but this time consective subgraphs are separated by a single level of vertices. This ensures that no edge connects vertices in distinct subgraphs, establishing the parts-to-whole property.

Given error parameter $\epsilon > 0$, plane graph G, and assignment weight$(\cdot)$ of weights to vertices, the algorithm is as follows.

- Arbitrarily choose a vertex r of G. Execute breadth-first search on G, rooted at r.
- Let $k = \lceil 1/\epsilon \rceil$. For $i = 0, 1, \ldots, k-1$ and $j = -1, 0, 1, 2, \ldots$ let G_{ij} denote the subgraph of G induced by the vertices at levels $jk + i + 1$ through $(j + 1)k + i - 1$, and let S_{ij} be the maximum-weight independent set in G_{ij}.
- For $i = 0, 1, 2, \ldots, k-1$, let $S_i = \bigcup_j S_{ij}$, and return whichever of $S_0, \ldots, S_{k-1}$ has the greatest weight.

As in the approximation scheme for minimum-weight vertex cover, the maximum-weight independent sets in the graphs G_{ij} can be found in time $2^{O(k)}n$. Thus the running time of the algorithm is $2^{O(k)}n$.

Performance analysis

The analysis is similar to that for vertex cover. As required by the *whole-to-parts* property, an independent set of the whole graph induces an independent set in any subgraph. Let OPT denote a maximum-weight independent set. Since

$$\begin{aligned} OPT &\cap \{\text{vertices whose levels are congruent to } 0 \bmod k\} \\ OPT &\cap \{\text{vertices whose levels are congruent to } 1 \bmod k\} \\ &\vdots \\ OPT &\cap \{\text{vertices whose level are congruent to } k-1 \bmod k\} \end{aligned}$$

are disjoint, at least one of these sets has weight at most $(1/k)\text{weight}(OPT)$. Let q be the integer whose congruence class has the least weight.

For each j, $OPT \cap V(G_{qj})$ is an independent set, so the the weight of S_{qj}, the maximum-weight independent set, is at least that of $OPT \cap V(G_{qj})$. Every vertex of OPT is in $\bigcup_j V(G_{qj})$ except for the vertices whose levels are congruent to q mod k. Therefore

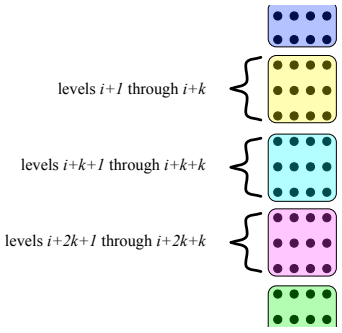
$$\begin{aligned}
\text{weight}(S_q) &= \text{weight}(\bigcup_j S_{qj}) \\
&\leq \text{weight}(OPT) - \text{weight}(\{\text{vertices whose levels are congruent to } q \text{ mod } k\}) \\
&\leq \text{weight}(OPT) - \frac{1}{k}\text{weight}(OPT) \\
&\leq (1-\epsilon)\text{weight}(OPT)
\end{aligned}$$

Remark 14.7.4. *Once again, the approximation error comes from the design of the decomposition, and that design is chosen to satisfy the* parts-to-whole *property.*

14.7.5 Maximum-weight set of edge-disjoint triangles

Now we show how to apply the methodology to a maximization problem involving edges. A *triangle* is a set of three edges xy, yz, xz on three endpoints. The goal is to find a maximum-weight set of edge-disjoint triangles. The approximation scheme for this problem can be formulated like the others, but we take a slightly different approach to illustrate another interpretation.

As before, let $k = 1/\epsilon$, and consider the breadth-first search levels with respect to an arbitrary vertex r. For $i = 0, 1, \ldots, k-1$, let E_i be the set of edges of G whose levels are congruent mod k to i. Removing E_i from G results in disconnected subgraphs of the form $G[V_{jk+i+1} \cup \cdots \cup V_{jk+i+k}]$.



Each of these subgraphs has branchwidth at most $2k$, so their union has branchwidth at most $2k$. We summarize this result in a lemma.

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 $2k$.*

For $i = 0, 1 \ldots, k$, the approximation finds the maximum-weight set of edge-disjoint triangles in $G - E_i$, and returns the solution of greatest weight. Because the branchwidth of $G - E_i$ is at most $2k$, the optimum solution in this graph can be found in time $2^{O(k)}n$.

Performance analysis

The performance analysis resembles that of maximum independent set. Let OPT be a maximum-weight set of edge-disjoint triangles. For $i = 0, \ldots, k-1$, let S_i be the set of triangles in OPT that have edges in E_i. Note that each triangle has edges in at most one set E_i. Let $q = \text{minarg}_i \text{weight}(S_i)$. Then $\text{weight}(S_q) \leq \frac{1}{k}\text{weight}(OPT)$. Thus the maximum weight of a set of edge-disjoint triangles in $G - E_q$ is at least $(1 - \frac{1}{k})\text{weight}(OPT)$.

14.7.6 Summary of approximation-scheme methodology

To summarize some aspects of the methodology, for a given maximization or minimization problem, one first chooses a family of decompositions of the graph. Each decomposition in the family consists of a collection of subgraphs, and each subgraph is that induced by a short sequence of consecutive levels (breadth-first search levels in the graph or its face-vertex incidence graph). The subgraphs comprising a decomposition are mostly disjoint; in the case of a maximization problem, a pair of consecutive subgraphs are separated by a small number of levels, and in the case of a minimization problem, they overlap on a small number of layers.

The maximization/minimization problem should satisfy the whole-to-parts property, i.e. a solution for the whole graph should induce a solution on each subgraph in a decomposition. Moreover, each decomposition should be chosen so as to satisfy the parts-to-whole property, i.e. the union of solutions for the subgraphs should constitute a solution for the entire graph.

The minimization/maximization problem might have to be slightly generalized in order to ensure that the whole-to-parts and parts-to-whole properties are satisfied. The following problems illustrate this.

Problem 14.5. *Give a linear-time approximation scheme for* minimum-weight edge dominating set *in planar graphs.*

Problem 14.6. *A* dominating set *of a graph is a set S of vertices such that each vertex of the graph is within one hop of a vertex of S, i.e. is in S or is adjacent to a vertex in S. Give a linear-time approximation scheme for* minimum-weight dominating set *in planar graphs.*