

Our solution will be similar to the algorithm for VERTEX COVER from the first section of this chapter. Consider the notation defined in the problem. For an element  $a \in A$ , we *reduce the instance by  $a$*  by deleting  $a$  from  $A$ , and deleting all sets  $B_i$  that contain  $a$ . Thus, reducing the instance by  $a$  producing a new, presumably smaller, instance of HITTING SET.

We observe the following fact. Let  $B_i = \{x_1, \dots, x_c\} \subseteq A$  be any of the given sets in the HITTING SET instance. Then at least one of  $x_1, \dots, x_c$  must belong to any hitting set  $H$ . So by analogy with (2.3) from the notes, we have the following fact

- Let  $B_i = \{x_1, \dots, x_c\}$  There is  $k$ -element hitting set for the original instance if and only if, for some  $i = 1, \dots, c$ , the instance reduced by  $x_i$  has a  $(k - 1)$ -element hitting set.

The proof is completely analogous to that of (2.3). If  $H$  is a  $k$ -element hitting set, then some  $x_i \in H$ , and so  $H - \{x_i\}$  is a  $(k - 1)$ -element hitting set for the instance reduced by  $x_i$ . Conversely, if the instance reduced by  $x_i$  has a  $(k - 1)$ -element hitting set  $H'$ , then  $H' \cup \{x_i\}$  is a  $k$ -element hitting set for the original instance.

Thus, our algorithm is as follows. We pick any set  $B_i = \{x_1, \dots, x_c\}$ . For each  $x_i$ , we recursively test if the instance reduced by  $x_i$  has a  $(k - 1)$ -element hitting set. We return “yes” if and only if the answer to one of these recursive calls is “yes.” Our running time satisfies  $T(m, k) \leq cT(m, k - 1) + O(cm)$ , and so it satisfies  $T(m, k) = O(c^k \cdot km)$ . This gives the desired bound, with  $f(c, k) = kc^{k+1}$  and  $p(m) = m$ .

---

<sup>1</sup>ex579.588.787

(a) We prove this by induction on  $d$ . If  $d = 0$ , then  $\Phi$  is a satisfying assignment, and  $Explore(\Phi, d)$  returns “yes.”

Now consider  $d > 0$ . If  $Explore(\Phi, d)$  returns “yes,” it is because one of the recursive calls  $Explore(\Phi_i, d - 1)$  returns “yes”; by induction, this means that  $\Phi_i$  has distance  $d - 1$  to a satisfying assignment, and so  $\Phi$  has distance  $d$  to a satisfying assignment.

Conversely, suppose  $\Phi$  has distance  $d$  to a satisfying assignment  $\Phi'$ . Consider any clause unsatisfied by  $\Phi$ ; since  $\Phi'$  satisfies it, it must disagree with  $\Phi$  on the setting of at least one of the variables in this clause. Thus, one of the assignments  $\Phi_i$ , which changes the assignment to this variable, is at distance  $d - 1$  to  $\Phi'$ ; by induction the recursive call  $Explore(\Phi_i, d - 1)$  will return “yes,” and so the full call  $Explore(\Phi, d)$  will also return “yes.”

The running time for  $Explore$  satisfies the recurrence  $T(n, d) \leq 3T(n, d - 1) + p(n)$ , for a polynomial  $p$ . Unwinding this to get  $d$  down to 0, we have a running time of  $O(3^d \cdot p(n))$ .

(b) We let  $\Phi_0$  denote the assignment in which all variables are set to 0, and we let  $\Phi_1$  denote the assignment in which all variables are set to 1. If there is any satisfying assignment, it is within distance at most  $n/2$  of one of these, so we can call both  $Explore(\Phi_0, n/2)$  and  $Explore(\Phi_1, n/2)$ , and see if either of these returns “yes.”

The running time of each of these calls is  $O(p(n) \cdot 3^{n/2}) = O(p(n) \cdot (\sqrt{3})^n)$ .

---

<sup>1</sup>ex695.88.327

Consider an ordered triple  $(S, i, j)$ ,  $1 \leq i, j \leq n$  and  $S$  is a subset of the vertices that includes  $v_i$  and  $v_j$ . Let  $B[S, i, j]$  denote the answer to the question, “Is there a Hamiltonian path on  $G[S]$  that starts at  $v_i$  and ends at  $v_j$ ?” Clearly, we are looking for the answer to  $B[V, 1, n]$ .

We now show how to construct the answers to all  $B[S, i, j]$ , starting from the smallest sets and working up to larger ones, spending  $O(n)$  time on each. Thus the total running time will be  $O(2^n \cdot n^3)$ .

$B[S, i, j]$  is true if and only if there is some vertex  $v_k \in S - \{v_i\}$  so that  $(v_i, v_k)$  is an edge, and there is a Hamiltonian path from  $v_k$  to  $v_j$  in  $G[S - \{v_i\}]$ . Thus, we set  $B[S, i, j]$  to be true if and only if there is some  $v_k \in S - \{v_i\}$  for which  $(v_i, v_k) \in E$  and  $B[S - \{v_i\}, k, j]$  is true. This takes  $O(n)$  time to determine.

We claim that such a graph  $G$  has a tree decomposition  $(T, \{V_t\})$  in which each piece  $V_t$  corresponds uniquely to an internal triangular face of  $G$ . We prove this by induction on the number of nodes in  $G$ .

Choose any internal edge  $e = (u, v)$  of  $G$ ; deleting  $u$  and  $v$  produces two components  $A$  and  $B$ . Let  $G_1$  be the subgraph induced on  $A \cup \{u, v\}$  and  $G_2$  the subgraph induced on  $B \cup \{u, v\}$ . By induction, there are tree decompositions  $(T_1, \{X_t\})$  and  $(T_2, \{Y_t\})$  of  $G_1$  and  $G_2$  respectively in which the pieces correspond uniquely to internal faces. Thus there are nodes  $t_1 \in T_1$  and  $t_2 \in T_2$  that correspond to the faces containing the edge  $(u, v)$ . If we let  $T$  denote the tree obtained by adding an edge  $(t_1, t_2)$  to  $T_1 \cup T_2$ , then  $(T, \{X_t\} \cup \{Y_t\})$  is a tree decomposition having the desired properties.

---

<sup>1</sup>ex203.262.545

(a) Let us root the tree at an arbitrary node  $r$ , and define subtrees  $T_u$  as we have done in Chapter 10.2 when solving the Weighted Independent Set Problem. There we defined two subproblems corresponding to each subtree depending on whether or not we include the root  $u$  in the set. We will use the same subproblems:  $OPT_{in}(u)$  denotes the maximum weight of an independent set of  $T_u$  that includes  $u$ , and  $OPT_{out}(u)$  denotes the maximum weight of an independent set of  $T_u$  that does not include  $u$ . Now the optimum we are looking for is  $\min(OPT_{in}(r), OPT_{out}(r))$ . It helps us to define a third subproblem for each subtree:  $OPT_{un}(u)$  denotes the maximum weight of an independent set of  $T_u$  that does not have to dominate  $u$ . When  $u$ 's parent is included in the dominating set,  $u$  is already dominated by its parent, hence the set selected in the subtree  $T_u$  does not have to dominate  $u$ .

Now that we have our sub-problems, it is not hard to see how to compute these values recursively. For a leaf  $u \neq r$  we have that  $OPT_{out}(u) = \infty$ ,  $OPT_{in}(u) = c(u)$ , and  $OPT_{un}(u) = 0$ . For all other nodes  $u$  we get a recurrence that defines  $OPT_{out}(u)$ ,  $OPT_{in}(u)$ , and  $OPT_{un}(u)$  using the values for  $u$ 's children.

(1) For a node  $u$ , the following recurrence defines the values of the sub-problems:

- $OPT_{in}(u) = c(u) + \sum_{v \in \text{children}(u)} OPT_{un}(v)$
- $OPT_{un}(u) = \sum_{v \in \text{children}(u)} \min(OPT_{in}(v), OPT_{out}(v))$
- $OPT_{out}(u) = \min_{v \in \text{children}(u)} (OPT_{in}(v) + \sum_{w \in \text{children}(u), w \neq v} \min(OPT_{out}(v), OPT_{in}(v)))$ .

Using this recurrence, we get a dynamic programming algorithm by building up the optimal solutions over larger and larger sub-trees. We define arrays  $Mo[u]$ ,  $Mi[u]$  and  $Mu[u]$ , which hold the values  $OPT_{out}(u)$ ,  $OPT_{in}(u)$  and  $OPT_{un}(u)$  respectively. For building up solutions, we need to process all the children of a node before we process the node itself.

To find a minimum-weight dominating set of a tree  $T$ :

Root the tree at a node  $r$ .

For all nodes  $u$  of  $T$  in post-order

    If  $u$  is a leaf then set the values:

$$\begin{aligned} Mo[u] &= \infty \\ Mi[u] &= c(u) \\ Mu[u] &= 0 \end{aligned}$$

    Else set the values:

$$\begin{aligned} Mi[u] &= c(u) + \sum_{v \in \text{children}(u)} Mu[v]. \\ Mu[u] &= \sum_{v \in \text{children}(u)} \min(Mi[v], Mo[v]). \end{aligned}$$

---

<sup>1</sup>ex573.411.14

$$Mo[u] = \min_{v \in \text{children}(u)} Mi[v] + \sum_{w \in \text{children}(u), w \neq v} \min(Mo[v], Mi[v])$$

Endif  
Endfor  
Return  $\max(Mo[r], Mi[r])$ .

The algorithms clearly runs in polynomial time, as there are  $3n$  subproblems for an  $n$  node tree, and each value associated with a subproblem of  $u$  of degree  $n_u$  can be determined in  $O(n_u)$  time. So the total time is  $O(\sum_u n_u) = O(n)$ .

(b) We extend the algorithm for the case of bounded tree-width by having subproblems associated with the nodes of the tree-decomposition. For each node  $t$  of the tree-decomposition, let  $V_t$  be the subset of nodes corresponding to tree-node  $t$ ,  $T_t$  the subtree rooted at  $t$ , and  $G_t$  the corresponding subgraph. Now for each disjoint sets  $U, W \subset V_t$  we will define a subproblem and have  $OPT(t, U, W)$  the minimum weight of a set in  $G_t$  that contains exactly the nodes  $U$  in  $V_t$  and covers all nodes of  $G_t$  except possibly not dominating the a subset of  $W$ . Recall that in the case of a tree, the recurrence for  $OPT_{out}(u)$  was a little awkward, as we needed to select a child of  $u$  that is covering  $u$ . Here we would need to do this for each node in  $V_t - (U \cup W)$ . To make this simpler to write, we will define more subproblems. Let  $t$  be a node of the tree decomposition, and let  $t_1, \dots, t_d$  be its children, then we define subproblems  $OPT(t, i, U, W)$  for each  $0 \leq i \leq d$  to be the minimum weight of a set in the graph corresponding to the union of subtrees  $T_{t_1}, \dots, T_{t_i}$  and the set  $V_t$  that contains exactly the nodes  $U$  in  $V_t$  and dominates all nodes of this subgraph except possibly not dominating the a subset of  $W$ .

So if  $d_r$  is the degree of the root, then the optimum value we are looking for is  $\min_{U \subset V_r} OPT(r, d_r, U, \emptyset)$ . For any node  $t$  we have  $OPT(t, 0, U, W) = \sum_{u \in U} c(u)$  if  $U$  dominates the nodes  $V_t - W$ , and  $\infty$  otherwise. This defines the values at the leafs.

Given the subproblems, we will get the value at a node  $t$  using the values for smaller  $i$ , and the values at the children of  $t$  as follows. For a set  $U$  we will use the notation  $c(U) = \sum_{u \in U} c(u)$ , and  $\delta(U)$  is the set dominated by  $U$ . Let  $t$  be a node of the tree decomposition and  $t_1, \dots, t_d$  its children, let  $n_i$  be the degree of child  $i$ .

(2) The value of  $OPT(t, i, U, W)$  for  $i \geq 1$  is given by the following recurrence:

$$OPT(t, i, U, W) = c(U) + \min_{U_i \subseteq V_{t_i}: U_i \cap V_t = U \cap V_{t_i}} (OPT(t, i-1, U, W \cup \delta(U_i)) + OPT(t_i, n_i, U_i, (\delta(U) \cup W) \cap V_{t_i}) - c(U \cap V_{t_i}))$$

For a tree-decomposition of width  $k$  there are  $3^{k+1}$  subproblems associated with a  $(t, i)$  pair, and there are  $n$  such pairs if the tree is of size  $n$ . Computing each value takes only  $O(1)$  time, so the total time required is  $O(3^k n)$ .

Let  $(T; \{V_t | t \in T\})$  be the given tree decomposition rooted at  $r$ . There are  $k$   $(s_i; t_i)$  terminal pairs. We focus on the tree-width 2 case. For convenience, we assume that there are no two pieces  $V_{t_1}$  and  $V_{t_2}$  where  $(t_1, t_2)$  is an edge and  $V_{t_1} \subset V_{t_2}$ . Consider the subgraph  $G_t$ . Note that there can be at most one  $i$  for which  $P_i$  both enters and leaves  $G_t$ , since any such path uses up at least 2 vertices of  $V_t$ . Note also that there can be at most 3  $s_i-t_i$  terminal pairs that have one end in  $G_t$  and the other one outside (as the paths connecting such pairs must go through  $V_t$ ).

- If there are 3 such pairs, that each node  $v \in V_t$  must be connected via disjoint paths to one of them, and terminal pairs inside  $G_t$  must connect via paths inside  $G_t$ . There are  $O(1)$  cases here to consider depending on which of the nodes in  $V_t$  is used to connect which of the 3 separated terminal pairs.
- If there are 2 such terminal pairs, than of the at most 3 nodes in  $V_t$  2 must be connected via disjoint paths to one of them, the third is either not used in any of the paths or is used by path connecting two terminals  $s_i$  and  $t_i$  inside, or two terminals outside  $G_t$ . Now there are  $O(k)$  cases to consider, depending on which terminal pair  $i$  is using the extra node.
- If there is only one such pair, than we can have one pair  $i$  with both terminals inside or outside of  $G_t$ , that uses 2 nodes in  $V_t$ , or the one path leaving  $G_t$ , can leave, come back and leave again, or one or two paths can use just one node in  $V_t$  while having both terminals inside or both outside of  $G_t$ . There are  $O(k^2)$  cases to consider here.
- If there are no such pairs, than one path can use 2 or 3 nodes in  $V_t$ , or multiple paths can use one node each. Now there are  $O(k^3)$  cases to consider.

We define multiple subproblems for each  $t$  according to the possibilities discussed above. For the at most 3 nodes in  $V_t$  there are at most  $O(k^3)$  possible cases. This defines  $O(k^3n)$  subproblems. The value of a subproblem is simply 0 or 1 (or true or false) depending whether or not there are disjoint paths in  $G_t$  that satisfy the state of the nodes in  $V_t$  corresponding to the subproblem, that is, connect each  $v \in V_t$  to the terminal in question inside  $G_t$  (and possibly connect the two nodes in  $V_t$  to each other, if needed), via disjoint paths inside  $G_t$ . The desired disjoint paths exists if and only if the value of one of the subproblems that connects all terminal pairs within the subgraph  $G_r$  (which is the whole graph).

Given values for all the subproblems associated with the children  $t_1, \dots, t_d$  of a node  $t$ , we want to get the value of the given subproblem efficiently. To do this consider a node  $v \in V_t$ , the subproblem under question wants a particular paths  $P_i$  to go through this vertex in  $O(d)$  time.

---

<sup>1</sup>ex209.650.476

Checking whether  $G$  is 1- or 2-colorable is easy. For  $k = 3, 4, \dots, w + 1$ , we test whether  $G$  is  $k$ -colorable by dynamic programming. We use notation similar to what we used in the Maximum-Weight Independent Set problem for graphs of bounded tree-width. Let  $(T, \{V_t : t \in T\})$  be a tree decomposition of  $G$ . For the subtree rooted at  $t$ , and every coloring  $\chi$  of  $V_t$  using the color set  $\{1, 2, \dots, k\}$ , we have a predicate  $q_t(\chi)$  that says whether there is a  $k$ -coloring of  $G_t$  that is equal to  $\chi$  when restricted to  $V_t$ . This requires us to maintain  $k^{(w+1)} \leq (w+1)^{(w+1)}$  values for each piece of the tree decomposition.

We compute the values  $q_t(\chi)$  when  $t$  is a leaf by simply trying all possible colorings of  $G_t$ . In general, suppose  $t$  has children  $t_1, \dots, t_d$ , and we know the values of  $q_{t_i}(\chi)$  for each choice of  $t_i$  and  $\chi$ . Then there is a coloring of  $G_t$  consistent with  $\chi$  on  $V_t$  if and only if there are colorings of the subgraphs  $G_{t_1}, \dots, G_{t_d}$  that are consistent with  $\chi$  on the parts of  $V_{t_i}$  that intersect with  $V_t$ . Thus we set  $q_t(\chi)$  equal to *true* if and only if there are colorings  $\chi_i$  of  $V_{t_i}$  such that  $q_{t_i}(\chi_i) = \text{true}$  and  $\chi_i$  is the same as  $\chi$  when restricted to  $V_t \cap V_{t_i}$ .

---

<sup>1</sup>ex897.854.812



We build the following bipartite graph  $G$ . There is a node  $u_i$  for each variable  $x_i$ , a node  $v_j$  for each clause  $C_j$ , and an edge  $(u_i, v_j)$  whenever  $x_i$  appears in  $C_j$ .

We first note that since each variable appears three times, and each clause has length three, the number of nodes on the left side of  $G$  equals the number of nodes on the right side. More strongly, we claim that  $G$  in fact has a perfect matching.

This is a consequence of a more general claim: that if every node in a bipartite graph  $G$  has the same degree  $d$ , then  $G$  has a perfect matching. (Here  $d = 3$ .) Indeed, if  $G$  does not have a perfect matching, then by Hall's Theorem it has a set  $A$  on the left side for which  $|A| < |\Gamma(A)|$ , where  $\Gamma(A)$  denotes the set of neighbors of any node in  $A$ . But  $A$  has  $d|A|$  nodes coming out of it, and  $\Gamma(A)$  can only absorb  $d|\Gamma(A)|$  of them, so this is not possible.

Consider the perfect matching in the graph  $G$  we constructed from the  $3$ -SAT instance. For each variable, we set it in a way that satisfies the clause it is matched to. In this way, we satisfy the full collection of clauses.

---

<sup>1</sup>ex592.206.332

We root the tree at some node  $r$  and associate a subtree  $T_v$  with each node  $v \in T$ , and let  $n_v$  denote the number of nodes in the subtree  $T_v$ . A solution for the graph partitioning problem may split the subtree  $T_v$  in any ratio, however, it is not hard to see that if  $(A, B)$  is the maximum weight cut splitting the tree  $T$  into two equal sides, and assume  $v \in A$  and  $|A \cap T_v| = k$ , then the induced partition of  $T_v$  by having  $A_v = A \cap T_v$ , and  $B_v = B \cap T_v$  is the maximum weight cut separating  $T_v$  into two sides, where the side containing  $v$  has size  $k$ . This is true as the only interaction of the partition of  $T_v$  with the remaining graph is through node  $v$ . More precisely, assume it is false, and consider the optimal such cut  $(A', B')$ . Replacing  $A_v$  by  $A'$  keeps all edges across the cut, except replacing edges that cross the  $(A_v, B_v)$  cut with edges that cross the  $(A', B')$  cut, and hence it results in a cut of larger weight. This contradiction proves the claim.

Given the above observation, we define subproblems for each subtree  $T_v$  and each integer  $k \leq n_v$ , where  $OPT(v, k)$  is the maximum cut of the subgraph  $T_v$  separating  $T_v$  into two sides where the side containing  $v$  has size  $k$ . The final answer is  $OPT(r, n/2)$  if  $n = n_r$  is the number of nodes in  $T$ .

We will use  $c(u, v)$  as the cost or weight of the edge  $e = (v, w)$  of the tree. For a leaf  $v$  we have  $k = 1$  as the only possible value and  $OPT(v, 1) = 0$ . Now consider  $OPT(v, k)$  for some non-leaf  $v$ . The side  $A$  containing  $v$  must contain  $k - 1$  nodes in addition to node  $v$ . If  $v$  has only one child  $u$  then we need to consider two cases depending on whether or not the child  $u$  is on the same side of the cut as  $v$ , so we get that in this case

$$OPT(v, k) = \max(opt(u, k - 1), c(v, u) + OPT(u, n_u - k + 1)).$$

If  $v$  has two children  $u$  and  $w$ , then we will consider all possible ways of dividing these  $k - 1 = \ell_1 + \ell_2$  nodes between the two subtrees rooted at the two children of  $v$ . For each such division, there are further cases depending whether or not the root of these subtrees is on the same side of the cut as  $v$ . We get the following recurrence in this case

$$\begin{aligned} OPT(v, k) = & \max_{0 \leq \ell_1 \leq k-1, \ell_2 = k-1-\ell_1} (\max(OPT(u, \ell_1), c(v, u) + OPT(u, n_u - \ell_1)) \\ & + \max(OPT(w, \ell_2), c(v, w) + OPT(w, n_w - \ell_2))). \end{aligned}$$

There are  $O(n^2)$  subproblems. We can compute the values of the subproblems starting at the leaves, and gradually considering bigger subtrees. This recurrence allows us to compute the value of a subproblem in  $O(1)$  time given the values of the subproblems associated with smaller trees. So the total time required is  $O(n^2)$ .