3
An Exact Exponential Algorithm for the 1-Hotlink Assignment Problem

As stated in the introduction, the proposed FPTAS for the 1-HAP makes use of the exact exponential algorithm presented in [PLS04]. For sake of completeness, we present a brief overview of this exact algorithm, called PATH. However, the FPTAS uses it as a 'black box' so the reader can skip this section without compromising the understanding of subsequent content.

With slight abuse of notation, we say that the height of an assignment A for a tree T is the height of the tree it induces, that is the height of T^A. Given a parameter D selected by the user, the algorithm PATH finds the best 1-assignment among the 1-assignments with height at most D. Therefore, by executing PATH with D set as the number of nodes of the input tree we can find an optimal 1-assignment.

A straightforward approach to solving the 1-HAP using dynamic programming would be the following. For each node u, we compute the best non-crossing 1-assignment A_u which contains the hotlink (r, u) and then select the best 1-assignment A_u among all u's. (Notice that as mentioned previously we can focus only on non-crossing assignments without compromising the optimality.) However, each assignment A_u has a structure that can be exploited on a dynamic programming fashion due to the following fact: apart from (r, u), all other hotlinks in A_u have both endpoints in $T - (T_u \cup \{r\})$ or in T_u. If that was not the case, the property that A_u is non-crossing would imply that there is another hotlink in A_u starting at r, contradicting the fact that A_u is a 1-assignment. In effect, the task of finding the assignment A_u can then be divided into finding the best assignment for $T - (T_u \cup \{r\})$ and for T_u separately. Furthermore, $T - (T_u \cup \{r\})$ is a forest composed by the trees $\{T_v - T_u\}_{v \in \text{child}(r)}$, where child$(r)$ denotes the set children of r in T, so we can describe this strategy with the following recursion:

$$\text{OPT}(T) = \min_{u \in T} \left\{ \sum_{v \in child(r)} \text{OPT}(T_v - T_u) + \text{OPT}(T_u) \right\}$$

Informally, it can be proved that if we truncate the recursion at depth D, this procedure finds an optimal 1-assignment among the 1-assignments with height at most D in $O(n^D)$ time.

As stated before, the time complexity of the PATH algorithm is simply exponential on the parameter D. For that, PATH uses the following strategy. For a given input tree rooted at r, it considers only two possibilities: assigning or not a hotlink from r to some node in T_f, where f is the last child of r in T (assuming some order). If such a hotlink is not assigned, then we must solve a subproblem with T_f as an input tree and another subproblem with $T - T_f$ as an input tree. On the other hand, if a hotlink is assigned from r to some node in T_f, then we must solve two modified subproblems. In the first subproblem, the input tree T_f has an additional hotlink available from one level lower than its root. The second subproblem has $T - T_f$ as an input tree where no hotlink can be assigned from the root r (since it must be assigned to some node in T_f). This approach leads to the more general 1-HAPG problem defined below.

Input:

i) a directed path $Q = (V_Q, E_Q)$ where $V_Q = \{q_1, \ldots, q_k\}$ and $E_Q = \{(q_i, q_{i+1}) : 1 \leq i \leq k-1\}$

ii) a vector $\mathbf{a} = (a_1, \ldots, a_k, a_{k+1}, b) \in \{0,1\}^{k+2}$

iii) a tree $\overline{T} = (V, E)$ rooted at r

iv) an integer D

v) a weight function w, where $w(u)$ is nonzero only if u is a leaf of $\overline{T}$

Output: A 1-assignment A to the tree $T_Q = (V_Q \cup V, E_Q \cup E \cup \{(q_k, r)\})$, satisfying the following six conditions:

(a) A is feasible in the sense of the 1-HAP

(b) No hotlink can point to a node in V_Q

(c) If $a_i = 0$, then no hotlink can leave q_i

(d) If $a_{k+1} = 0$, then no hotlink can leave r

(e) If $b = 0$, then no hotlink can point to r

(f) The height of the enhanced tree T_Q^A is at most D

Objective: Minimize $\mathrm{EP}(\overline{T}, A, w)$.

Observe that the 1-HAP is a particular case of the 1-HAPG when Q is empty, $b = a_1 = 1$, $\overline{T} = T$ and D is the number of nodes of T. Thus, an exact algorithm for 1-HAPG is also an exact algorithm for 1-HAP.

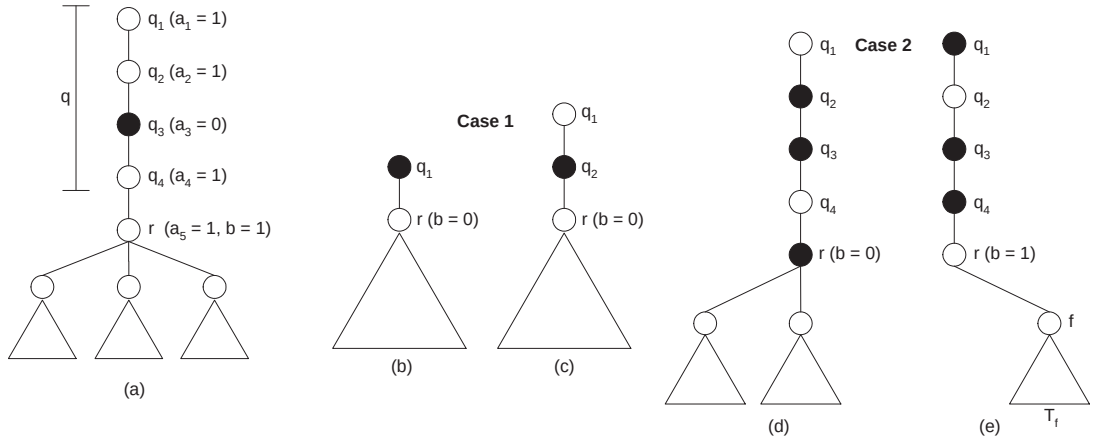

Figure 3.1: (a) An instance of the 1-HAPG problem. (b) and (c) two possible subproblems generated in Case 1. (d) and (e) the decomposition in Case 2 when $\mathbf{c} = (0, 1, 0, 0, 1)$.

3.1 Solving 1-HAPG

Figure 3.1 is used throughout this section to illustrate the PATH execution. Figure 3.1.a presents an instance of 1-HAPG where the path Q consists of four nodes q_1, q_2, q_3 and q_4. If a node $q_i \in Q$ is such that $a_i = 1$, then q_i is said to be *available*. The only non-available node, q_3, is black colored. Since $a_5 = b = 1$, hotlinks can be assigned from and to the node r.

We need to introduce the following conventions: given two binary vectors $\mathbf{c}$ and $\mathbf{d}$, we use $\mathbf{cd}$ to indicate the vector obtained by concatenating $\mathbf{c}$ and $\mathbf{d}$. For example, if $\mathbf{c} = (0, 1)$ and $\mathbf{d} = (1, 0, 0)$, then $\mathbf{cd} = (0, 1, 1, 0, 0)$. We use $\mathbf{c}_i$ to denote the vector obtained by removing all but the ith first components of c. Using the previous example, $\mathbf{c}_1 = (0)$ and $\mathbf{c}_2 = (0, 1)$. We also use $|\mathbf{c}|$ to denote the number of elements (zeros and ones) in the vector $\mathbf{c}$. For a directed path Q and a node u, we use $(Q \rightarrow u)$ to indicate the path obtained by inserting u at the end of Q. We use Q_i to denote the subpath of Q formed by its ith first nodes, and $|Q|$ denote the number of nodes in Q.

Let $\text{OPT}(Q, \mathbf{a}, \overline{T}, w)$ denote the cost of an optimal solution of a 1-HAPG instance. If $|Q| > D$, then PATH sets $\text{OPT}(Q, \mathbf{a}, \overline{T}, w) = \infty$. Hence, let us assume that $|Q| \leq D$. In order to solve this instance we must consider the following cases:

Case 1: some hotlink is assigned from a node of Q to r in the optimal solution;

Case 2: no hotlink is assigned from a nodes of Q to r in the optimal solution;

Case 1. This case is only considered when $b = 1$. In this case, we must add a hotlink from some available node to r. Thus, we have $\sum_{i=1}^{k} a_i$ possibilities. As an example, if (q_1, r) is assigned to the tree of Figure 3.1.a, then PATH generates the subproblem of Figure 3.1.b. In fact, the addition of hotlink (q_1, r) creates an improved tree where q_1 has two children: $\overline{T}$ and the path $(q_2 \rightarrow q_3 \rightarrow q_4)$. However, since q_2, q_3 and q_4 are not ancestors of nodes with nonzero weight in this enhanced tree (which must belong to $\overline{T}$), they can be removed without modifying the cost of the solution. Observe that b is set to 0 since we can assume that no two hotlinks point to the same node . In general, if some hotlink points to r in the optimal solution, we have that

$$\mathrm{OPT}(Q, \mathbf{a}, \overline{T}, w) = \min_{i \in \{1,2,\ldots,k\}:\ a_i = 1} \left\{ \mathrm{OPT}(Q_i, \mathbf{a}_{i-1}(0, a_{k+1}, 0), \overline{T}, w) \right\} \qquad (1)$$

Case 2. In this case all the available nodes of Q may only point to some node in $V - \{r\}$. Thus, PATH must decide which of the available nodes are allowed to point to the nodes of $\overline{T}_f$, the maximal subtree of $\overline{T}$ rooted at the last child f of r (assuming any order). Let $k' = \sum_{i=1}^{k+1} a_i$ be the number of available nodes. Then, PATH has $2^{k'}$ possibilities to take such a decision. Since it is not clear which one is the best, then all of them are considered.

In order to clarify this case, let us consider the possibility where q_2 and r remain available for $\overline{T}_f$ (see Figure 3.1.e). As a consequence, only q_1 and q_4 will be allowed to point to nodes in $\overline{T} - \overline{T}_f$ (Figure 3.1.d). Figure 3.1.d defines a new subproblem $(Q, \mathbf{a}', \overline{T} - \overline{T}_f, w)$, where $\mathbf{a}' = (1, 0, 0, 1, 0, 0)$. Note that b is set to 0 since we are in Case 2. On the other hand, Figure 3.1.e defines a new subproblem $(Q \rightarrow r, \mathbf{a}'', \overline{T}_f, w)$, where $\mathbf{a}'' = (0, 1, 0, 0, 1, 1, 1)$.

Thus, the sum of the optimal solutions for the subproblems defined by Figures 3.1.d and 3.1.e is the cost of the optimal solution for the problem of Figure 3.1.a under the assumption that no hotlink can be assigned to r (Case 2), the nodes q_2 and r cannot point to nodes in $\overline{T} - \overline{T}_f$, and the nodes q_1 and q_4 cannot point to nodes in $\overline{T}_f$.

In general, let C be a set of binary vectors defined by $C = \{(c_1, \ldots, c_{k+1}) : c_i \leq a_i \text{ for } i = 1, \ldots, k+1\}$. Each $\mathbf{c} \in C$ corresponds to one of the $2^{k'}$ possibilities for selecting the nodes that will remain available to point to nodes in $\overline{T}_f$. Furthermore, let $\bar{\mathbf{c}} = \mathbf{a} - \mathbf{c}$. This vector defines which nodes from $\mathbf{q}$ will remain available to point to nodes in $\overline{T} - \overline{T}_f$. Then, by considering all choices

for $\mathbf{c}$, we have that:

$$\begin{gathered}\mathrm{OPT}(Q, \mathbf{a}, \overline{T}, w) = \\ \min_{\mathbf{c} \in C}\{\mathrm{OPT}(Q \rightarrow r, \mathbf{c}(1,1), \overline{T}_f, w) + \mathrm{OPT}(Q, \overline{\mathbf{c}}(0), \overline{T} - \overline{T}_f, w)\} \end{gathered} \quad (2)$$

Cases 1 and 2 together. Let RHS_1 and RHS_2 be respectively the right-hand side of equations 1 and 2. Thus:

$$\mathrm{OPT}(Q, \mathbf{a}, \overline{T}, w) = \begin{cases} \mathrm{RHS}_2 & \text{if } b = 0 \\ \min\{\mathrm{RHS}_1, \mathrm{RHS}_2\} & \text{if } b = 1 \end{cases}$$

Stop conditions. If $\overline{T}$ has only one node, say l, then the best choice is to assign a hotlink from the first available node in Q to l. Thus:

$$\mathrm{OPT}(Q, \mathbf{a}, \overline{T}, w) = \begin{cases} \min\{i \cdot w(l) : 1 \leq i \leq k \text{ and } a_i = 1\}, & \text{if } Q \text{ has some} \\ & \text{available node} \\ k \cdot w(l), \quad \text{otherwise} & \end{cases}$$

In addition, if $|Q| > D$ then we set $\mathrm{OPT}(Q, \mathbf{a}, \overline{T}, w) = \infty$.

Computational complexity. First, let us analyze the number of generated subproblems. During this analysis, we consider the instance $(Q', \mathbf{a}', T, w, D)$ as the input given by the user for the PATH algorithm, that is, all recursive call occurred in the execution of PATH were caused by $\mathrm{PATH}(Q', \mathbf{a}', T, w, D)$. In addition, T has n nodes.

Notice that in a subproblem $\mathrm{OPT}(Q, \mathbf{a}, \overline{T}, w)$ the tree $\overline{T}$ can only take two forms: (i) it is a subtree T_u of the original tree T, for some node u (ii) it is a subtree of T_u obtained by removing all the maximal trees rooted at the last q children of u, for some number q and node u. Clearly there are at most n trees of form (i). Moreover, $\sigma(u)$ trees of form (ii) are generated for each node u of T, where $\sigma(u)$ is the number of children of u. As T is a tree, it follows that $\sum_{u \in T} \sigma(u) = n - 1$ and consequently there are $n - 1$ trees of form (ii). Hence, $n + n - 1 = O(n)$ trees are generated during the execution of PATH. For each generated subtree, PATH generates all possible path vectors $\mathbf{a}$. Since we have exactly 2^{i+2} possible vectors $\mathbf{a}$ corresponding to a path Q with i nodes, the number of generated vectors is given by $\sum_{i=0}^{D} 2^{i+2} = O(2^D)$. We remark that for sake of the subproblems, only the length of Q is important and not its specific constituent nodes. As a result, we have that PATH uses $O(n2^D)$ space.

Finally, we obtain the time complexity of PATH by counting the number of subproblems checked to calculate each value of $\mathrm{OPT}(Q, \mathbf{a}, \overline{T}, w)$. Let $\overline{r}$ be the root of $\overline{T}$. In the Case 1, PATH checks $O(D)$ subproblems, since we have $O(D)$ possible hotlinks from a node in Q to $\overline{r}$. On the other hand, in Case 2 the number of subproblems checked depends on the number of available hotlinks in both Q and $\overline{r}$. For j available hotlinks, $O(2^j)$ subproblems are checked, since this is the number of possible distributions for these hotlinks between two subproblems. Moreover, we have $O\left(n\binom{D+2}{j}\right)$ subproblems with j available hotlinks, for $j = 0, 1, \ldots, D+2$. Hence, using the Binomial Theorem we have that the time complexity of PATH is given by:

$$O(n2^D D) + \sum_{j=0}^{D+2} O\left(n\binom{D+2}{j}2^j\right) = O(n2^D D) + O(n(2+1)^{D+2}) = O(n3^D)$$

Theorem 1 *Consider an instance (T, w) of the 1-HAP, where T has n nodes. Given an integer D, the PATH algorithm finds in $O(n3^D)$ time an assignment A^* satisfying $EP(T, A^*, w) \leq EP(T, A, w)$ for any assignment A such that the height of T^A is at most D.*