On the Complexity of Searching in Trees: Average-case Minimization

Tobias Jacobs
Albert-Ludwigs-University Freiburg, Germany
jacobs@informatik.uni-freiburg.de

Ferdinando Cicalese
University of Salerno, Italy
cicalese@dia.unisa.it

Eduardo Laber
PUC-Rio, Brazil
laber@inf.puc-rio.br

Marco Molinaro
Carnegie Mellon, USA
molinaro@cmu.edu

Abstract

We study the following tree search problem: in a given tree $T = (V,E)$ a node has been marked and we want to identify it. In order to locate the marked node, we can use edge queries. An edge query $e$ asks in which of the two connected components of $T \setminus e$ the marked node lies. The worst-case scenario where one is interested in minimizing the maximum number of queries is well understood, and linear time algorithms are known for finding an optimal search strategy [Onak et al. FOCS’06, Mozes et al. SODA’08]. Here we study the more involved average-case analysis: A function $w : V \rightarrow \mathbb{Z}^+$ is given which defines the likelihood for a node to be the one marked, and we want the strategy that minimizes the expected number of queries. Prior to this paper, very little was known about this natural question and the complexity of the problem had remained so far an open question.

We close this question and prove that the above tree search problem is $\mathcal{NP}$-complete even for the class of trees with diameter at most 4. This results in a complete characterization of the complexity of the problem with respect to the diameter size. In fact, for diameter not larger than 3 the problem can be shown to be polynomially solvable using a dynamic programming approach.

In addition we prove that the problem is $\mathcal{NP}$-complete even for the class of trees of maximum degree at most 16. To the best of our knowledge, the only known result in this direction is that the tree search problem is solvable in $O(|V| \log |V|)$ time for trees with degree at most 2 (paths).

We match the above complexity results with a tight algorithmic analysis. We first show that a natural greedy algorithm attains a 2-approximation. Furthermore, for the bounded degree instances, we show that any optimal strategy (i.e., one that minimizes the expected number of queries) performs at most $O(\Delta(T)(\log |V| + \log w(T)))$ queries in the worst case, where $w(T)$ is the sum of the likelihoods of the nodes of $T$ and $\Delta(T)$ is the maximum degree of $T$. We combine this result with a non-trivial exponential time algorithm to provide an FPTAS for trees with bounded degree.
1 Introduction

Searching is one of the fundamental problems in Computer Science and Discrete Mathematics. In his classical book [20], D. Knuth discusses many variants of the searching problem, most of them dealing with totally ordered sets. There has been some effort to extend the available techniques for searching and for other fundamental problems (e.g. sorting and selection) to handle more complex structures such as partially ordered sets [25, 11, 29, 28, 8]. Here, we focus on searching in structures that lay between totally ordered sets and the most general posets. We wish to efficiently locate a particular node in a tree.

More formally, as input we are given a tree $T = (V, E)$ which has a ‘hidden’ marked node and a function $w : V \rightarrow \mathbb{Z}^+$ that gives the likelihood of a node being the one marked. In order to discover which node of $T$ is marked, we can perform edge queries: after querying the edge $e \in E$ we receive an answer stating in which of the two connected components of $T \setminus e$ the marked node lies. To simplify our notation let us assume that our input tree $T$ is rooted at a node $r$ so that we can specify a query to an edge $e = uv$, with $u$ being the parent of $v$, by referring to $v$.

A search strategy is a procedure that decides the next query to be posed based on the outcome of the previous queries. Every search strategy for a tree $T = (V, E)$ (or for a forest) can be represented by a binary search (decision) tree $D$ such that a path from the root of $D$ to a leaf $\ell$ indicates which queries should be made at each step to discover that $\ell$ is the marked node. More precisely, a search tree for $T$ is a triple $D = (N, E', A)$, where $N$ and $E'$ are the nodes and edges of a binary tree and the assignment $A : N \rightarrow V$ satisfies the following properties: (a) for every node $v$ of $V$ there is exactly one leaf $\ell$ in $D$ such that $A(\ell) = v$; (b)[search property] if $v$ is in the right (left) subtree of $u$ in $D$ then $A(v)$ is (not) in the subtree of $T$ rooted at $A(u)$. For an example we refer to Figure 1.

Given a search tree $D$ for $T$, let $d(u, v)$ be the length (in number of edges) of the path from $u$ to $v$ in $D$. Then the cost of $D$, or alternatively the expected number of queries of $D$ is given by

$$ cost(D) = \sum_{v \in \text{leaves}(D)} d(\text{root}(D), v)w(A(v)). $$

Therefore, our problem can be stated as follows: given a rooted tree $T = (V, E)$ with $|V| = n$ and a function $w : V \rightarrow \mathbb{Z}^+$, the goal is to compute a minimum cost search tree for $T$. This is a natural generalization of the problem of searching an element in a sorted list with non-uniform access probabilities.

The State of the Art. The variant of the problem in which the goal is to minimize the number of edge queries in the worst case, rather than minimizing the expected number of queries, has been studied in several recent papers [5, 29, 28]. It turns out that an optimal (worst-case) strategy can be found in linear time [28]. This is in great contrast with the state of the art (prior to this paper) about the average-case minimization we consider here. The known results amount to the $O(\log n)$-approximation obtained by Kosaraju et al. [21], and Adler and Heeringa [2] for the much more general binary identification problem, and the constant factor approximation algorithm that two of the authors gave in [23]. However, the complexity of the average-case minimization of the tree search problem has so far remained unknown.

Our Results. We significantly narrow the gap of knowledge in the complexity landscape of the tree search problem under two different points of view. We prove that this problem is $\mathcal{NP}$-Complete even for the class of trees with diameter at most 4. This results in a complete characterization of the problem’s complexity with respect to the parametrization in terms of the diameter. In fact, the problem can be shown to be polynomially solvable for the class of trees of diameter at most 3. We also show that the tree search problem under average minimization is $\mathcal{NP}$-Complete for trees of degree at most 16 (note that in any infinite class of trees either the diameter or the degree is non-constant). This substantially improves upon the state of the art, the only known result in this direction being an $O(n \log n)$ time solution [16] [14] for the class of trees with maximum degree 2. The hardness results are obtained by fairly involved reductions from the Exact 3-Set Cover (X3C) with multiplicity 3 [13].

In addition to the complexity results, we also significantly improve the previous known results from the algorithmic perspective. We first show that we can attain 2-approximation by a simple greedy approach
that always seeks to divide the remaining tree as evenly as possible. For bounded-degree trees, we match the new hardness results with an FPTAS. In order to obtain the FPTAS, we first devise a non-trivial Dynamic Programming based algorithm that, roughly speaking, computes the best possible search tree, among the search trees with height at most $H$, in $O(n^2 2^H)$ time. Then, we show that every tree $T$ admits a minimum cost search tree whose height is $O(\Delta \cdot (\log n + \log w(T)))$, where $\Delta$ is the maximum degree of $T$ and $w(T)$ is the total weight of the nodes in $T$. This bound is of independent interest because the height of any search tree for a complete tree of degree $\Delta$ is $\Omega(\frac{\Delta}{\log \Delta} \log n)$. Furthermore, it allows us to execute the DP algorithm with $H = c \cdot \Delta \cdot (\log n + \log w(T))$, for a suitable constant $c$, obtaining a pseudo-polynomial time algorithm for trees with bounded degree. By scaling the weights $w$ in a fairly standard way we obtain the FPTAS.

The worst-case scenario has also been studied for the case where a question is posed to some node $u$ and the answer is either that $u$ is the marked node or in which connected component of the forest $T \setminus \{u\}$ the marked node lies \cite{30, 29}. We remark that it is possible to adapt our techniques to prove that for the average-case minimization, this “node query”-variant of the tree search problem is also $\mathcal{NP}$-Complete; furthermore, we can provide for it a (degree independent) FPTAS. Due to the space constraints we have to defer these results to the full version of the paper.

**Other Related Work.** Besides the above mentioned papers, the worst-case version of searching in trees had already been studied and solved under a different name, one decade ago, as pointed out by Dereniowski \cite{10}. That is because the problem of searching a node in a tree is equivalent to the problem of ranking the edges of a tree \cite{19, 9, 25}.

The problem studied here can also be seen as a particular case of the binary identification problem (BIP) \cite{12}. Suppose we are given a set of elements $U = \{u_1, \ldots, u_n\}$, a set of tests $\{t_1, \ldots, t_m\}$, with $t_i \subseteq U$, a ‘hidden’ marked element and a likelihood function $w : U \mapsto \mathbb{R}^+$. A test $t$ allows to determine whether the marked element is in the set $t$ or in $U \setminus t$. The BIP consists of defining a strategy (decision tree) that minimizes the (expected) number of tests to find the marked element. Both the average-case and the worst-case minimization are $\mathcal{NP}$-Complete \cite{17}, and none of them admits an $O(\log n)$-approximation unless $\mathcal{P} = \mathcal{NP}$ \cite{24, 7}. For both versions, simple greedy algorithms attain $O(\log n)$-approximation \cite{21, 4, 2}. When we impose some structure in the set of tests we have interesting particular cases. If the set of tests consists of all the subsets of $U$ (i.e., $2^U$), then the strategy that minimizes the average cost is a Huffman tree. Let $G$ be a DAG with vertex set $U$. If the set of tests is $\{t_1, \ldots, t_m\}$, where $t_i = \{u_j | u_i \prec u_j \text{ in } G\}$, then we have the problem of searching in a poset \cite{27, 21, 6}. When $G$ is a directed path we have the alphabetic coding problem \cite{16}. The problem we study here corresponds to the particular case where $G$ is a directed tree.

**Applications.** The problem of searching in posets (and in particular in trees) has practical applications in file system synchronization and software testing according to \cite{3, 28}.

Strategies for searching in trees have also potential application to asymmetric communication protocols \cite{11, 8, 15, 22, 31}. In this scenario, a client has to send a binary string $x \in \{0, 1\}^t$ to the server. $x$ is drawn from a probability distribution $\mathcal{D}$ only available to the server. The asymmetry comes from the client having much larger bandwidth for downloading than for uploading. In order to benefit from this discrepancy, both parties agree on a protocol to exchange bits until the server learns the string $x$, trying to minimize the number of bits sent by the client (though other factors, e.g., the number of rounds should also be taken into account). In one of the first protocols \cite{3, 22}, at each round the server sends a binary string $y$ and the client replies with a 0 or 1 depending on whether $y$ is a prefix of $x$ or not. Based on the client’s answer, the server updates his knowledge about $x$ and sends another string if he has not learned $x$ yet. This protocol corresponds to a strategy for searching a marked leaf in a complete binary tree of height $t$, where only the leaves have non-zero probability. In fact, the binary strings in $\{0, 1\}^t$ can be represented by a complete binary tree of height $t$ where every edge that connects a node to its left (right) child is labeled with 0 (1). This gives a 1-1 correspondence between binary strings of length at most $t$ and edges of the tree, and the message $y$ sent by the server naturally corresponds to an edge query.
2 hardness

In this section we shall prove that the tree search problem defined above is \(NP\)-Complete. We shall use a reduction from the Exact 3-Set Cover problem with multiplicity bounded by 3, i.e., each element of the ground set can appear in at most 3 sets.

An instance of the 3-bounded Exact 3-Set Cover problem (X3C) is defined by: (a) a set \(U = \{u_1, \ldots, u_n\}\), with \(n = 3k\) for some \(k \geq 1\); (b) a family \(\mathcal{X} = \{X_1, \ldots, X_m\}\) of subsets of \(U\), such that \(|X_i| = 3\) for each \(i = 1, \ldots, m\) and for each \(i = 1, \ldots, n\), we have that \(u_i\) appears in at most 3 sets of \(\mathcal{X}\). Given an instance \(\mathcal{I} = (U, \mathcal{X})\) the X3C problem is to decide whether \(\mathcal{X}\) contains a partition of \(U\), i.e., whether there exists a family \(\mathcal{C} \subseteq \mathcal{X}\) such that \(|\mathcal{C}| = k\) and \(\bigcup_{X \in \mathcal{C}} X = U\). This problem is well known to be \(NP\)-Complete [13].

For our reduction it will be crucial to define an order among the sets of the family \(X\). Any total order \(<\) on \(U\), say \(u_1 < u_2 < \cdots < u_n\), can be extended to a total order \(<\) on \(\mathcal{X} \cup U\) by stipulating that:

(a) for any \(X = \{x_1, x_2, x_3\}\), \(Y = \{y_1, y_2, y_3\} \in \mathcal{X}\) (with \(x_1 < x_2 < x_3\) and \(y_1 < y_2 < y_3\)) the relation \(X < Y\) holds if and only if the sequence \(x_3 x_2 x_1\) is lexicographically smaller than \(y_3 y_2 y_1\); (b) for every \(j = 1, \ldots, n\), the relation \(u_j < X\) holds if and only if the sequence \(u_j u_1 u_3\) is lexicographically smaller than \(x_3 x_2 x_1\).

Assume an order \(<\) on \(U\) has been fixed and \(<\) is its extension to \(U \cup \mathcal{X}\), as defined above. We denote by \(\Pi = (\pi_1, \ldots, \pi_{n+m})\) the sequence of elements of \(U \cup \mathcal{X}\) sorted in increasing order according to \(<\). From now on, w.l.o.g., we assume that according to \(<\) and \(<\), it holds that \(u_1 < \cdots < u_n\) and \(X_1 < \cdots < X_m\).

For each \(i = 1, \ldots, m\), we shall denote the elements of \(X_i\) by \(u_{i1}, u_{i2}, u_{i3}\) so that \(u_{i1} < u_{i2} < u_{i3}\).

Example 1. Let \(U = \{a, b, c, d, e, f\}\), and \(\mathcal{X} = \{\{a, b, c\}, \{b, c, d\}, \{d, e, f\}, \{b, e, f\}\}\). Then, fixing the standard alphabetical order among the elements of \(U\), we have that the sets of \(\mathcal{X}\) are ordered as follows: \(X_1 = \{a, b, c\}\), \(X_2 = \{b, c, d\}\), \(X_3 = \{b, e, f\}\), \(X_4 = \{d, e, f\}\). Then, we have \(\Pi = (\pi_1, \ldots, \pi_{10}) = (a, b, c, X_1, d, X_2, e, f, X_3, X_4)\).

Because of the orders we fixed and the fact that each element of \(U\) appears in at most 3 sets of \(\mathcal{X}\), it follows that that we cannot have more than three sets of \(\mathcal{X}\) appearing consecutively in \(\Pi\). This will be important to prove the hardness for bounded degree instances.

We shall first show a polynomial time reduction that maps any instance \(\mathcal{I} = (U, \mathcal{X})\) of 3-bounded X3C to an instance \(\mathcal{I} = (T, w)\) of the tree search problem, such that \(T\) has diameter 4 but unbounded degree. We will then modify such reduction and show hardness for the bounded case too.

The structure of the tree \(T\). The root of \(T\) is denoted by \(r\). For each \(i = 1, \ldots, m\) the set \(X_i \in \mathcal{X}\) is mapped to a tree \(T_i\) of height 1, with root \(r_i\) and leaves \(t_{i1}, t_{i2}, t_{i3}\). In particular, for \(j = 1, 2, 3\), we say that \(s_{ij}\) is associated with the element \(u_{ij}\). We make each \(r_i\) a child of \(r\). For \(i = 1, \ldots, m\), we also create four leaves \(a_{i1}, a_{i2}, a_{i3}, a_{i4}\) and make them children of the root \(r\). We also define \(\hat{X}_i = \{t_{i1}, t_{i2}, s_{i2}, a_{i1}, \ldots, a_{i4}\}\) to be the set of leaves of \(T\) associated with \(X_i\). For the example given above, the corresponding tree is given in Figure 2.

The weights of the nodes of \(T\). Only the leaves of \(T\) will have non-zero weight, i.e., we set \(w(r) = w(r_1) = \cdots = w(r_m) = 0\). While defining the weight of the leaves of \(T\) it will be useful to assign weight also to each \(u \in U\). In particular, our weight assignment will be such that each leaf in \(T\) which is associated with an element \(u\) will be assigned the same weight we assign to \(u\). Also, when we fix the weight of \(u\) we shall understand that we are fixing the weight of all leaves in \(T\) associated with \(u\). We extend the function \(w()\) to sets, so the weight of a set is the total weight of its elements. Also we define the weight of a tree as the total weight of its nodes.

The weights will be set in order to force any optimal search tree for \((T, w)\) to have a well-defined structure. The following notions of Configuration and Realization will be useful to describe such a structure of an optimal search tree. In describing the search tree we shall use \(q_r\) to denote the node in the search tree under consideration that represents the question about the node \(\nu\) of the input tree \(T\). Moreover, we shall in general only be concerned with the part of the search tree meant to identify
the nodes of $T$ of non-zero weight. It should be clear that the search tree can be easily completed by appending the remaining queries at the bottom.

**Definition 1.** Given leaves $\ell_1, \ldots, \ell_h$ of $T$, a sequential search tree for $\ell_1, \ldots, \ell_h$ is a search tree of height $h$ whose left path is $q_{\ell_1}, \ldots, q_{\ell_h}$. This is the strategy that asks about one leaf after another until they have all been considered. See Figure 3 (a) for an example.

**Configurations, and Realizations of $\Pi$.** For each $i = 1, \ldots, m$, let $D^A_i$ be the search tree with root $q_{s_i}$, with right subtree being the sequential search tree for $t_i, s_{i+1}, s_{i+2}, s_{i+1}$, and left subtree being a sequential search tree for (some permutation of) $a_1, \ldots a_4$. We also refer to $D^A_i$ as the $A$-configuration for $X_i$.

Moreover, let $D^B_i$ be the search tree with root $q_{s_i}$ and left subtree being a sequential search tree for (some permutation of) $a_1, \ldots a_4$. We say that $D^B_i$ is the $B$-configuration for $X_i$. See Figure 3 (b)-(c).

**Definition 2.** Given two search trees $T_1, T_2$, the extension of $T_1$ with $T_2$ is the search tree obtained by appending the root of $T_2$ to the leftmost leaf of $T_1$. The extension of $T_1$ with $T_2$ is a new search tree that “acts” like $T_1$ and in case of all NO answers continues following the strategy represented by $T_2$.

**Definition 3.** A realization (of $\Pi$) with respect to $\mathcal{Y} \subseteq \mathcal{X}$ is a search tree for $(T, w)$ defined recursively as follows: $^1$ For each $i = 1, \ldots, n + m$, a realization of $\pi_1 \pi_{i+1} \ldots \pi_{n+m}$ is an extension of the realization of $\pi_{i+1} \ldots \pi_{n+m}$ with another tree $T'$ chosen according to the following two cases:

Case 1. If $\pi_i = u_j$, for some $j = 1, \ldots, n$, then $T'$ is a (possibly empty) sequential search tree for the leaves of $T$ that are associated with $u_j$ and are not queried in the realization of $\pi_{i+1} \ldots \pi_{n+m}$.

Case 2. If $\pi_i = X_j$, for some $j = 1, \ldots, m$, then $T'$ is either $D^B_j$ or $D^A_j$ according as $X_j \in \mathcal{Y}$ or not.

We denote by $D^A_i$ the realization of $\Pi$ w.r.t. the empty family, i.e., $\mathcal{Y} = \emptyset$. Figure 4 shows some of the realizations for the Example 1 above.

We are going to set the weights in such a way that every optimal solution is a realization of $\Pi$ w.r.t. some $\mathcal{Y} \subseteq \mathcal{X}$ (our Lemma $^2$). Moreover, such weights will allow to discriminate between the cost of solutions that are realizations w.r.t. to an exact cover for the X3C instance and the cost of any other solutions that are realizations w.r.t. to an exact cover for $\mathcal{X}$. Let $D^*$ be an optimal search tree and $\mathcal{Y}$ be such that $D^*$ is a realization of $\Pi$ w.r.t. $\mathcal{Y}$. In addition, for each $u \in \mathcal{Y}$ define $W_u = \sum_{\ell : X_i \prec u} w(X_\ell)$. It is not hard to see that the difference between the cost of $D^A_i$ and $D^*$ can be expressed as follows:

$$
cost(D^A_i) - cost(D^*) = \sum_{X_i \in \mathcal{Y}} \left( w(t_i) - (W_{u_{i+1}} + W_{u_{i+2}} + W_{u_{i+3}}) - \sum_{j=1}^3 d_{B}^i(q_{s_{ij}})w(u_{ij}) \right),
$$

where $d_{B}^i(q_{s_{ij}})$ is the difference between the level of the node $q_{s_{ij}}$ in $D^*$ and the level $q_{s_{ij}}$ in a realization of $\Pi$ w.r.t. $\mathcal{Y} \setminus \{X_i\}$. To see this, imagine to turn $D^A_i$ into $D^*$ one step at a time. Each step being the changing of configuration from $A$ to $B$ for a set of leaves $X_i$ such that $X_i \in \mathcal{Y}$. Such a step implies: (a) moving the question $q_{s_{ij}}$ exactly $d_{B}^i(q_{s_{ij}})$ levels down, so increasing the cost by $d_{B}^i(q_{s_{ij}})w(u_{ij})$; (b) because of (a) all the questions that were below the level where $q_{s_{ij}}$ is moved, are also moved down one level. This additional increase in cost is accounted for by the $W_{u_{i+1}}$’s; (c) moving one level up the question about $t_i$, so gaining cost $w(t_i)$.

We will define the weight of $t_i$ in order to: compensate the increase in cost (a)-(b) due to the relocation of $q_{s_{ij}}$; and to provide some additional gain only when $\mathcal{Y}$ is an exact cover. In general, the value of $d_{B}^i(q_{s_{ij}})$ depends on the structure of the realization for $\mathcal{Y} \setminus \{X_i\}$; in particular, on the length of the sequential search trees for the leaves associated to $u_k$’s, that appear in $\Pi$ between $X_i$ and $u_j$. However, when $\mathcal{Y}$ is an exact cover, each such sequential search tree has length one. A moment’s reflection shows that in this case $d_{B}^i(q_{s_{ij}}) = \gamma(i, j)$, where, for each $i = 1, \ldots, m$ and $j = 1, 2, 3$, we define

$$
\gamma(i, j) = j - 5 + |\{u_k : u_{ij} < u_k \leq X_i\}| + 5 \cdot |\{X_k : u_{ij} < X_k \leq X_i\}|
$$

$^1$For sake of definiteness we set $\pi_{m+n+1} = \emptyset$ and the realization of $\pi_{n+m+1}$ w.r.t. $\mathcal{Y}$ to be the empty tree.

$^2$The existence of such a $\mathcal{Y}$ will be guaranteed by Lemma $^1$. 

4
To see this, assume that \( \mathcal{Y} \) is an exact cover. Let \( D' \) be the realization for \( \mathcal{Y} \setminus X_i \), and \( \ell \) be the level of the root of the A-configuration for \( X_i \) in \( D' \). The node \( q_{s_{ij}} \) is at level \( \ell + (5 - j) \) in \( D' \). In \( D^* \), the root of the B-configuration for \( X_i \) is also at level \( \ell \). Also, in \( D^* \), between level \( \ell \) and the level of \( q_{s_{ij}} \), there are only nodes associated with elements of some \( \pi_k \) s.t. \( u_{ij} \prec \pi_k \preceq X_j \). Precisely, there is 1 level per each \( u_k \) s.t. \( u_{ij} \prec \pi_k \preceq X_j \) (corresponding to the sequential search tree for the only leaf associated with \( u_k \)); and 5 levels per each \( u_k \) s.t. \( u_{ij} \prec \pi_k \preceq X_j \) (corresponding to the left path of the A or B-configuration for \( X_k \)). In total, the difference between the levels of \( q_{s_{ij}} \) in \( D' \) and \( D^* \) is exactly \( \gamma(i, j) \).

Note that \( \gamma(i, j) \) is still well defined even if there is not an exact cover \( \mathcal{Y} \subseteq \mathcal{X} \). This quantity will be used to define \( w(t_i) \).

We are now ready to provide the precise definition of the weight function \( w \). We start with \( w(u_1) = 1 \). Then, we fix the remaining weights inductively, using the sequence \( \Pi \) in the following way: let \( i > 1 \) and assume that for each \( i' \), the weights of all leaves associated with \( \pi_{i'} \) have been fixed\(^3\). We now proceed according to the following two cases:

**Case 1.** \( \pi_i = u_j \), for some \( j \in \{1, \ldots, n\} \). Then, we set \( w(u_j) = 1 + 6 \max\{|T| w(u_{j-1}), W_{u_j}\} \), where \(|T|\) denotes the number of nodes of \( T \).

**Case 2.** \( \pi_i = X_j \), for some \( j \in \{1, \ldots, m\} \). Note that in this case the weights of the leaves \( s_{j1}, s_{j2}, s_{j3} \) have already been fixed, respectively to \( w(u_{j1}), w(u_{j2}), \) and \( w(u_{j3}) \). This is because we fix the weights following the sequence \( \Pi \) and we have \( u_{j1} \prec u_{j2} \prec u_{j3} \prec X_j \). In order to define the weights of the remaining elements in \( X_j \) we set \( w(a_{j1}) = \cdots = w(a_{j4}) = W_{u_{j1}} + W_{u_{j2}} + W_{u_{j3}} + \sum_{k=1}^3 \gamma(j, k)w(u_{j k}) \). Finally, we set \( w(t_j) = w(a_{j1}) + w(X_j)/2 \).

**Remark 1.** For each \( i = 1, \ldots, n + m \), let \( w(\pi_i) \) denote the total weight of the leaves associated with \( \pi_i \). It is not hard to see that \( w(\pi_i) = O(|T|^3) \). Therefore we have that the maximum weight is not larger than \( w(\pi_{m+n}) = O(|T|^{3(n+m-n)}) \). It follows that we can encode all the weights using \( O(3|T|(n+m) \log |T|) \) bits, hence the size of the instance \( (T, w) \) is polynomial in the size of the \( X3C \) instance \( \mathbb{I} = (U, \mathcal{X}) \).

Since \( t_m \) is the heaviest leaf, one can show that in an optimal search tree \( D^* \) the root can only be \( q_{t_m} \) or \( q_{r_m} \). For otherwise moving one of these questions closer to the root of \( D^* \) results in a tree with smaller cost, violating the optimality of \( D^* \). Moreover, by a similar “exchange” argument it follows that if \( q_{r_m} \) is the root of \( D^* \) then the right subtree must coincide with a sequential search tree for \( t_m, s_{m1}, s_{m2}, s_{m3} \) and the left subtree of \( q_{r_m} \) must be a sequential tree for \( a_{m1}, \ldots, a_{m4} \). Therefore the top levels of \( D^* \) coincide either with \( D^A_m \) or with \( D^B_m \), or equivalently they are a realization of \( \pi_{m+n} \). Repeating the same argument on the remaining part of \( D^* \) we have the following (the complete proof is in appendix):

**Lemma 1.** Any optimal search tree for the instance \( (T, w) \) is a realization of \( \Pi \) w.r.t. some \( \mathcal{Y} \subseteq \mathcal{X} \).

Recall now the definition of the search tree \( D^A \). Let \( D^* \) be an optimal search tree for \( (T, w) \). Let \( \mathcal{Y} \subseteq \mathcal{X} \) be such that \( D^* \) is a realization of \( \Pi \) w.r.t. \( \mathcal{Y} \). Equation [1] and the definition of \( w(t_i) \) yield

\[
\text{cost}(D^A) - \text{cost}(D^*) = \sum_{X_i \in \mathcal{Y}} \left( \frac{w(X_i)}{2} + \sum_{j=1}^{3} (\gamma(i, j) - d^A_{\phi}(q_{s_{ij}})) w(u_{ij}) \right) = \sum_{j=1}^{n} \sum_{X_i \in \mathcal{Y}} \left( \frac{w(u_{ij})}{2} + \Gamma(i, j) w(u_{ij}) \right),
\]

where \( \Gamma(i, j) = \gamma(i, \kappa) - d^A_{\phi}(q_{s_{i\kappa}}) \), and \( \kappa \in \{1, 2, 3\} \) is such that \( s_{i\kappa} = u_j \).

By definition, if for each \( j = 1, \ldots, n \), there exists exactly one \( X_i \in \mathcal{Y} \) such that \( u_j \in X_i \), then we have \( \Gamma(i, j) = 0 \). Therefore, equation [2] evaluates exactly to \( \sum_{j=1}^{n} \frac{w(u_{ij})}{2} \). Conversely, we can prove that this never happens when for some \( 1 \leq j \leq n \), \( u_j \) appears in none or in more than one of the sets in \( \mathcal{Y} \). For this we use the exponential (in \( |T| \)) growth of the weights \( w(u_{ij}) \) and the fact that in such case the

\(^3\)By the leaves associated with \( \pi_{i'} \) we mean the leaves in \( \tilde{X}_j \), if \( \pi_i = X_j \) for some \( X_j \in \mathcal{X} \), or the leaves associated with \( u \) if \( \pi_{i'} = u \) for some \( u \in U \).
Lemma 2. Let $D^*$ be an optimal search tree for $(T, w)$. Let $\mathcal{Y} \subseteq \mathcal{X}$ be such that $D^*$ is a realization of $\Pi$ w.r.t. $\mathcal{Y}$. We have that $\text{cost}(D^*) \leq \text{cost}(D^A) - \frac{1}{2} \sum_{u \in U} w(u)$ if and only if $\mathcal{Y}$ is a solution for the $X3C$ instance $\Pi = (U, \mathcal{X})$.

The $NP$-Completeness of 3-bounded X3C [13], Remark [1] and Lemma [2] imply the following.

Theorem 1. The search tree problem is $NP$-Complete in the class of trees of diameter at most 4.

Note that this result is tight. In fact, for trees of diameter at most 3 the problem is polynomially solvable, e.g., via dynamic programming (see Appendix).

$NP$-Completeness for bounded-degree instances. We can adapt our proof to show that the search tree problem is $NP$-Complete also for bounded-degree trees. For that, we modify the input tree as follows. We partition the subsets of $\mathcal{X}$ so that sets that are adjacent in $\Pi$ are put together. For the instance in the Example 1 the corresponding partition would be $\{X_1, X_2, X_3, X_4\}$.

Let $\mathcal{Z} = \{\mathcal{Z}_1, \ldots, \mathcal{Z}_p\}$ be the partition obtained from the input instance $(U, \mathcal{X})$. Recall the definitions of the subtrees $T_j$ and the leaves $a_{j1}, \ldots, a_{j4}$ ($j = 1, \ldots, m$) given for the construction of the tree $T$. We now create a new tree $T^b$ as follows. For each $i = 1, \ldots, p$ in $T^b$ there is a subtree $H_i$ that corresponds to the element $\mathcal{Z}_i \in \mathcal{Z}$. $H_i$ has root $h_i$. For each $j$ such that $X_j \in \mathcal{Z}_i$ we make the root of $T_j$, i.e., $r_j$, and the leaves $a_{j1}, \ldots, a_{j4}$ children of $h_i$. Finally, we create nodes $z_1, \ldots, z_p$ and make $h_1$ a child of $z_1$ and for $i = 2, \ldots, p$ we make $z_{i-1}$ and $h_i$ children of $z_i$. See Fig. 5 for the tree $T^b$ corresponding to the instance in Example 1.

The fact that in $\Pi$ there are no more than three elements of $\mathcal{X}$ which appear consecutively, implies that any $\mathcal{Z}_i$ contains at most three elements. This gives that the maximum degree in $T^b$ is at most 16.

Regarding the weight function, we extend to $T^b$ the weight function defined for the tree $T$ by setting $w(h_i) = w(z_i) = 0$, for each $i = 1, \ldots, p$ and leaving the other weights as before.

It turns out that Lemma [1] still holds for the new instance $(T^b, w)$. In fact, in each subtree $H_i$ the structure of the instance is exactly the same as in the tree $T$, so one can prove that any optimal solution for such subinstance is a realization of the corresponding subsequence of $\Pi$. Moreover, because of the way we partitioned $\mathcal{X}$, and the weight function $w$, it follows that the smallest weight of an $a_{jk}$ in $\mathcal{Z}_i$ is bigger than the total weight of the leaves in $\mathcal{Z}_1, \ldots, \mathcal{Z}_{i-1}$. This is enough to enforce the order of a realization of $\Pi$, i.e., that the leaves $t_j, a_{j1}, \ldots, a_{j4}$ are queried before the leaves in $\mathcal{Z}_1, \ldots, \mathcal{Z}_{i-1}$. We have proved the following (a formal proof is in the appendix).

Lemma 3. Any optimal search tree for the instance $(T^b, w)$ is a realization of $\Pi$ w.r.t. some $\mathcal{Y} \subseteq \mathcal{X}$.

By using this lemma together with Lemma [2] we have that Theorem [1] holds also for bounded-degree instances of the tree search problem.

3 Approximation Algorithms

We need to introduce some notation. For any forest $F$ of rooted trees and node $j \in F$, we denote by $F_j$ the subtree of $F$ composed by $j$ and all of its descendants. We denote the root of a tree $T$ by $r(T)$, $\delta(u)$ denotes the number of children of $u$ and $c_i(u)$ is used to denote the $i$th child of $u$ according to some arbitrarily fixed order. The following operation will be useful for modifying search trees: Given a search tree $D$ and a node $u \in D$, a left deletion of $u$ is the operation that transforms $D$ into a new search tree by removing both $u$ and its left subtree from $D$ and, then, by connecting the right subtree of $u$ to the parent of $u$ (if it exists). A right deletion is analogously defined.

Given a search tree $D$ for $T$, we use $l_u$ to denote the leaf of $D$ assigned to node $u$ of $T$. 
3.1 The natural greedy algorithm attains 2-approximation

Consider a search tree $D$ for $T$. Notice that when we follow a path from the root of $D$ to one of its leaves, we reduce the search space (eliminate part of $T$) whenever we visit a new node. Therefore, we can associate with each node of $D$ the subtree of $T$ which may still contain the node we search for. Notice that the tree $T'$ associated with node $v \in D$ is exactly the one induced by the nodes of $T$ that correspond to the leaves of $D_v$, hence $w(T') = w(D_v)$. E.g., in Fig. 1 the node $< f >$ in $D$ is associated with $T_f$.

We can transform a search tree $D$ for $T$ into a search tree $D'$ for an arbitrary subtree $T'$ of $T$. This search tree $D'$ is computed by taking each node $v \in D$ assigned to a node $A(v)$ in $T - T'$ and applying a left deletion if $A(v)$ is an ancestor of $r(T')$ or a right deletion otherwise. The important property of this construction is that the path $r(D') \sim l_x$, for every $x \in T'$, is exactly the subpath obtained by removing all queries to nodes in $T - T'$ from $r(D) \sim l_x$. The next lemma formalizes this discussion:

**Lemma 4.** Consider a tree $T$ and a search tree $D$ for it. Let $T'$ be a subtree of $T$. Then there is a search tree $D'$ for $T'$ such that $d(r(D'), l_x) = d(r(D), l_x) - n_x$, where $n_x$ is the number of nodes in the path $r(D) \sim l_x$ assigned to nodes in $T - T'$.

We show that the natural greedy algorithm guarantees an approximation factor of 2. The algorithm can be formulated in two sentences. (1) Let $x$ be a node such that $|w(T_x) - w(T \setminus T_x)|$ is minimized. Set $A(r(D)) = x$. (2) Construct the right and left subtree of $D$ by recursively applying the algorithm to $T_x$ and $T \setminus T_x$, respectively.

In order to prove that this algorithm results in a 2-approximation, we show that any search tree $D^*$ can be turned into the greedy search tree $D$ while the cost increases by at most $\text{cost}(D^*)$.

The proof is by induction on the number of nodes $n$ of the input tree $T$. For the basic case $n = 1$ there is nothing to show. Assume that the claim holds for any tree with at most $n - 1$ nodes. In order to prove it true for $T$ we proceed in two steps.

Let $x$ be the node queried at the root of $D$. Also let $D^*_0$ (resp. $D_0$) and $D^*_1$ (resp. $D_1$) be the search tree for $T$ and $T \setminus T_x$ obtained from $D^*$ (resp. $D$) via Lemma 4. (a) Construct a search tree $D'$ with $A(r(D')) = x$ and the left and right subtree being $D^*_1$ and $D^*_0$ respectively. It is not hard to see that $D'$ is a legal search tree. (b) Use the induction hypothesis for turning $D^*_0$ and $D^*_1$ into $D_0$ and $D_1$ respectively. It is straightforward to see that the transformation results in the tree $D$.

**Lemma 5.** We have $\text{cost}(D') \leq \text{cost}(D^*) + w(T)/2$.

*Proof sketch.* Let $x$ and $x^*$ be the nodes queried at the root of $D'$ and $D^*$, respectively. W.l.o.g. we assume $x \neq x^*$, as otherwise the lemma trivially holds. We can also assume that $x^*$ is a node from $T_x$, because the opposite case is analyzed analogously.

We shall first analyze the case $w(T_x) \leq w(T - T_x)$, i.e., $w(T_x) \leq w(T)/2$. As any path from $r(D^*)$ to a leaf in $D^*$ contains $r(D^*)$ and $T - T_x$ does not contain $x^*$, Lemma 4 states that the depth of any leaf in $D^*_1$ is at least by one smaller than it is in $D^*$. The lemma also implies that the depth of any leaf in $D^*_0$ is not greater than it is in $D^*$. So we have

\[
\text{cost}(D') = w(T) + \text{cost}(D^*_0) + \text{cost}(D^*_1) \leq w(T) + \text{cost}(D^*) - w(T - T_x) \leq \text{cost}(D^*) + w(T)/2.
\]

The case $w(T_x) > w(T - T_x)$ requires a more involved analysis and we defer it to the appendix due to the space limitations.

It follows that the cost of $D$ can be bounded from above by

\[
\text{cost}(D) = w(T) + \text{cost}(D_0) + \text{cost}(D_1) \leq w(T) + 2\text{cost}(D^*_0) + 2\text{cost}(D^*_1) = 2\text{cost}(D') - w(T) \leq 2\text{cost}(D^*).
\]

The first inequality follows from the induction hypothesis and the second one is due to Lemma 5.

We have proven the following result.

**Theorem 2.** The greedy strategy is a polynomial 2-approximation algorithm for the tree search problem.
3.2 An FPTAS for Searching in Bounded-Degree Trees

We now present an FPTAS for searching in trees with bounded degree. First, we devise a dynamic programming algorithm whose running time is exponential in the height of optimal search trees. Then we essentially argue that the height of optimal search trees is \( O(\Delta(T) \cdot (\log w(T) + \log n)) \), thus the previous algorithm has a pseudo-polynomial running time. Finally, we employ a standard scaling technique to obtain an FPTAS.

We often construct a search trees starting with its ‘left part’. In order to formally describe such constructions, we define a left path as an ordered path where every node has only a left child. In addition, the left path of an ordered tree \( T \) is defined as the ordered path we obtain when we traverse \( T \) by only going to the left child, until we reach a node which does not have a left child.

A dynamic programming algorithm. In order to find an optimal search tree in an efficient way, we need to define a family of auxiliary problems denoted by \( \mathcal{P}^B(F,P) \). In the following paragraphs we describe the essential structures needed in these subproblems and then we show how to use the subproblems to find an optimal search tree.

First we introduce the concept of an extended search tree, which is basically a search tree with some extra nodes that have not been associated with a query yet (unassigned nodes) and some other nodes that cannot be associated with a query (blocked nodes).

**Definition 4.** An extended search tree (EST) for a forest \( F = (V,E) \) is a triple \( D = (N,E',A) \), where \( N \) and \( E' \) are the nodes and edges of an ordered binary tree and the assignment \( A : N \rightarrow V \cup \{ \text{blocked, unassigned} \} \) simultaneously satisfy the following properties:

(a) For every node \( v \) of \( F \), \( D \) contains both a leaf \( \ell \) and an internal node \( u \) such that \( A(\ell) = A(u) = v \);

(b) \( \forall u, v \in D \), with \( A(u), A(v) \in F \), the following holds: If \( v \) is in the right subtree of \( u \) then \( A(v) \in F_{A(u)} \). If \( v \) is in left subtree of \( u \) then \( A(v) \notin F_{A(u)} \);

(c) If \( u \) is a node in \( D \) with \( A(u) \in \{ \text{blocked, unassigned} \} \), then \( u \) does not have a right child.

If we drop (c) and also the requirement regarding internal nodes in (a) we have the definition of a search tree for \( F \). The cost of an EST \( D \) for \( F \) is analogous to the cost of a search tree and is given by \( \text{cost}(D) = \sum d(r(D), u)w(A(u)) \), where the summation is taken over all leaves \( u \in D \) for which \( A(u) \in F \).

At this point we establish a correspondence between optimal EST’s and optimal search trees. Given an EST \( D \) for a tree \( T \), we can apply a left deletion to the internal node of \( D \) assigned to \( r(T) \) and right deletions to all nodes of \( D \) that are blocked or unassigned, getting a search tree \( D' \) of cost \( \text{cost}(D') \leq \text{cost}(D) - w(r(T)) \). Conversely, we can add a node assigned to \( r(T) \) to a search tree \( D' \) and get an EST \( D \) such that \( \text{cost}(D) \leq \text{cost}(D') + w(r(T)) \). Employing these observations we can prove the following lemma:

**Lemma 6.** Any optimal EST for a tree \( T \) can be converted into an optimal search tree for \( T \) (in linear time). In addition, the existence of an optimal search tree of height \( h \) implies the existence of an optimal EST of height \( h + 1 \).

So we can focus on obtaining optimal EST’s. First, we introduce concepts which serve as a building blocks for EST’s. A partial left path (PLP) is a left path where every node is assigned (via a function \( A \)) to either blocked or unassigned. Now consider an EST \( D \) and let \( L = \{l_1, \ldots, l_{|L|}\} \) be its left path. We say that \( D \) is compatible with a PLP \( P = \{p_1, \ldots, p_{|P|}\} \) if \( |P| = |L| \) and \( A(p_i) = \text{blocked} \) implies \( A(l_i) = \text{blocked} \). The tree in Figure 7(c) is compatible with the path of Figure 7(b).

This definition of compatibility implies a natural one to one correspondence between nodes of \( L \) and \( P \). Therefore, without ambiguity, we can use \( p_i \) when referring to node \( l_i \) and vice versa.

Now we can introduce our subproblem \( \mathcal{P}^B \). First, fix a tree \( T \) with \( n \) nodes and a weight function \( w \). Given a forest \( F = \{T_{e_1(u)}, T_{e_2(u)} \ldots, T_{e_f(u)}\} \), a PLP \( P \) and an integer \( B \), the problem \( \mathcal{P}^B(F,P) \) consists
of finding an EST for $F$ with minimum cost among those EST’s for $F$ that are compatible with $P$ and have height at most $B$. We shall note that $F$ is not a general subforest of $T$, but one consisting of subtrees rooted at the first $f$ children of some node $u \in T$, for some $1 \leq f \leq \delta(u)$.

Notice that if $P$ is a PLP where all nodes are unassigned and $P$ and $B$ are sufficiently large, then $\mathcal{P}^B(T, P)$ gives an optimal EST for $T$.

**Algorithm for $\mathcal{P}^B(F, P)$.** We have a base case and also two other cases depending on the structure of $F$. In all these cases, although not explicitly stated, if $P$ does not contain unassigned nodes then the algorithm returns ‘not feasible’. If during its execution the algorithm encounters a ‘not feasible’ subproblem it ignores this choice in the enumeration.

**Base case:** $F$ has only one node $u$. In this case, the optimal solution for $\mathcal{P}^B(F, P)$ is obtained from $P$ by assigning its first unassigned node, say $p_i$, to $u$ and then adding a leaf assigned to $u$ as a right child of $p_i$. Its cost is $i \cdot w(u)$.

**Case 1:** $F$ is a forest $\{T_{c_1(u)}, \ldots, T_{c_f(u)}\}$. The idea of the algorithm is to decompose the problem into subproblems for the forests $T_{c_f(u)}$ and $F \setminus T_{c_f(u)}$. For that, it needs to select which nodes of $P$ will be assigned to each of these forests.

The algorithm considers all possible bipartitions of the unassigned nodes of $P$ and for each bipartition $U = (U^f, U^o)$ it computes an EST $D^{df}$ for $F$ compatible with $P$. At the end, the algorithm returns the tree $D^{df}$ with smallest cost. The EST $D^{df}$ is constructed as follows:

1. Let $P^f$ be the PLP constructed by starting with $P$ and then setting all the nodes in $U^o$ as blocked (Figure 6.b). Similarly, let $P^o$ be the PLP constructed by starting with $P$ and setting all nodes in $U^f$ as blocked. Let $D^f$ and $D^o$ be optimal solutions for $\mathcal{P}^B(T_{c_f(u)}, P^f)$ and $\mathcal{P}^B(F \setminus T_{c_f(u)}, P^o)$, respectively (Figure 6.c).

2. The EST $D^{df}$ is computed by taking the ‘union’ of $D^f$ and $D^o$ (Figure 6.d). More formally, the ‘union’ operation consists of starting with the path $P$ and then replacing: (i) every node in $P \cap U^f$ by the corresponding node in the left path of $D^f$ and its right subtree; (ii) every node in $P \cap U^o$ by the corresponding node in the left path of $D^o$ and its right subtree.

Notice that the height of every EST $D^{df}$ is at most $B$; this implies that the algorithm returns a feasible solution for $\mathcal{P}^B(F, P)$. Also, the cost of $D^{df}$ is given by $OPT(\mathcal{P}^B(T_{c_f(u)}, P^f)) + OPT(\mathcal{P}^B(F \setminus T_{c_f(u)}, P^o))$.

The optimality of the above procedure relies on the fact we can build an EST $\tilde{D}^f$ for $T_{c_f(u)}$ by starting from an optimal solution $D^*$ for $\mathcal{P}^B(F, P)$ and performing the following operation at each node $v$ of its left path: (i) if $v$ is unassigned we assign it as blocked; (ii) if $v$ is assigned to a node in $F \setminus T_{c_f(u)}$ we assign it as blocked and remove its right subtree. We can construct an EST $\tilde{D}^o$ for $F \setminus T_{c_f(u)}$ analogously. Notice that $cost(\tilde{D}^f) + cost(\tilde{D}^o) = cost(D^*)$. The proof is then completed by noticing that, for a particular choice of $\mathcal{U}$, $D^f$ and $D^o$ are feasible for $\mathcal{P}^B(T_{c_f(u)}, P^f)$ and $\mathcal{P}^B(F \setminus T_{c_f(u)}, P^o)$, so the solution returned by the above algorithm costs at most $OPT(\mathcal{P}^B(T_{c_f(u)}, P^f)) + OPT(\mathcal{P}^B(F \setminus T_{c_f(u)}, P^o)) \leq cost(D^*)$.

**Case 2:** $F$ is a tree $T_v$. Let $p_i$ be an unassigned node of $P$ and let $t$ be an integer in the interval $[i+1, B]$. The algorithm considers all possibilities for $p_i$ and $t$ and computes an EST $D^{i,t}$ for $T_v$ of smallest cost satisfying the following: (i) $D^{i,t}$ is compatible with $P$; (ii) its height is at most $B$; (iii) the node of the left path of $D^{i,t}$ corresponding to $p_i$ is assigned to $v$; (iv) the leaf of $D^{i,t}$ assigned to $v$ is located at level $t$. The algorithm then returns the tree $D^{i,t}$ with minimum cost.

In order to compute $D^{i,t}$ the algorithm executes the following steps:

1. Let $P^i$ be the subpath of $P$ that starts at the first node of $P$ and ends at $p_i$. Let $P^{i,t}$ be a left path obtained by appending $t - i$ unassigned nodes to $P^i$ and assigning $p_i$ as blocked (Figure 7.b). Compute an optimal solution $D'$ for $\mathcal{P}^B(\{T_{c_1(u)}, T_{c_2(u)}, \ldots, T_{c_{i-1}(u)}\}, P^{i,t})$. 
2. Let \( p_i' \) be the node of \( D' \) corresponding to \( p_i \) and let \( y' \) be the last node of the left path of \( D' \) (Figure 4(c)). The tree \( D_i^{1,i} \) is constructed by modifying \( D’ \) as follows (Figure 7(d)): make the left subtree of \( p_i' \) becomes its right subtree; assign \( p_i' \) to \( v \); add a leaf assigned to \( v \) as the left child of \( y' \); finally, as a technical detail, add some blocked nodes to extend the left path of this structure until the left path has the same size of \( P \).

It follows from properties (i) and (ii) of the trees \( D_i^{1,i} \)'s that the above procedure returns a feasible solution for \( \mathcal{P}^B(T_v, P) \). The proof of the optimality of this solution uses the same type of arguments as in Case 1 and is deferred to the appendix.

**Computational complexity.** Notice that it suffices to consider problems \( \mathcal{P}^B(F, P)'s \) where \( |P| \leq B \), since all others are infeasible. We claim that, by employing a Dynamic Programming strategy, we can compute all these problems in \( O(n^{2B^2}) \) time. First, there are \( O(n^{2B}) \) such problems; this follows from the fact that for each node \( u \) in \( T \) there are two possible forests \( F \) considered in subproblems (\( F = T_u \) or \( F = \{ T_{e_1(u')}, T_{e_2(u')}, \ldots, T_{e_f(u')} = T_u \} \), where \( u \) is the \( f \)-th child of \( u' \)) and the fact there are \( O(2^{2B}) \) PLP’s of size at most \( B \). It is not difficult to see that each of these problems can be solved in \( O(n + 2^B) \) time, so the claim holds.

**An upper bound on the height of optimal search trees.** We now argue that there is an optimal search tree for \((T, w)\) whose height is \( O(\Delta(T) \cdot (\log w(T) + \log n)) \).

The following lemma is the core of our ‘geometric decrease’ argument. It essentially states that we can cut a constant factor of the total weight of an optimal search tree by going down a number of levels that only depends on the maximum degree of \( T \).

**Lemma 7.** Consider an instance \((T, w)\) for our search problem and let \( D^* \) be an optimal search tree for it. Fix \( 0 \leq \alpha < 1 \) and an integer \( c > 3(\Delta(T) + 1)/\alpha \). Then, for every node \( v^* \in D^* \) with \( d(r(D^*), v^*) \geq c \) we have that \( w(D^*_{v^*}) \leq \alpha \cdot w(D^*) \).

**Proof sketch.** (The full proof is deferred to the appendix.) By means of contradiction assume the lemma does not hold for some \( v^* \) satisfying its conditions. Let \( \tilde{T} \) be the tree associated with \( v^* \), rooted at node \( \tilde{r} \). Since by hypothesis \( \tilde{T} \) contains a large portion of the total weight (greater than \( \alpha \cdot w(D^*) \)), we create the following search tree \( D' \) which makes sure parts of \( \tilde{T} \) are queried closer to \( r(D') \): the root of \( D' \) is assigned to \( \tilde{r} \); the left tree of \( r(D') \) is a search tree for \( T - T_{\tilde{r}} \) obtained via Lemma 4 in the right tree of \( r(D') \) we build a left path containing nodes corresponding to queries for \( c_1(\tilde{r}), c_2(\tilde{r}), \ldots, c_\delta(\tilde{r}) \), each having as right subtree a search tree for the corresponding \( T_{c_i(\tilde{r})} \) obtained via Lemma 4. If \( \tilde{s} \) is the number of nodes of \( T - T_{\tilde{r}} \) queried in \( r(D^*) \sim v^* \), then Lemma 4 implies that \( D' \) saves at least \( \tilde{s} - (\Delta(T) + 1) \) queries for each node in \( \tilde{T} \) when compared to \( D^* \); this gives the expression \( \text{cost}(D') \leq \text{cost}(D^*) - \tilde{s} \cdot w(\tilde{T}) + (\Delta(T) + 1)w(T) \).

Using the hypothesis on \( c \) and \( w(\tilde{T}) \), this is enough to reach the contradiction \( \text{cost}(D') < \text{cost}(D^*) \) when \( \tilde{s} \geq c/3 \). The case when \( \tilde{s} < c/3 \) is a little more involved but uses a similar construction, only now the role of \( \tilde{r} \) is taken by a node inside \( T_{\tilde{r}} \) in order to obtain a more ‘balanced’ search tree.

Assume that the weight function \( w \) is strictly positive (see Appendix E.3 for the general case). Since \( w \) is integral, employing Lemma 7 repeatedly shows that \( D^* \) has height at most \( O(\Delta(T) \cdot (\log w(T) + \log n)) \).

**From the DP algorithm to an FPTAS.** By Lemmas 6 and 7 we can obtain an optimal search tree for \((T, w)\) by finding an optimal EST of height \( B = O(\Delta(T) \cdot (\log w(T) + \log n)) \) (via \( \mathcal{P}^B \)) and then converting it into an optimal search tree. Since we can employ the algorithm presented in the previous section to achieve this in \( O((n \cdot w(T))^{O(\Delta(T))}) \) time, we obtain a pseudo-polynomial time algorithm for trees with bounded degree. Furthermore, such an algorithm can be transformed into an FPTAS by scaling and rounding the weights \( w \), just as in the well-known FPTAS for the knapsack problem (see the appendix for details):

**Theorem 3.** Consider an instance \((T, w)\) to our search problem where \( \Delta(T) = O(1) \). Then there is a \( \text{poly}(n \cdot w(T)) \)-time algorithm for computing an optimal search tree for \((T, w)\). In addition, there is a \( \text{poly}(n/\epsilon) \)-time algorithm for computing an \((1 + \epsilon)\)-approximate search tree for \((T, w)\).
References

[1] M. Adler, E. Demaine, N. Harvey, and M. Patrascu. Lower bounds for asymmetric communication channels and distributed source coding. In SODA, pages 251–260, 2006.

[2] M. Adler and B. Heeringa. Approximating optimal binary decision trees. In APPROX-RANDOM, pages 1–9, 2008.

[3] M. Adler and B. Maggs. Protocols for asymmetric communication channels. Journal of Computer and System Sciences, 63(4):573–596, 2001.

[4] E. Arkin, H. Meijer, J. Mitchell, D. Rappaport, and S. Skiena. Decision trees for geometric models. International Journal of Computational Geometry and Applications, 8(3):343–364, 1998.

[5] Y. Ben-Asher, E. Farchi, and I. Newman. Optimal search in trees. SIAM Journal on Computing, 28(6):2090–2102, 1999.

[6] R. Carmo, J. Donadelli, Y. Kohayakawa, and E. Laber. Searching in random partially ordered sets. Theoretical Computer Science, 321(1):41–57, 2004.

[7] V. Chakaravarthy, V. Pandit, S. Roy, P. Awasthi, and M. Mohania. Decision trees for entity identification: Approximation algorithms and hardness results. In PODS, pages 53–62, 2007.

[8] C. Daskalakis, R. Karp, E. Mossel, S. Riesenfeld, and E. Verbin. Sorting and selection in posets. In SODA, pages 392–401, 2009.

[9] P. de la Torre, R. Greenlaw, and A. Schäffer. Optimal edge ranking of trees in polynomial time. Algorithmica, 13(6):592–618, 1995.

[10] D. Dereniowski. Edge ranking and searching in partial orders. Discrete Applied Mathematics, 156(13):2493–2500, 2008.

[11] U. Faigle, L. Lovász, R. Schrader, and Gy. Turán. Searching in trees, series-parallel and interval orders. SICOMP: SIAM Journal on Computing, 15, 1986.

[12] M. Garey. Optimal binary identification procedures. SIAM Journal on Applied Mathematics, 23(2):173–186, 1972.

[13] M. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York, NY, 1979.

[14] A. Garsia and M. Wachs. A new algorithm for minimum cost binary trees. SIAM Journal on Computing, 6(4):622–642, 1977.

[15] S. Ghazizadeh, M. Ghodsi, and A. Saberi. A new protocol for asymmetric communication channels: Reaching the lower bounds. Scientia Iranica, 8(4), 2001.

[16] T. Hu and A. Tucker. Optimal computer search trees and variable-length alphabetic codes. SIAM Journal on Applied Mathematics, 21(4), 1971.

[17] L. Hyafil and R. Rivest. Constructing optimal binary decision trees is NP-complete. Information Processing Letters, 5(1):15–17, 1976.

[18] O. Ibarra and C. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM, 22(4):463–468, 1975.

[19] A. Iyer, H. Ratliff, and G. Vijayan. On an edge ranking problem of trees and graphs. Discrete Applied Mathematics, 30(1):43–52, 1991.

[20] D. Knuth. The Art of Computer Programming, Vol. 3: Sorting and Searching. Addison-Wesley, Reading, Massachusetts, 1973.

[21] R. Kosaraju, T. Przytycka, and R. Borgstrom. On an optimal split tree problem. In WADS, pages 157–168, 1999.

[22] E. Laber and L. Holanda. Improved bounds for asymmetric communication protocols. Information Processing Letters, 83(4):205–209, 2002.
[23] E. Laber and M. Molinaro. An approximation algorithm for binary searching in trees. In *ICALP*, pages 459–471, 2008.
[24] E. Laber and L. Nogueira. On the hardness of the minimum height decision tree problem. *Discrete Applied Mathematics*, 144(1-2):209–212, 2004.
[25] T. Lam and F. Yue. Optimal edge ranking of trees in linear time. In *SODA*, pages 436–445, 1998.
[26] N. Linial and M. Saks. Searching ordered structures. *Journal of Algorithms*, 6, 1985.
[27] M. Lipman and J. Abrahams. Minimum average cost testing for partially ordered components. *IEEE Transactions on Information Theory*, 41(1):287–291, 1995.
[28] S. Mozes, K. Onak, and O. Weimann. Finding an optimal tree searching strategy in linear time. In *SODA*, pages 1096–1105, 2008.
[29] K. Onak and P. Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *FOCS*, pages 379–388, 2006.
[30] A. Schäffer. Optimal node ranking of trees in linear time. *Information Processing Letters*, 33(2):91–96, 1989.
[31] J. Watkinson, M. Adler, and F. Fich. New protocols for asymmetric communication channels. In *SIROCCO*, pages 337–350, 2001.
Appendix

A   The proof of lemma 1

We need two inequalities regarding the weights.
Fact 1 For each $1 \leq i' < i \leq m$ it holds that

\[ w(t_i) > w(a_{i1}) > w(t_{i'}) + w(u_{i'}1) + w(u_{i'}2) + w(u_{i'}3) \]  

(proof of the fact. The first inequality follows by definition. In order to prove the second inequality let us consider the difference

\[ Diff = w(a_{i1}) - (w(t_{i'}) + w(u_{i'}1) + w(u_{i'}2) + w(u_{i'}3)). \]

By definition we have

\[ Diff = \sum_{j=1}^{3} (W_{uij} + \gamma(i,j)w(u_{ij})) - \sum_{j=1}^{3} (W_{u_{i'}j} + (\gamma(i',j) + 3/2)w(u_{i'}j)). \]

Case 1. $u_{i3} = u_{i'3}$. Note that $\gamma(i,3) \geq 5 + \gamma(i',3)$. Since $W_{uij}, W_{u_{i'}j} \geq 0$ and $0 < \gamma(i,j), \gamma(i',j) \leq |T|$ we get that

\[ Diff \geq 5w(u_{i3}) - 3W_{u_{i'3}} - (2|T| + 3)w(u_{i'2}). \]

Let $\kappa$ be such that $u_{i3} = u_{\kappa}$. It follows from the definition of the function $w()$ that

\[ w(u_{i3}) = w(u_{\kappa}) = 1 + 6 \max\{W_{u_{\kappa}}, |T|^3 w(u_{\kappa-1})\} > 3W_{u_{i'3}} + (2|T| + 3)w(u_{i'2}). \]

Thus, $Diff > 0$.

Case 2. $u_{i'3} < u_{i3}$. Then, it must also hold that $X_{i'} < u_{i3}$. Therefore we have

\[ w(a_i) \geq W_{ui3} \geq w(X_{i'}) > w(t_{i'}) + w(u_{i'}1) + w(u_{i'}2) + w(u_{i'}3). \]

Fact 2 For each $1 \leq i \leq m$ and $\kappa = 1, \ldots, 4$, it holds that

\[ w(a_{i\kappa}) \geq 3(w(u_{i3}) + w(u_{i2}) + w(u_{i1})) + W_{u_{i3}} \]  

It follows directly from the definition of $w(a_{i\kappa})$ and the the fact that $\gamma(i,j) \geq 3$ ($j = 1, 2, 3$).

Proof of Lemma 1. Let $D$ be an optimal search tree for $(T, w)$.

Let $\ell$ be the deepest node in the left path of $D$ such that $D - D_\ell$ is the realization of $\pi_{i+1} \ldots \pi_{n+m}$ for some $i = 0, \ldots, n + m$. In particular, we take $i = n + m$ if $\ell$ is the root of $D$, i.e., no upper part of $D$ looks like a realization of suffix of $\Pi$.

By contradiction, assume that $D$ is not a realization of $\Pi$, in particular $i > 0$. We shall prove that by modifying $D_\ell$ in such a way that its top part becomes a realization of $\pi_i$ we obtain a new search tree with cost smaller than the cost of $D$. The desired result will follow by contradiction. We consider the following cases:

Case 1. $\pi_i = X_j$, for some $j = 1, 2, \ldots, m$. First we argue that $\ell \in \{q_{t_1}, q_{r_j}\}$. Let $q_\nu$ (for some $\nu \in T$) be the parent of $q_{r_j}$. If $\nu \in T_j$ we swap $q_{r_j}$ with $q_\nu$ otherwise we swap $q_{r_j}$ with $q_{r_\nu}$ 4 Let $D'$ be the new tree so obtained.

4When swapping we imply that the two nodes are exchanging position and they are carrying along also their right subtrees. This is possible because $q_{r_j}$ is the left child of $q_\nu$. 

13
If \( \nu \) is a leaf in \( T \), then we have \( \text{cost}(D') \leq \text{cost}(D) - w(t_j) + w(\nu) < \text{cost}(D) \) since \( t_j \) is the leaf of largest weight in \( D_\ell \). Otherwise, it must be that \( \nu = r_{j'} \) for some \( j' < j \). In this case, by (3), we have \( \text{cost}(D') \leq \text{cost}(D) - w(t_j) + w(t_{j'}) + w(u_{j'2}) + w(u_{j'1}) < \text{cost}(D) \). In either case we obtain a tree of average weight smaller than \( D \), violating the optimality of \( D \).

Alternatively, if \( q_t \) is not the right child of \( q_{r_j} \), then we swap \( q_t \) with its parent. Note that \( q_t \) must be the left child of its parent. By proceeding as above, we can prove that the resulting tree has cost smaller than \( D \), again a violation to the optimality of \( D \).

Therefore, it must be \( \ell \in \{ q_{t_j}, q_{r_j} \} \). We now split the analysis according to this two possible cases.

**Subcase 1.1.** \( \ell = q_{r_j} \). Then, because of the assumption on \( D - D_\ell \) and the search property, it follows that the right subtree of \( q_{r_j} \) contains the nodes \( q_{t_j}, q_{s_{j3}}, q_{s_{j2}}, q_{s_{j1}} \). Also, it is not hard to see that they must appear in this order, for otherwise by reordering them we would decrease the average cost of \( D \), since \( w(t_j) > w(s_{j3}) > w(s_{j2}) > w(s_{j1}) \). Therefore the right subtree of \( \ell \) coincides with the right subtree of \( D_j^A \).

Suppose now w.l.o.g. that for each \( \kappa = 2, 3, 4 \), it holds that \( q_{a_{j\kappa-1}} \) is closer to the root of \( D \) than \( q_{a_{j\kappa}} \). For the sake of contradiction, assume that \( q_{a_{j1}} \) is not a child of \( q_{r_j} \). Let \( q_\nu \) be the parent of \( q_{a_{j1}} \). Note that \( q_{a_{j1}} \) can only be the left child of \( q_\nu \). By swapping \( q_{a_{j1}} \) with \( q_\nu \) the resulting tree has smaller expected cost than \( D \), again in contradiction with the assumed optimality of \( D \). In fact, if \( \nu \) is a leaf in \( T \) then it follows from inequality (4) that \( w(a_{j1}) > w(u_{j3}) \geq w(\nu) \). Otherwise, if \( \nu = r_{j'} \) for some \( j' < j \), and then, by (3) we have that \( w(a_{j1}) \) is greater than the weight of the right subtree of \( q_\nu \). The same arguments show that \( q_{a_{j\kappa}} \) is the left child of \( q_{a_{j\kappa-1}} \), for each \( \kappa = 2, 3, 4 \).

We can conclude that in the left path of \( D \), the nodes following \( \ell \) are exactly \( q_{a_{j1}}, \ldots, q_{a_{j4}} \). Let \( \ell' \) be the left child of \( q_{a_{j4}} \). We have showed that in this subcase \( D_\ell - D_{\ell'} \) coincides with \( D_j^A \).

**Subcase 1.2.** \( \ell = q_{t_j} \). There is nothing to prove about the right subtree of \( \ell \). In order to prove that in the left path of \( D \), the node \( \ell \) is followed by \( q_{a_{j1}}, \ldots, q_{a_{j4}} \) we proceed as before. Assume (by contradiction) that \( q_{a_{j1}} \) is not a child of \( q_{r_j} \). Let \( q_\nu \) be the parent of \( q_{a_{j1}} \). Note that \( q_{a_{j1}} \) can only be the left child of \( q_\nu \). We swap \( q_{a_{j1}} \) with \( q_\nu \). Let \( D' \) be the resulting search tree. If \( \nu = r_{j'} \) or a leaf in \( T_j \setminus \{ t_j \} \), we have that \( \text{cost}(D') = \text{cost}(D) - w(a_{j1}) + w(X_j) < 0 \), where \( w(X_j) \) accounts for the weight of the right subtree of \( q_\nu \) and the last inequality follows by (4). On the other hand, if \( \nu \) is either a leaf in \( T \) or is equal to \( r_{j'} \) for some \( j' < j \), then we can apply the same argument as in Subcase 1.1, to reach the same conclusion, i.e., we violate the optimality of \( D \).

Therefore, we conclude that \( q_{a_{j1}} \) is the left child of \( q_\ell \). Repeating the same argument we can also show that \( q_{a_{j\kappa}} \) is the left child of \( q_{a_{j\kappa-1}} \), for each \( \kappa = 2, 3, 4 \). Let \( \ell' \) be the left child of \( q_{a_{j4}} \). We have showed that in this subcase, \( D_\ell - D_{\ell'} \) coincides with \( D_j^B \).

We can conclude that in both subcases of Case 1, the tree \( D - D_{\ell'} \) is realization of \( \pi_1, \ldots, \pi_{n+m} \) against the assumption that \( \ell \) is the deepest node for which such a condition holds.

**Case 2.** \( \pi_i = u_j \), for some \( j = 1, 2, \ldots, n \).

Let us consider the set of leaves \( L \) of \( T^b \) which are associated with \( u_j \) and such that they are not queried in \( D - D_\ell \). Since \( D - D_\ell \) is a realization of \( \pi_{i+1} \ldots \pi_{n+m} \), the leaves of \( T^b \) which are not in \( L \) and are queried in \( D_\ell \) are either in \( \bigcup_{X < u_j} X \) or are associated to \( u_{j'} \) for some \( j' < j \). For the sake of contradiction we assume that one of the first \( |L| \) nodes in the left path of \( D_\ell \) does not correspond to a leaf in \( L \).

Let us construct a tree \( D' \) from \( D_\ell \) as follows: first we construct an auxiliary tree by removing from \( D_\ell \) all the nodes corresponding to the leaves in \( L \). Then, we add a left path with these nodes to the top of this auxiliary tree. Our assumption that one of the first \( |L| \) nodes in the left path of \( D_\ell \) does not correspond to a leaf in \( L \) implies that

---

5We are again assuming, w.l.o.g., that for each \( \kappa = 2, 3, 4 \), it holds that \( q_{a_{j\kappa-1}} \) is closer to the root of \( D \) than \( q_{a_{j\kappa}} \).
Therefore, we have the desired contradiction:

The negative term in the equation above is because the sum of the levels of the nodes associated with $u_j$ in $D_\ell$ is at least 7 while this sum is exactly 6 in $D'$. The other terms are due to the fact that the level of a node can increase by at most $|L|$ units in our construction. The definitions of $W_{u_j}$ and $w(u_j)$ imply that

$$
cost(D') \leq cost(D_\ell) - w(u_j) + |L| \sum_{X \prec u_j} w(X) + 3 \cdot |L| \cdot \sum_{u < u_j} w(u)
$$

Since $|L| \leq 3$ and $w(u_j) > 6 \max\{W_{u_j}, |T|^3 w(u_{j-1})\}$ we get that $cost(D') < cost(D_\ell)$. This implies, however, that $D$ can be improved, a contradiction.

Thus, the $D_\ell$’s $|L|$ top levels coincide with a sequential search tree for $L$. Let $\ell'$ the left most query of such sequential search. Therefore, $D - D_\ell'$ is realization of $\pi_1, \ldots, \pi_{n+1}$, which contradicts also in this Case 2 the hypothesis that $\ell$ is the deepest node for which such a condition holds.

The proof is complete.

\[ \square \]

**B The proof of Lemma 2**

**Lemma 2.** Let $D^*$ be an optimal binary search tree for $(T, w)$. Let $\mathcal{Y} \subseteq \mathcal{X}$ be such that $D^*$ is a realization of $\Pi$ w.r.t. $\mathcal{Y}$. We have that $cost(D^*) \leq cost(D^A) - \frac{1}{2} \sum_{u \in U} w(u)$ if and only if $\mathcal{Y}$ is a solution for the X3C instance $\mathcal{I} = (U, \mathcal{X})$.

**Proof.** We start proving the only if part. Assume that $cost(D^A) - cost(D^*) \geq \frac{1}{2} \sum_{u \in U} w(u)$. We shall use induction on $j$ to prove that for each $j = n, \ldots, 1$ there exists exactly one $X \in \mathcal{Y}$, such that $u_j \in X$.

Fix $j^* \leq n$ and assume that for every $j > j^*$ it holds that there exists exactly one $X \in \mathcal{Y}$ such that $u_j \in X$.

Suppose that there is no $i \in \{1, \ldots, m\}$ such that $u_i^* \in X_i \in \mathcal{Y}$. We can rewrite (2) as follows:

$$
cost(D^A) - cost(D^*) = \sum_{j=1}^{n} \sum_{\substack{X_i \in \mathcal{Y} \\ u_j \in X_i}} \left( \frac{w(u_j)}{2} + \Gamma(i, j) w(u_j) \right),
$$

where $\Gamma(i, j) = \gamma(i, \kappa) - d_B^A(q_{s_i, \kappa})$, and $\kappa \in \{1, 2, 3\}$ such that $s_i^{\kappa} = u_j$.

Now, since we are assuming that for all $j > j^*$ there exists only one $i$ such that $X_i \in \mathcal{Y}$ and $u_j \in X_i$, by the definition of $d_B^A(\cdot)$ and $\gamma(i, \kappa)$, we have $\Gamma(i, j) = 0$. So we obtain

$$
cost(D^A) - cost(D^*) = \sum_{j > j^*} \frac{w(u_j)}{2} + \sum_{j < j^*} \sum_{\substack{X_i \in \mathcal{Y} \\ u_j \in X_i}} \left( \frac{w(u_j)}{2} + \Gamma(i, j) w(u_j) \right),
$$

where we also used the assumption that no $X \in \mathcal{Y}$ contains $u_j^*$ and therefore $u_j^*$ does not contribute to the sum.

Now we can observe that, for each $j < j^*$, there are at most 3 set in $\mathcal{X}$ containing $u_j$. Moreover, $\Gamma(i, j)$ being a difference of levels in $D^*$ can be bounded by $|T|$. Also $w(u_j) \leq w(u_{j^*})/6|T|^3$, for each $j < j^*$. Therefore, we have the desired contradiction:

$$
cost(D^A) - cost(D^*) \leq \sum_{j > j^*} \frac{w(u_j)}{2} + 3(j^* - 1)(|T| + 1/2) \frac{w(u_{j^*})}{6|T|^3} \leq \sum_{j > j^*} \frac{w(u_j)}{2} + \frac{w(u_{j^*})}{2} \leq \frac{1}{2} \sum_{u \in U} w(u).
$$
Suppose now that there are $\kappa > 1$ subsets in $\mathcal{Y}$ that contain $u_{j^*}$. Rewriting (2) as before, we obtain:

$$\text{cost}(D_A) - \text{cost}(D^*) \leq \sum_{j > j^*} \frac{w(j)}{2} + \sum_{\begin{subarray}{c}i,j \in \mathcal{Y} \\ u_j \in X_i \end{subarray}} \left(\frac{w(u_{j^*})}{2} + \Gamma(i, j^*)w(u_{j^*})\right) + \sum_{j < j^*} \sum_{\begin{subarray}{c}i,j \in \mathcal{Y} \\ u_j \in X_i \end{subarray}} \left(\frac{w(u_j)}{2} + \Gamma(i, j)w(u_j)\right).$$

Let us observe that among the $\kappa$ sets $X_i \in \mathcal{Y}$ such that $u_{j^*} \in X_i$ only one contributes with a positive weight $w(u_{j^*})/2$ since $\Gamma(i, j^*) = 0$. For the others, we have a negative contribution of at least $w(u_{j^*})/2$, since $\Gamma(i, j^*)$ becomes negative. Moreover, for the $j < j^*$ we can repeat the argument we used in the previous case. Therefore we obtain the desired contradiction:

$$\text{cost}(D_A) - \text{cost}(D^*) \leq \sum_{j > j^*} \frac{w(j)}{2} - \frac{\kappa - 2}{2} w(u_{j^*}) + 3(j^* - 1)(|\mathcal{T}| + 1/2) \frac{w(u_{j^*})}{6|\mathcal{T}|^3} < \sum_{j > j^*} \frac{w(u_j)}{2} < \frac{1}{2} \sum_{u \in \mathcal{U}} w(u).$$

This concludes the inductive argument and the proof of the only if part.

In order to prove the if part of the statement we notice that if $\mathcal{Y}$ is a solution for $\mathcal{I}$ then for each $j = 1, \ldots, n$ there exists exactly one index $i$ such that $X_i \in \mathcal{Y}$ and $u_j \in X_i$. Then, the desired result follows directly by equation (2), and by the fact that in this case the definition of $d_B^i(\cdot)$ and $\gamma(\cdot, \cdot)$, yields $\Gamma(i, j) = 0$. 

\[\square\]

**C The proof of Lemma 3**

**Proof of Lemma 3.** Let $D$ be an optimal search tree for $(T^b, w)$.

Let $\ell$ be the deepest node in the left path of $D$ such that $D - D_\ell$ is the realization of $\pi_{i+1} \ldots \pi_{n+m}$ for some $i = 0, \ldots, n + m$. In particular, we take $i = n + m$ if $\ell$ is the root of $D$, i.e., no upper part of $D$ looks like a realization of some suffix of $\mathcal{I}$.

By contradiction, assume that $D$ is not a realization of $\mathcal{I}$, whence $i > 0$. We shall prove that by modifying $D_\ell$ in such a way that its top part becomes a realization of $\pi_i$ we obtain a new search tree with cost smaller than the cost of $D$. The desired result will follows by contradiction. We consider the following cases:

**Case 1.** $\pi_i = X_j$, for some $j = 1, 2, \ldots, m$.

In this case, our assumption regarding $\ell$ implies that if a node $\nu \in D_\ell$ is associated with a leaf $\ell'$ in $T^b$ then $\ell'$ either corresponds to an element $u \in \mathcal{U}$ such that $u < X_j$ or $\ell' \in \mathcal{X}_{j'}$ such that $X_{j'} \preceq X_j$. Let $\kappa$ be such that $X_j \in H_\kappa$. We need to prove the following claim

**Claim 1.** $\ell \in \{q_{j'}, q_{r_j}\}$.

**Proof.** We shall show it by contradiction. We split the proof into cases I and II.

**Case I.** Suppose that the node $q_{t_j}$ is the right child of $q_{r_j}$. Let $q_{\nu}$ (for some $\nu \in \mathcal{T}$) be the parent of $q_{r_j}$.

We have two cases according as $q_{j'}$ is a right or a left child of $q_{\nu}$.

Subcase I.a $q_{j'}$ is a right child of $q_{\nu}$. Note that because of the search tree property $\nu$ must be an ancestor of $h_\kappa$ in $T^b$. We perform a left rotation on $q_{\nu}$. Let $D'$ be the new tree obtained. We have that $\text{cost}(D') \leq \text{cost}(D) - w(t_j) + w(\alpha)$, where $\alpha$ is the left subtree of $q_{\nu}$. We observe if a node in $\alpha$ corresponds to a leaf $\ell'$ then $\ell'$ must be in $T^b \setminus H_\kappa$.

Thus, the nodes of $\alpha$ can take care of:

(a) leaves that are associated to some $u \in \mathcal{U}$, such that $u < u_{j_3}$. The sum of the weights of these leaves is at most $|\mathcal{T}| \cdot w(u_{j_3})/6|\mathcal{T}|^3 < w(u_{j_3})/2$;

(b) at most two leaves associated with $u \in \mathcal{U}$ such that $u = u_{j_3}$. The fact that every $u \in \mathcal{U}$ appears in at most three sets of $\mathcal{X}$ together with the fact that $s_{j_3} \in H_\kappa$ explain that we have at most two leaves;
(c) leaves in $\bar{X}_{j'}$ such that $X_{j'} < u_{j3}$. The sum of the weights of these leaves sum at most $W_{u_{j3}}$.

Thus, we can conclude that $w(\alpha) \leq 2.5w(u_{j3}) + W_{u_{j3}}$. Since $w(t_j) > 2.5w(u_{j3}) + W_{u_{j3}}$ we conclude that $\text{cost}(D') < \text{cost}(D)$, contradicting the optimality of $D$.

Subcase I.b $q_{r_j}$ is a left child of $q_\nu$. This implies that $\nu$ is not an ancestor of $r_j$ in $T^b$. Let $D'$ be a tree obtained as follows: we swap $q_{r_j}$ with $q_\nu$ if $\nu$ is not in $T_j$; otherwise, we swap $q_{r_j}$ with $q_\nu$. Let $\alpha$ be the right subtree of $q_{r_j}$. Again, we have $\text{cost}(D') \leq \text{cost}(D) - w(t_j) + w(\alpha)$.

If $\nu \notin H_\kappa$ then the analysis is identical to the one employed in Subcase I.a because $\alpha$ can take care of the same leaves considered in that case.

If $\nu$ is a leaf in $H_\kappa$ then $w(t_j) > w(\nu) = w(\alpha)$ because $t_j$ is the heaviest leaf among the leaves in $H_\kappa$ that corresponds to a node in $D_\ell$. Finally, if $\nu$ is an internal node in $H_\kappa \setminus \{h_\kappa\}$ then $\nu = r_{j'}$ for some $j' < j$ and it follows from inequality (3) that $w(t_j) > w(t'_j) + w(u_{j'3}) + w(u_{j'2}) + w(u_{j'1}) = w(\alpha)$.

In either Subcase we obtain a tree of cost smaller than $D$ violating the optimality of $D$.

Case II. Alternatively, if $q_{r_j}$ is not the right child of $q_{r_j}$, then we can proceed as before. We consider the case where $q_{r_j}$ is the right child of its parent and also the case where it is the left child. In the former case we apply a left rotation and in the latter a simple swap. Again we can prove that the resulting tree has cost smaller than $D$, a violation to the optimality of $D$.

The proof of the claim is complete. \hfill \square

Therefore, it must be $\ell \in \{q_{t_k}, q_{r_j}\}$. We now split the analysis according to this two cases.

Subcase 1.1. $\ell = q_{r_j}$. Then, because of the assumption on $D - D_\ell$ and the search property, it follows that the right subtree of $q_{r_j}$ contains the nodes $q_{l_j}, q_{s_{j3}}, q_{s_{j2}}, q_{s_{j1}}$. Also, it is not hard to see that they must appear in this order, for otherwise, by reordering them we would decrease the average cost of $D$, since $w(t_j) > w(s_{j3}) > w(s_{j2}) > w(s_{j1})$. Therefore the right subtree of $\ell$ coincides with the right subtree of $D_j^A$.

Let us assume w.l.o.g that the level of $q_{a_{jk}}$ is smaller than or equal to the level $q_{a_{jk}}$, in $D$, for $k < k'$. First, we argue that the left child of $\ell$ must be $q_{a_{j1}}$. Assume that $q_{a_{j1}}$ is not the left child of $\ell$ and let $\nu$ be the parent of $q_{a_{j1}}$. We have two cases:

A. $q_{a_{j1}}$ is a right child of $\nu$.

We perform a left rotation on $q_\nu$. Let $D'$ be the new tree obtained. We have that $\text{cost}(D') \leq \text{cost}(D) - w(q_{a_{j1}}) + w(\alpha)$ where $\alpha$ is the left subtree of $\nu$. Note that the search property assures that $\nu$ is an ancestor of $h_\kappa$. Thus, the analysis of Subcase I.a in the above Claim 1, shows that the sum of the weights of the leaves of $\alpha$ can be upper bounded by $2.5w(u_{j3}) + W_{u_{j3}}$. Since $w(q_{a_{j1}}) > 3w(u_{j3}) + W_{u_{j3}}$ we conclude that $\text{cost}(D') < \text{cost}(D)$.

B. $q_{a_{j1}}$ is a left child of $\nu$. In this case, we swap $q_{a_{j1}}$ and $\nu$. Let $D'$ be the new tree obtained. We have that $\text{cost}(D') \leq \text{cost}(D) - w(q_{a_{j1}}) + w(\alpha)$ where $\alpha$ is the right subtree of $\nu$. Note that $\nu$ is not an ancestor of $a_{j1}$ in $T^b$.

If $\nu \notin H_\kappa$ the arguments employed in subcase IA shows that $w(\alpha) \leq 2.5w(u_{j3}) + W_{u_{j3}}$. Since $w(q_{a_{j1}}) > 3w(u_{j3}) + W_{u_{j3}}$ we conclude that $\text{cost}(D') < \text{cost}(D)$.

If $\nu \in T_j$, with $T_j \notin H_\kappa$ and $j' < j$, it follows from inequality (3) that $w(a_{j1}) > w(\alpha)$. If $\nu \in T_j$ it follows from inequality (4) that $w(a_{j1}) > w(\alpha)$. Finally, if $\nu = a_{j'k}$ with $j' < j$ we have that $w(a_{j1}) > w(a_{j'k}) = w(\alpha)$.

We can conclude that $q_{a_{j4}}$ is the left child of $\ell$. Since $w(a_{j1}) = w(a_{j2}) = w(a_{j3}) = w(a_{j4})$, the same arguments show that the nodes following $a_{j1}$ in the left path are $q_{a_{j2}}, q_{a_{j3}}$ and $q_{a_{j4}}$. Let $\ell'$ be the left child of $q_{a_{j4}}$. We have showed that in this subcase $D_\ell - D_{\ell'}$ coincides with $D_j^A$.

Subcase 1.2. $\ell = q_{t_k}$. There is nothing to prove about the right subtree of $\ell$. On the other hand, in order to prove that the nodes following $\ell$ in the left path of $D$ are exactly $q_{a_{j1}}, q_{a_{j2}}, q_{a_{j3}}$, and $q_{a_{j4}}$, we can proceed as in Subcase 1.1. The only additional case to be taken care of, in the argument by contradiction used
there, is when the parent of \(q_{a_{ij}}\) is \(q_{r_j}\). However, in this case we can employ the same argument we used for the analogous situation in Subcase 1.2. of the proof of Lemma 1. Let \(\ell'\) be the left child of \(q_{a_{ij}}\). We have showed that in this subcase, \(D_k - D_{\ell'}\) coincides with \(D_{\ell}^B\).

We can conclude that in both Subcase 1.1 and 1.2, the tree \(D - D_{\ell'}\) is a realization of \(\pi_i, \ldots, \pi_{n+m}\) against the assumption that \(v\) is the deepest node for which such a condition holds.

**Case 2.** \(\pi_i = u_j\), for some \(j = 1, 2, \ldots, n\).

The proof is identical to that employed for Case 2 of Lemma 1.

---

**D  The proof of Lemma 5**

Let \(x\) and \(x^*\) be the nodes queried at the root of \(D'\) and \(D^*\), respectively. W. l. o. g. we assume \(x \neq x^*\), as otherwise the lemma trivially holds. We can also assume that \(x^*\) is a node from \(T_x\), because the opposite case is analyzed analogously.

**Case 1:** \(w(T_x) \leq w(T - T_x)\). In other words, \(w(T_x) \leq w(T)/2\). As any path from \(r(D^*)\) to a leaf in \(D^*\) contains \(r(D^*)\) and \(T - T_x\) does not contain \(x^*\), Lemma 4 states that the depth of any leaf in \(D_1^*\) is at least by one smaller than it is in \(D^*\). The lemma also implies that the depth of any leaf in \(D_0^*\) is not greater than it is in \(D^*\). So we have

\[
\text{cost}(D') = w(T) + \text{cost}(D_0^*) + \text{cost}(D_1^*) \\
\leq w(T) + \sum_{v \in T_x} w(v)d(r(D^*), l_v) + \sum_{v \in T - T_x} w(v)(d(r(D^*), l_v) - 1) \\
= w(T) + \text{cost}(D^*) - w(T - T_x) \leq \text{cost}(D^*) + w(T)/2.
\]

**Case 2:** \(w(T_x) > w(T - T_x)\). Let \(x_1, \ldots, x_n\) be the nodes successively queried when the path \(r(D^*) \leadsto r(D')\) is traversed in \(D^*\). In particular, \(x_1 = x^*\) and \(x_n = x\). Let \(k < n\) be such that \(x_i\) is a node from \(T_x - \{x\}\) for \(i = 1, \ldots, k\) and \(x_{k+1} \notin T_x - \{x\}\).

In this extended abstract we assume that \(w(T_x - T_{x_i}) > 0\) for \(i = 1, \ldots, k\). The case of \(w(T_x - T_{x_i}) = 0\) can only occur when there is tie regarding the choice of node \(x\) in step (1) of the algorithm, and then the above scenario can be avoided by employing a suitable tie breaking rule. In the full paper we will show by a more intricate case analysis that the approximation factor holds regardless of the tie breaking rule.

For \(i = 1, \ldots, k\) we know that \(w(T_{x_i}) < w(T - T_{x_i})\), because otherwise, using the assumption that \(w(T_x - T_{x_i}) > 0\), we would have \(w(T_{x_i}) - w(T - T_{x_i}) = w(T_{x_i}) - w(T_x - T_{x_i}) - w(T - T_x) = w(T_{x_i}) - w(T - T_x) - 2w(T_{x_i} - T_{x_i}) < w(T_x) - w(T - T_x)\), and so \(x_i\) would have been chosen instead of \(x\) in step (1) of the algorithm.

From this fact, it follows that \(w(T_{x_i}) \leq w(T - T_x)\) for \(i = 1, \ldots, k\). This is because otherwise \(w(T_x) - w(T - T_x) = w(T_{x_i}) + w(T_x - T_{x_i}) - w(T - T_x) > w(T - T_x) + w(T_x - T_{x_i}) - w(T_{x_i}) = w(T - T_{x_i}) - w(T_{x_i}) \geq 0\), so \(x_i\) would have been chosen instead of \(x\) in step (1).

Let \(T' := \bigcup_{i=1}^{k} T_{x_i}\) and let \(T'' := T_x - T'\). Note that \(T'\) is a forest in general and \(T' \cup T'' = T_x\). We are going to reason about the search tree depths of the nodes in \(T - T_x\), \(T'\), and \(T''\) separately.

\(D_0^*\) queries all nodes from \(T'\), and Lemma 4 states that the depth of those nodes is not greater in \(D_0^*\) than it is in \(D^*\).

The nodes from \(T''\) are as well all queried in \(D_0^*\). For these nodes we know that in \(D^*\) the node \(x_{k+1}\) is queried before them. As \(x_{k+1}\) is not queried by \(D_0^*\), the depth of each node from \(T''\) in \(D_0^*\) is by at least by one smaller than it is in \(D^*\).

Finally, the leaves in \(D^*\) corresponding to the nodes from \(T - T_x\) are descendants of the nodes in \(D^*\) querying \(x_1, \ldots, x_k\). These \(k\) nodes are not contained in \(D_1^*\), so the depth of each leaf in \(D_1^*\) is at least
by $k$ smaller than it is in $D^*$. Combining the findings, we obtain
\[
\text{cost}(D') = w(T) + \sum_{v \in T'} w(v)d(r(D^*_{l_v}), l_v) + \sum_{v \in T''} w(v)d(r(D^*_{l_v}), l_v) \\
\leq w(T) + \sum_{v \in T'} w(v)d(r(D^*_{l_v}), l_v) + \sum_{v \in T''} w(v)(d(r(D^*_{l_v}), l_v) - 1) + \sum_{v \in T - T_x} w(v)(d(r(D^*_{l_v}), l_v) - k) \\
= w(T) + \text{cost}(D^*) - w(T'') - kw(T - T_x).
\]
As $T' = T - ((T - T_x) \cup T'')$, we have $w(T') = w(T) - w(T - T_x) - w(T'')$, so
\[
\text{cost}(D') \leq \text{cost}(D^*) + w(T') - (k - 1)w(T - T_x).
\]
We have argued above that $w(T_{x_i}) \leq w(T - T_x)$ for $i = 1, \ldots, k$. Therefore, $w(T') = w(\bigcup_{i=1}^k T_{x_i}) \leq \sum_{i=1}^k w(T_{x_i}) \leq kw(T - T_x)$, and
\[
\text{cost}(D') \leq \text{cost}(D^*) + kw(T - T_x) - (k - 1)w(T - T_x) = \text{cost}(D^*) + w(T - T_x) \leq \text{cost}(D^*) + w(T)/2.
\]
\[\square\]

**E  An FPTAS for Searching in Bounded-Degree Trees**

**E.1 Algorithm for $\mathcal{P}^B(F, P)$**

In this section we complete the correctness proof of the proposed algorithm for solving $\mathcal{P}^B(F, P)$. It has already been argued in Section 3.2 that the algorithm always returns a feasible solution. In addition, in Case 1 of the algorithm, the returned solution is also optimal. Here we prove the optimality for the second case:

**Case 2:** $F$ is a tree $T_v$. Let $D^*$ be an optimal solution for $\mathcal{P}^B(T_v, P)$. Consider the internal node of $D^*$ assigned to $v$; since $D^*$ is compatible with $P$ and since this node belongs to the left path of $D^*$, it corresponds to a node $p_i$ of $P$. Thus, we denote this internal node of $D^*$ assigned to $v$ by $p_i'$. Let $v'$ be the leaf of $D^*$ assigned to $v$ and notice that $v'$ lies in the left path of the right subtree of $p_i'$. We construct $D'$ from $D^*$ by essentially applying the inverse of Step 2 of the algorithm: remove from $D^*$ the right subtree of $p_i'$; this removed subtree becomes the subtree of $p_i'$; assign $p_i'$ as blocked and remove $v'$. (One can use Figures 4 and 5c to better visualize this construction.)

The tree $D'$ is actually an EST for the forest $\{T_{c_1(v)}, \ldots, T_{c_d(v)}(v)\}$ and has height at most $B$. Now construct $\tilde{P}'$ by taking the left path of $D'$, setting all the non-blocked nodes as unassigned and also setting every node after $p_i'$ as unassigned. Clearly $\tilde{D}'$ is compatible with $\tilde{P}'$ and thus feasible for $\mathcal{P}^B(\{T_{c_1(v)}, \ldots, T_{c_d(v)}(v)\}, \tilde{P}')$.

Notice, however, that $\tilde{P}'$ starts with the prefix of $P$ until $p_i$ (in terms of its assignment), then it has a blocked node corresponding to $p_i$ and then some unassigned nodes. Let $\tilde{t}$ be the number of nodes in $\tilde{P}'$. Since the last node of $\tilde{P}'$ comes from the parent of $v'$ in $D^*$ and $D^*$ has height at most $B$, we have that $\tilde{t} \leq B$. Thus, the path $\tilde{P}'$ coincides with the path $P^{i, \tilde{t}}$ constructed by the algorithm when $t = \tilde{t}$.

It is easy to see that the tree $D^{i, \tilde{t}}$, as defined in the algorithm, has cost
\[
\text{OPT}(\mathcal{P}^B(\{T_{c_1(v)}, \ldots, T_{c_d(v)}(v)\}, P^{i, \tilde{t}}) + \tilde{t} \cdot w(v) = \text{OPT}(\mathcal{P}^B(\{T_{c_1(v)}, \ldots, T_{c_d(v)}(v)\}, \tilde{P}') + \tilde{t} \cdot w(v),
\]
which is at most $\text{cost}(D') + \tilde{t}w(v)$ due to the feasibility of $\tilde{D}'$. Finally, notice that this last quantity is actually the cost of $D^*$, so $\text{cost}(D^{i, \tilde{t}}) \leq \text{cost}(D^*)$. Since the procedure returns a solution which is at least as good as $D^{i, \tilde{t}}$, its optimality follows.
E.2 Proof of Lemma 7

By means of contradiction suppose \( v^* \in D^* \) with \( d(r(D^*), v^*) \geq c \) but \( w(D^*_{v^*}) > \alpha \cdot w(D^*) \). Let \( \tilde{T} \) be the subtree of \( T \) associated with \( v^* \) and let \( x \) be the root of \( \tilde{T} \).

Let \( y \) be a node in \( T_x \) to be specified later. Let \( T^0 = T - T_y \) and \( T^i = T_{c_i(y)} \) for \( i = 1, \ldots, \delta(y) \). Moreover, let \( D^i \) be the search tree for \( T^i \) obtained from \( D^* \) via Lemma 4. We shall construct a new search tree \( D' \) for \( T \) as follows: the root of \( D' \) is assigned to \( y \); the left tree of \( r(D') \) is the search tree \( D^0 \); in the right tree of \( r(D') \) we build a left path containing nodes corresponding to queries for \( c_1(y), c_2(y), \ldots, c_{\delta(y)}(y) \) and we make \( D^i \) becomes the right subtree of node querying \( c_i(y) \).

It is easy to see that the cost of \( D' \) is at most \( \sum_{i=0}^{\delta(y)} \text{cost}(D^i) + (\Delta(T) + 1) \cdot w(T) \). We claim that, for a suitable choice of \( y \), \( D' \) improves over \( D^* \). For this, let \( S \) be the set of nodes of \( T_x \) which are queried in the path \( r(D^*) \sim v^* \). We distinguish the following cases.

Case 1: \( |S| \geq \frac{2c}{3} \). Set \( y \) as a node in \( T_x \) such that \( |T_y \cap S| \geq |S|/2 \) and \( |T_{c_i(y)} \cap S| \leq |S|/2 \) for every child \( c_i(y) \) of \( y \) and construct \( D' \) as described previously. To find such a node \( y \), traverse \( T_x \) starting at its root and proceeding as follows: if \( u \) is the current node then move to the child \( v \) of \( u \) with largest \( |T_v \cap S| \); the traversal ends when \( |T_y \cap S| \leq |S|/2 \). The parent of the node where the traversal ends is the desired \( y \).

To bound the cost of \( D' \) we first consider the cost of a particular tree \( D^i \). From its construction we have that \( d(r(D^i), l_u) \leq d(r(D^*), l_u) \) for any node \( u \in T^i \). Moreover, for any node \( u \in T^i \) the path \( r(D^*) \sim l_u \) contains \( v^* \) and therefore it contains \( |S \setminus T^i| \) queries to nodes in \( T_y \cap T^i \). Since these nodes were removed in the construction of \( D^i \), we have that for every \( u \in T^i \cap \tilde{T} \)

\[
d(r(D^i), l_u) \leq d(r(D^*), l_u) - |S \setminus T^i| \leq d(r(D^*), l_u) - \frac{|S|}{2},
\]

where the last inequality follows from the definition of \( y \). It follows that

\[
\text{cost}(D^i) \leq \sum_{u \in T_i} d(r(D^*), l_u) \cdot w(u) - \frac{|S| \cdot w(T^i \cap \tilde{T})}{2}.
\]

Combining this bound with our upper bound on the cost of \( D' \) we get that

\[
\text{cost}(D') \leq \text{cost}(D^*) - d(r(D^*), l_y) \cdot w(y) - \frac{|S| w(T - y)}{2} + (\Delta(T) + 1) \cdot w(T).
\]

We claim that actually \( \text{cost}(D') \leq \text{cost}(D^*) - \frac{|S| w(T - y)}{2} + (\Delta(T) + 1) \cdot w(T) \). To see this, first suppose \( y \in \tilde{T} \); then \( d(r(D^*), l_y) \cdot w(y) \geq |S| \cdot w(y) \) and the claim holds. In the other case where \( y \notin \tilde{T} \), the claim follows from the fact \( w(T - y) = w(\tilde{T}) \).

By making use of this claim, the hypothesis on \( |S| \) and the facts that \( w(\tilde{T}) = w(D_{v^*}) > \alpha \cdot w(D^*) \) and \( c \cdot \alpha > 3(\Delta(T) + 1) \), we conclude that \( D' \) improves over \( D^* \), which is a contradiction.

Case 2: \( |S| < \frac{2c}{3} \). We set \( y = x \) and construct \( D' \) as described at the beginning of the proof.

Again, we are trying to reach the contradiction \( \text{cost}(D') < \text{cost}(D^*) \). Recall that \( \text{cost}(D') \leq \sum_{i=0}^{\delta(y)} \text{cost}(D^i) + (\Delta(T) + 1) \cdot w(T) \), so we bound the cost of the trees \( D^i \)’s.

By construction we have that \( \text{cost}(D^0) \leq \sum_{u \in T^0} d(r(D^*), l_u) w(u) \). Now consider some tree \( D^i \) for \( i \neq 0 \). From its construction we have that \( d(r(D^i), l_u) \leq d(r(D^*), l_u) \) for any node \( u \in T^i \). Moreover, for any node \( u \in T^i \cap \tilde{T} \) the path \( r(D^*) \sim l_u \) contains \( v^* \) and therefore it contains at least \( c \cdot |S| \) queries to nodes in \( T - T_x = T^0 \). Then Lemma 4 guarantees that for every \( u \in T^i \cap \tilde{T} \) we have \( d(r(D^i), l_u) \leq d(r(D^*), l_u) - (c - |S|) \).
Weighting these bounds over all nodes in $T$ we have:

$$
\sum_{i=0}^{\delta(y)} \text{cost}(D^l) \leq \sum_{i=0}^{\delta(y)} \sum_{u \in T^i} d(r(D^\ast), l_u)w(u) - \sum_{i=1}^{\delta(y)} \sum_{u \in T^{\gamma(i)}} (c - |S|) \cdot w(u)
$$

$$
= \text{cost}(D^\ast) - d(r(D^\ast), l_x)w(x) - (c - |S|) \cdot (w(\hat{T}) - w(x))
$$

$$
\leq \text{cost}(D^\ast) - (c - |S|) \cdot w(\hat{T}),
$$

where the last inequality is valid because $l_x$ is a descendant of $v^\ast$ in $D^\ast$ so that $d(r(D^\ast), l_x) \geq c$. Thus, by combining the upper bound on $\text{cost}(D')$ with the previous equation in the display we get that $\text{cost}(D') \leq \text{cost}(D^\ast) - (c - |S|) \cdot w(\hat{T}) + (\Delta(T) + 1) \cdot w(T)$. By making use of the hypothesis $|S| < \frac{n}{\Delta}$ and the facts that $w(\hat{T}) = w(D^\ast_{w^\ast}) > \alpha \cdot w(D^\ast) = \alpha \cdot w(T)$ and $c \cdot \alpha > 3(\Delta(T) + 1)$, we conclude that $D'$ improves over $D^\ast$, which gives the desired contradiction.

### E.3 Proof of Theorem 3

The following lemma shows that the bound on the height of the shortest optimal tree holds even when the weight function is not strictly positive.

**Lemma 8.** There is an optimal search tree for $(T, w)$ of height at most $O(\Delta(T) \cdot (\log w(T) + \log n))$.

**Proof.** Consider an optimal search tree $D^\ast$ for $(T, w)$. Notice that for any $v \in D^\ast$, $D^\ast_{v^\ast}$ is an optimal search tree for the subtree of $T$ associated with $v$. So we can employ the Lemma 7 repeatedly and get that for every node $v$ of $D^\ast$ at a level $l = O(\Delta(T) \cdot \log w(T))$, $w(D^\ast_{v^\ast}) = 0$.

Now let $L$ be all the nodes of $D^\ast$ at level $l$. For each $v \in L$ let $D^v$ be the shortest search tree for the subtree of $T$ associated with node $v$. It was proved in [5] that the height of $D^v$ can be upper bounded by $(\Delta(T) + 1) \cdot \log n$. Then we can construct the search tree $D'$ for $T$ as follows: start with $D^\ast$ and for each $v \in L$ replace $D^v$ by $D^v$. Clearly $D'$ has height at most $O(\Delta(T) \cdot (\log w(T) + \log n))$. Moreover, since $w(D^v) = w(D^\ast_{v^\ast}) = 0$ for all $v \in L$, it follows that $D'$ has the same cost as $D^\ast$ and hence is optimal. \qed

**Theorem 3.** Consider an instance $(T, w)$ to our search problem where $\Delta(T) = O(1)$. Then there is an algorithm for computing an optimal search tree for $(T, w)$ that runs in $\text{poly}(n \cdot w(T))$ time. In addition, there is an algorithm for computing an $(1 + \epsilon)$-approximate search tree for $(T, w)$ that runs in $\text{poly}(n/\epsilon)$ time.

**Proof.** The existence of an exact pseudo-polynomial algorithm which runs in $\text{poly}(n \cdot w(T))$ time follows from the discussion presented in Section 3.2 (see From the DP algorithm to an FPTAS). Thus, we only prove the second claim of the theorem, namely, that our search problem admits an FPTAS.

We claim that the following procedure gives the desired FPTAS:

1. Let $W$ be the weight of the heaviest node of $T$, namely $W = \max_{u \in T} \{w(u)\}$. Define $K = \frac{W}{\epsilon w}$ and the weight function $w'$ such that $w'(u) = \lfloor w(u)/K \rfloor$ for every node $u \in T$.

2. Find an optimal search tree $D$ for $(T, w')$ using the pseudo-polynomial algorithm and return $D$.

First we analyze the running time this procedure. Clearly Step 1 takes at most $O(n)$ time. In order to analyze Step 2 let $W' = \max_{u \in T} \{w'(u)\}$ and notice that $W' = \lfloor W/K \rfloor \leq (n^2)/\epsilon + 1$. Thus, $w'(T) \leq nW' \leq (n^3)/\epsilon + n$. Then the pseudo-polynomial algorithm employed in Step 2 runs in $\text{poly}(n \cdot w'(T)) = \text{poly}(n/\epsilon)$. The running time of the whole procedure is then $\text{poly}(n/\epsilon)$, as desired.

Now we argue that the solution $D$ returned by the procedure is $(1 + \epsilon)$-approximate for the instance $(T, w)$. Let us make the weights explicit in the cost function, e.g. we denote by $\text{cost}(D, w)$ and $\text{cost}(D, w')$ the cost of $D$ with respect to the weights $w$ and $w'$. Thus we want to prove that $\text{cost}(D, w) \leq (1 + \epsilon)\text{cost}(D^\ast, w)$, where $D^\ast$ is an optimal search tree for $(T, w)$.
Clearly for each node \( u \in T \) we have \( K \cdot w'(u) \leq w(u) + K \) and hence
\[
K \cdot \text{cost}(D^*, w') \leq \text{cost}(D^*, w) + \sum_{u \in T} d(r(D^*), l_u) \cdot K \leq \text{cost}(D^*, w) + n^2 \cdot K = \text{cost}(D^*, w) + \epsilon \cdot W,
\]
where the last inequality follows from the fact that the distances are trivially upper bounded by \( n \).

Excluding the trivial case where \( T \) is empty, notice that every path in \( D^* \) from \( r(D^*) \) to a leaf has length at least one. Thus, \( \text{cost}(D^*, w) \) can be lower bounded by \( W \), and the previous displayed inequality gives \( K \cdot \text{cost}(D^*, w') \leq (1 + \epsilon) \cdot \text{cost}(D^*, w) \). But since \( w(u) \leq K \cdot w'(u) \) for all \( u \), we have that
\[
\text{cost}(D, w) \leq K \cdot \text{cost}(D, w') \leq K \cdot \text{cost}(D^*, w') \leq (1 + \epsilon) \cdot \text{cost}(D^*, w),
\]
where the second inequality follows from the optimality of \( D \). Therefore, \( D \) is a \((1 + \epsilon)\)-approximate search tree for the instance \((T, w)\), which concludes the proof of the theorem.

\[\square\]

**F**  
Polynomiality of the tree search problem for instances of diameter at most 3

First consider an instance \((T, w)\) of our search problem where \( T \) has diameter two, i.e., it is a star. Let us root the star in its center. Employing a simple exchange argument it is easy to show that the children of \( r(T) \) must be queried according to their weights, in decreasing order. Thus, an optimal search tree for \((T, w)\) can be built based on any sorting algorithm in \( O(n \log n) \).

Now assume \( T \) has diameter 3. Notice that the only possible structure for \( T \) is the following: there are two nodes \( r \) and \( r' \) joined by an edge and all other nodes are either adjacent to \( r \) or to \( r' \). In order to define the questions, let us take \( r \) as the root. Let \( l \) (\( l' \)) be the heaviest leaf among the children of \( r \) (\( r' \)). It should not be difficult to see that the root of any optimal search tree must query one of the nodes in the set \( \{r', l, l'\} \). This can be proved using a simple exchange argument. If \( r(D) \) is assigned to \( r' \) then its right subtree is an optimal search tree for \( T_{r'} \) and its left subtree is an optimal search tree for \( T - T_{r'} \). If \( r(D) \) is assigned to \( l \) then its right subtree is a leaf assigned to \( l \) and its right subtree is an optimal search tree for \( T - l \). Analogously, when \( r(D) \) is assigned to \( l' \) its right subtree is an optimal search tree for \( T - l' \). Finally, notice that in the first case, both \( T_{r'} \) and \( T - T_{r'} \) have diameter at most 2.

Consider the recursion tree of the above procedure; notice that every subproblem \((T', w)\) has a specific structure: \( T' \) is the subtree of \( T \) induced by nodes \( r, r', 9th \) heaviest leaf-children of \( r \) and \( jth \) heaviest children of \( r' \) (for some \( i, j \)). Employing a Dynamic Programming strategy together with an \( O(n \log n) \) preprocessing for the two stars centered at \( r \) and \( r' \), it is not difficult to see that each of these \( O(n^2) \) problems can be solved in \( O(1) \) time. This gives an \( O(n^2) \) algorithm for finding an optimal search tree for \((T, w)\).
Figures

Figure 1: (left) The input tree T; (right) a search tree D for T

\[ X_1 = \{a, b, c\} \]
\[ X_2 = \{b, c, d\} \]
\[ X_3 = \{b, e, f\} \]
\[ X_4 = \{d, e, f\} \]

Figure 2: The tree obtained from instance \( I = (\{a, b, c, d, e, f\}, \{X_1, X_2, X_3, X_4\}) \) of 3-bounded X3C.

Figure 3: The two possible configurations we use for the part of the search tree that concerns the subtree \( T_i \) and the leaf \( a_i \) and a sequential search tree for \( T_i \).

23
Figure 4: Realization $D^A$ (left), and the (optimal) Realization w.r.t. the exact cover $\{X_1, X_4\}$ (right)—in bold are the questions involved in the configuration changes. Only the leaves associated to nodes of $T$ with non-zero weights are shown here.
Figure 5: The tree $T^b$ obtained from the instance $I = (\{a, b, c, d, e, f\}, \{X_1, X_2, X_3, X_4\})$ of 3-bounded X3C.

Figure 6: (a) PLP $P$ with partition $U = \{U^f, U^o\}$ indicated. The blank nodes are unassigned and the black ones are blocked. (b) PLP's $P^f$ and $P^o$. (c) The optimal EST's $D^f$ and $D^o$ and (d) the resulting EST $D^U$ constructed by taking the ‘union’ of $D^f$ and $D^o$. 

\[ X_1 = \{a, b, c\} \]
\[ X_2 = \{b, c, d\} \]
\[ X_3 = \{b, e, f\} \]
\[ X_4 = \{d, e, f\} \]

$\Pi = <\pi_1, ..., \pi_{m+n}> = <a, b, c, X_1, d, X_2, e, f, X_3, X_4>$

$Z_1 = \{X_1\}, Z_2 = \{X_2\}, Z_3 = \{X_3, X_4\}$
Figure 7: (a) PLP $P$ (b) PLP $P_{i,t}$. (c)-(d) Construction of $D_{i,t}$—given in picture (d)—starting from an EST $D'$ given in picture (c) —for $P_B^B(\{T_{c_1(v)}, \ldots, T_{c_δ(v)}\}, P_{i,t})$. 