Randomized Binary and Tree Search under Pressure

Agustín Caracci[*] Christoph Dürr[†] José Verschae[‡]

September 14, 2025

Abstract

We study a generalized binary search problem on the line and general trees. On the line (e.g., a sorted array), binary search finds a target node in $O(\log n)$ queries in the worst case, where n is the number of nodes. In situations with limited budget or time, we might only be able to perform a few queries, possibly sub-logarithmic many. In this case, it is impossible to guarantee that the target will be found regardless of its position. Our main result is the construction of a randomized strategy that maximizes the minimum (over the target position) probability of finding the target. Such a strategy provides a natural solution where there is no apriori (stochastic) information of the target's position. As with regular binary search, we can find and run the strategy in $O(\log n)$ time (and using only $O(\log n)$ random bits). Our construction is obtained by reinterpreting the problem as a two-player (*seeker* and *hider*) zero-sum game and exploiting an underlying number theoretical structure.

Furthermore, we generalize the setting to study a search game on trees. In this case, a query returns the edge's endpoint closest to the target. Again, when the number of queries is bounded by some given k, we quantify a *the-less-queries-the-better* approach by defining a seeker's profit p depending on the number of queries needed to locate the hider. For the linear programming formulation of the corresponding zero-sum game, we show that computing the best response for the hider (i.e., the separation problem of the underlying dual LP) can be done in time $O(n^2 2^{2k})$, where n is the size of the tree. This result allows to compute a Nash equilibrium in polynomial time whenever $k = O(\log n)$. In contrast, computing the best response for the hider is NP-hard.

1 Introduction

Searching for an object in an ordered list with n elements is one of the most fundamental tasks in computer science. Binary search, one of the most basic discrete algorithms, yields a search strategy that finds the target item with $O(\log(n))$ comparisons, regardless of its position. However, in situations when comparing item values is expensive, we are faced with the problem of searching under a potentially sub-logarithmic budget of k comparisons. For small values of k, there will be positions of the target value where the strategy will fail. By using a randomized strategy, we can aim to find the target with certain minimum probability, independently of its position. More

[*]Pontificia Universidad Católica de Chile, Institute for Mathematical and Computational Engineering, Chile. Email: juan.caracci@uc. Supported by Fondecyt-ANID Nr. 1221460, ANID.

[†]Sorbonne University, CNRS, LIP6, France. `https://www.lip6.fr/Christoph.Durr`. Email: Christoph.Durr@lip6.fr. Work partially conducted while C.D. was affiliated with Universidad de Chile, CMM. Partially supported by the Center for Mathematical Modeling grant FB210005 and the grants ANR-19-CE48-0016, ANR-23-CE48-0010 from the French National Research Agency (ANR).

[‡]Pontificia Universidad Católica de Chile, Institute for Mathematical and Computational Engineering, Chile. `https://sites.google.com/site/jverschae/`. Email: jverschae@uc.cl. Partially supported by Fondecyt-ANID Nr. 1221460, ANID and by the Center for Mathematical Modeling grant FB210005, Basal, ANID.

precisely, we study the problem of devising a randomized strategy that maximizes the worst-case (with respect to the position of the target) probability of finding the target.

The described setting can be equivalently posed as a two-player zero-sum game played on a line graph $G = (V, E)$ with vertex set $V = \{0, 1, \dots, n-1\}$ and edge set $E = \{\{v, v+1\} : 0 \leq v \leq n-2\}$. The first player, the *hider* chooses a node $v^* \in V$ to hide (i.e., the target). The second player, the *seeker* must decide on a search strategy, i.e., an adaptive sequence of edges to query that aims to find the hidden node v^*. If edge $\{v, v+1\}$ is queried, the seeker learns whether $v^* \leq v$. The game finishes after k queries or after the seeker can univocally find the position of v^*. If the seeker finds v^* within the given number of queries she gets a profit of 1 and the seeker obtains a profit of -1 (i.e., she pays a cost of 1). If after the k queries the seeker cannot certify the position of v^*, then both player get a profit of 0. We are interested in finding a mixed (i.e. randomized) strategy for each player that yields a Nash equilibrium. As this is a zero-sum game, in a Nash equilibrium the seeker chooses a randomized strategy that maximizes $\min_{v^* \in V} P(v^*)$, where $P(v^*)$ denotes the probability that the seeker finds the target given that the hider chooses v^* to hide.

This fundamental problem is motivated by several questions related to the search of substances in water networks. For example, a new trend of water surveillance became prominent during the COVID pandemic, where PCR tests were taken within a sewage water network (normally a tree) in order to efficiently detect virus outbreaks [2, 3, 16, 21, 23, 25]. Similar strategies have been proposed to detect the presence of illicit drugs [27], pollution [15] and even illicit explosives [1]. Observe that if the water network is a path and our aim is to find a single infected node, taking a PCR test on a given edge (or node) will tell us whether the infected node is upstream or downstream. Crucially, in all these scenarios, the number of comparisons is constrained by the time that the sought substance stays in the network. Arguably, these settings induce models where the number of queries is sub-logarithmic, or even constant, with respect to the number of nodes of the network. Hence, maximizing the worst-case probability of finding the infected node is particularly meaningful, especially if there is no known probabilistic data on the source of the target. On the other hand, the (equivalent) game theoretical perspective gives a relevant model where there is an adversary that chooses the location of the target deliberately.

Inspired by these scenarios, we further extend our model in two directions. First, we consider the case in which the graph $G = (V, E)$ is not only a path, but a general tree. As before, the seeker queries edges. If $e \in E$ is queried, she obtains as an answer the connected component of $G - e := (V, E \setminus \{e\})$ that contains v^*. As a second extension, we further consider a time-dependent non-increasing profit function $p : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$, where $p(t) = 0$ of $t > k$. If the seeker finds the hider within t queries, she obtains a profit of $p(t)$ and the hider perceives a cost of $p(t)$ (or, equivalently, a profit of $-p(t)$). Observe that we obtain the same problem as before if $p(t) = 1$ for $t \leq k$, which we call the *unit profit* case. As before, the objective is to efficiently find a mixed Nash equilibrium.

Related Literature Previous literature mostly focused on minimizing the worst case number queries for finding the target. When the target is hidden in a tree, Lam and Yue [20] (see also [14]) provided a linear time algorithm which generates an optimal search tree for the edge query model, a result that was later rediscovered in a sequence of papers, including the works by Ben-Asher et al. [5], Onak and Parys [24], and Mozes et al. [22]. A more general setting was later considered by Cicalese et al. [12] where each edge has a different cost. The objective is to find the target with the worst case minimum total cost. This version turns out to be strongly NP-hard and admits an approximation algorithm with a slightly sublogarithmic guarantee.

An alternative model considers queries on the vertex of the tree. If the target is not in the queried vertex, one learns which of the neighbors is closest to the target. For this model, Schäffer provided a linear time algorithm generating an optimal search tree [26]. Minimizing the number of vertex queries on an arbitrary undirected graph becomes quite intriguing. There is an $O(m^{\log n} \cdot n^2 \log n)$ time algorithm, where m is the number of edges and n the number of vertices. The quasi-polynomial part is necessary, as there is no $O(m^{(1-\epsilon)\log n})$ time algorithm under the Strong Exponential-Time Hypothesis (SETH) [17]. For directed acyclic graphs, a vertex query returns 3 types of answers: the target is on the queried vertex, the target is reachable from the queried vertex, or it is not. In this model, minimizing the number of queries is NP-complete [9].

A different line of research considers a known probability distribution of the position of the target, and aims to find a search strategy (either for node or edge queries) that minimizes the expected number of queries until the target is found. Knuth [19] showed that dynamic programming can solve this problem on the line with running time $O(n^2)$. For the more complicated problem on trees with the edge query model, Cicalese et al. [10, 11] showed that finding an optimal search strategy for a tree is NP-hard. Moreover, it admits a 1.62-approximation algorithm and even an FPTAS on tree of bounded degrees. On the other hand, for node queries, the complexity of the problem is still open. Recently, Berendsohn and Kozma [7] showed that the problem admits a PTAS, and Berendsohn et al. [6] analyzed the algorithm that iteratively select the centroid of the tree. Besides given a fast output-sensitive algorithm for finding the centroid, they show that the obtained strategy is a 2-approximation and that the analysis is tight even when the target distribution is uniform.

Our Contribution We study two scenarios: *(i)* For the unit profit case in the line, we provide an algorithm that computes the optimal strategy for the seeker in $O(\log n)$ time; *(ii)* if G is a general tree and the profit p is arbitrary, we show that we can find a Nash equilibrium in time 2^kpoly(n) while computing the best-response for the seeker is NP-hard.

Our main result is *(i)*. Our algorithm samples a search strategy in time $O(\log n)$ (using $O(\log n)$ random bits). Afterwards, choosing the next node to query takes constant time in each time step. The solution is based on picking a random interval, on which we will perform binary search. The interval is chosen uniformly at random from a set constructed with a greedy algorithm based on modular arithmetics (see Figure 3). This construction naturally connects to Bezout's identity and the Extended Euclidean algorithm, which we exploit to obtain the claimed running times. On the other hand, we provide a construction that yields an optimal strategy for the hider, which we then use to show that both strategies (hider and seeker) are optimal by linear programming duality. For some regimes of values of n and k, the optimal hider's solution requires an intricate construction as she needs to unevenly distribute probability mass to avoid giving the seeker an advantage (see Figure 4).

For *(ii)*, where G is an arbitrary tree and p is any non-increasing profit function, we again consider the corresponding zero-sum game as a linear program (LP), which has an exponential number of variables. We show that the separation of the dual problem — the problem of computing the best-response of the hider — can be solved in time $O(n^2 2^{2k})$. In contrast, when k is part of the input the separation problem is NP-hard, even if the diameter or the degree of the tree is constant (see Theorem 4.1), which justifies the exponential dependency on k[1]. The separation problem of the dual takes as input a distribution of the target node over V. It consists of finding a search strategy with at most k queries that maximizes the expected profit of the seeker. We devise a dynamic program that solves this problem. It is based on the concept of *visibility vectors* [24, 20, 14], introduced for the problem of minimizing the time to find the target in the worst-case. By rooting the tree, we can apply a bottom-up approach to extend a solution for each possible visibility vector.

[1]Observe, however, that this does not imply that computing a Nash equilibrium is NP-hard.

The exponential dependency on k comes from the fact that the number of different visibility vectors is exponential in k. Together with the Ellipsoid method, we obtain an algorithm to compute a Nash equilibrium in time $2^k\text{poly}(n)$.

Observe that in regimes where the budget is logarithmic, $k \leq C \cdot \log_2(n)$, then our algorithm for the separation problem runs in polynomial time $O(n^{2+2C})$. Moreover, k is logarithmic in many natural cases. For example, if G has maximum degree d and we are in the unit profit case, then there are search strategies that can unequivocally find the target within $O(d \log n)$ queries [4, 17]. Therefore, if d is a constant, our model only makes sense when k is at most logarithmic, as otherwise we have a strategy that finds the hider with probability 1. Hence, for these cases we can assume that $k \leq C \cdot \log_2(n)$ and hence our algorithm takes polynomial time $O(n^{2+2C})$. In particular, this yields a running time of $O(n^4)$ for the separation problem when G is a line; which is significantly worse than our dedicated solution.

In Section 2 we introduce the problem formally. Our main results are given in Section 3, where we show our solution for the case that G is a path. Finally, Section 4 shows the dynamic program for the general case. Some of the technical proofs are deferred to the appendix.

2 Problem Definition

For an arbitrary graph H, we denote $V(H)$ and $E(H)$ its node and edge set, respectively. We will represent edges as sets $\{u, v\} \in E$, and we will use the shortcut $\{u, v\} = uv$ if appropriate.

Let $G = (V, E)$ be a tree with n nodes. To represent a search strategy, consider a rooted binary out-tree $T = (N, D)$, where each internal node $\nu \in N$ is labeled with an edge $e(\nu) \in E$. By definition, the root $\rho \in N$ is labeled, unless $|N| = 1$. The edge $e(\rho) = uv$ corresponds to the first edge queried by the strategy. Let G_u (resp. G_v) denote the connected component of $G - e := (V, E \setminus \{e\})$ that contains u (resp. v). One child of ρ is associated to G_u, and the other child to G_v. The output of the query points to child of ρ associated to the connected component that contains the target. The search is then resumed in said child. Based on this notation, we give a recursive definition of a search strategy. We say that T with root label $e(\rho) = uv \in E$ is a search strategy for $G = (V, E)$ if the subtrees of ρ in T, denoted by T_u and T_v, are also search strategies for G_u and G_v, respectively.

Finally, $T = (N, D)$ is a search strategy for any tree G if $|N| = 1$.

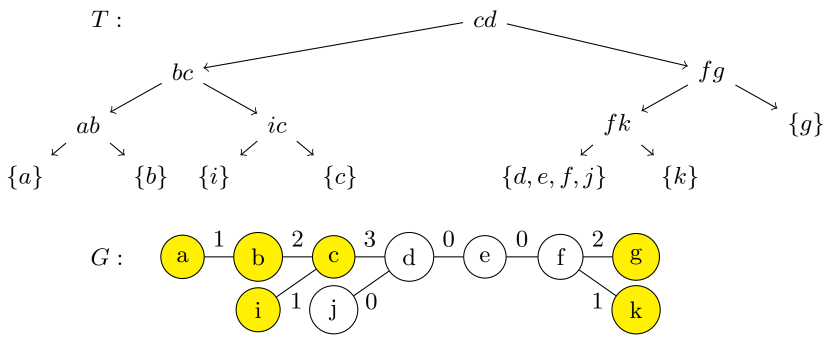


Figure 1: Example of a search tree T over a tree G. Leafs ν of T are labeled with $V(\nu)$. Covered vertices in G are marked in yellow. Edge labels of G will be explained in Section 4.1

Let T be a search strategy for $G = (V, E)$. Observe that each node ν of T can be naturally associated to a set $V(\nu) \subseteq V$ that potentially contains the target node v^*. More precisely, let us define $V(\rho)$ as V. If $e(\rho) = uv$, we define recursively $V(\rho_u) = V(G_u)$ and $V(\rho_v) = V(G_v)$, where

ρ_u (resp. ρ_v) is the root of T_u (resp. T_v) and $V(G_u)$ (resp. $V(G_v)$) is the vertex-set of G_u (resp G_v). Hence, at the moment that we query according to node $\nu \in N$, the set $V(\nu)$ corresponds to the smallest node set in G for which we can guarantee that $v^* \in V(\nu)$.

Let $L \subseteq N$ be the set of leaves of T. It is easy to see that $\{V(\lambda) : \lambda \in L\}$ defines a partition into connected components of V. We say that node $v \in V$ is *covered* by T if the singleton $\{v\}$ belongs to $\{V(\lambda) : \lambda \in L\}$. The set of nodes covered by T is denoted by $C(T) \subseteq V$. Observe that if the hider chooses v^*, then applying the search strategy T we can assert that v^* is the target if and only if $v^* \in C(T)$.

Finally, the *height* of a search strategy T is the height of the underlying binary tree, i.e., the length of the longest path from the root ρ to a leaf. For a fixed tree G and budget $k \in \mathbb{N}$, we denote by $\mathcal{T}_k$ the set of all the search strategies for G of height at most k.

Let Δ_k be the set of distributions supported on $\mathcal{T}_k$, that is, $\Delta_k := \{x : x \geq 0 \text{ and } \sum_{T \in \mathcal{T}_k} x_T = 1\}$. Our aim is to find a vector $x \in \Delta_k$ that maximizes the worst-case probability of finding the target $v^* \in V$, that is,

$$\max_{x \in \Delta_k} \min_{v^* \in V} \sum_{T \in \mathcal{T}_k : v^* \in C(T)} x_T. \tag{1}$$

More generally, let $p : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be a non-increasing function, where $p(t)$ represents the profit of finding the target with exactly t queries, and $p(t) = 0$ for $t > k$. Our most general model considers the maximization of the worst-case expected profit,

$$\max_{x \in \Delta_k} \min_{v^* \in V} \sum_{T \in \mathcal{T}_k} x_T \cdot p(h_T(v^*)), \tag{2}$$

where $h_T(v^*)$ is $k+1$ if $v^* \notin C(T)$ (and hence $p(h_T(v^*)) = 0$) and, otherwise, it equals the distance from the root ρ to the leaf λ such that $V(\lambda) = \{v^*\}$.

We can interpret this problem as a two-player zero-sum game as follows. The first player, the *seeker*, selects a search strategy $T \in \mathcal{T}_k$. The second player, the *hider*, selects a node $v \in V$. Hence, for a given pure strategy pair (T, v), the seeker obtains a profit of $p(h_T(v))$ while the hider incurs a cost of $-p(h_T(v))$ (or a negative profit of $p(h_T(v))$. A mixed-strategy for the seeker is a probability vector $x \in \Delta_k$, where x_T represents the probability of selecting the search strategy T. A mixed-strategy for the hider is a vector $y \in \Delta_V := \{y : y \geq 0 \text{ and } \sum_{v \in V} y_v = 1\}$. For a pair (x, y), the expected profit of the seeker (cost of the hider) is $p(x, y) = \sum_{T \in \mathcal{T}_k} \sum_{v \in V} y_v \cdot x_T \cdot p(h_T(v))$. A pair $(x, y) \in \Delta_k \times \Delta_V$ is said to be a Nash equilibrium if no player can unilaterally improve their profit or cost, that is,

$$p(x, y) \geq p(x', y) \quad \text{for all } x' \in \Delta_k \qquad \text{and} \qquad p(x, y) \leq p(x, y') \quad \text{for all } y' \in \Delta_V.$$

Von Neumann's minimax theorem [28], a basic game-theoretic fact, states that a pair (x, y) is a Nash equilibrium if and only if x defines an optimal solution to the LP,

$$\text{[P]} \qquad \max \left\{ z : \ z \leq \sum_{T \in \mathcal{T}_k} x_T \cdot p(h_T(v)) \text{ for all } v \in V, \text{ and } x \in \Delta_k \right\},$$

and y is an optimal solution to

$$\text{[D]} \qquad \min \left\{ t : \ t \geq \sum_{v \in V} y_v \cdot p(h_T(v)) \text{ for all } T \in \mathcal{T}_k, \text{ and } \ y \in \Delta_V \right\}.$$

Observe that [P] and [D] are dual LPs, and hence they attain the same optimal value. We refer to this value as u^*, the *optimal profit* or the *value of the game*. Moreover, as in an optimal solution for [P] the value of z is simply the minimum value of $\sum_{T\in\mathcal{T}_k} x_T \cdot p(h_T(v))$ over all $v \in V$, then [P] is a restatement of our original problem in Equation (2). For a given probability vector $x \in \Delta_k$ we say that $\min_{v\in V} \sum_{T\in\mathcal{T}_k} x_T \cdot p(h_T(v))$ is its objective value. Similarly, for a vector $y \in \Delta_V$ we say that $\max_{T\in\mathcal{T}_k} \sum_{v\in V} y_v \cdot p(h_T(v))$ is its objective value.

3 Search on a line

In this section we focus on the special case where $G = (V, E)$ is a line, that is, $V = \{0, 1, \ldots, n-1\}$ and $E = \{\{v, v+1\} : 0 \leq v \leq n-2\}$. Also, we restrict ourselves to unit profit functions, where $p(t) = 1$ for all $t \leq k$ and $p(t) = 0$ if $t > k$. To avoid trivial border cases, we assume that $k \geq 2$. Moreover, we can assume that $n > 2^k$, as otherwise there exists a single search strategy, namely the usual binary search, that finds the target in every single node, and hence the value of the game is trivially 1. The aim of this section is to compute a highly structured Nash equilibrium.

We start by characterizing the sets $C \subseteq V$ that arises as covered sets of a search strategy $T \in \mathcal{T}_k$. Then, we restrict the set $\mathcal{T}_k$ to a smaller set of strategies, which we call *efficient*. Using our characterization of covered sets together with LP duality, we will able show that there exists an optimal solution for the seeker that only uses such strategies.

Covered Sets and Efficient Strategies For a given search strategy T, recall that $C(T)$ denotes the set of covered nodes. Most of the technical work we do is on showing that an randomized optimal solution only selects search strategies where $C(T)$ corresponds to a set of consecutive nodes modulo n. For $u, v \in \mathbb{N}$, let $[u, v]_r$ denote the set $\{w \mod r : u \leq w \leq v\}$ if $u \leq v$, and $[u, v]_r = \emptyset$ otherwise. We say that $[u, v]_r$ is an *interval modulo* r, or simply an *interval*. To avoid unnecessary notation we omit the subscript r when $r = n$, i.e., $[u, v] = [u, v]_n$. Moreover, for an integer $\ell \leq r$, we denote by $[v \oplus \ell]_r = [v, v+\ell-1]_r$, that is, the interval that starts at v of length ℓ.

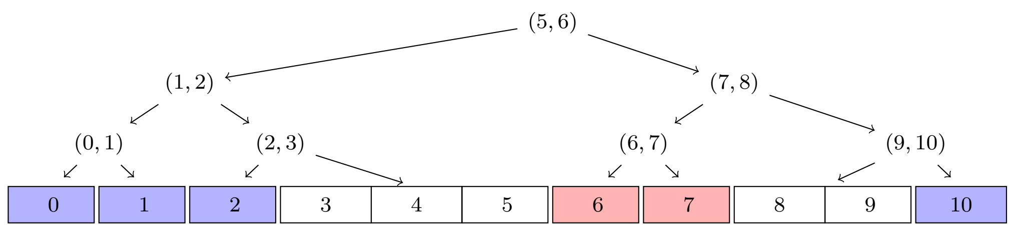
It will be particularly useful to understand the sets that appear as a cover set of a search strategy $T \in \mathcal{T}_k$. Let $c := 2^k - 2$. Observe that a search strategy that aims to cover a single interval in $[1, n-2]$, is able to cover an interval of length c: just apply a binary search restricted to a given interval. On the other hand, if the strategy covers a single interval that contains either 0 or $n-1$ (or both), it will be able to cover $c+1$ nodes. Notice also that there cannot be a strategy that covers exactly $[1, c]$, as this immediately implies that 0 is covered, similarly with $[n-c-1, n-2]$. Strategies that cover more than one interval cover a smaller number of nodes. For example, if it covers two disjoint intervals (that do not contain 0 or $n-1$), the number of covered nodes will be $c-1$. The next proposition formalizes this intuition by giving a characterization of maximal cover sets $C(T)$ (that is, the ones such that there is no $T' \in \mathcal{T}_k$ with $C(T) \subset C(T')$) for search strategies $T \in \mathcal{T}_k$.

Proposition 3.1. *Denote $c = 2^k - 2$. Let $C = [u_1 \oplus \ell_1] \cup \ldots \cup [u_s \oplus \ell_s] \subseteq V$ be a set where $0 \leq u_1 < \ldots < u_s \leq n-1$ (and s is minimal). The set C is a maximal covered set $C(T)$ for some $T \in \mathcal{T}_k$ if and only if*

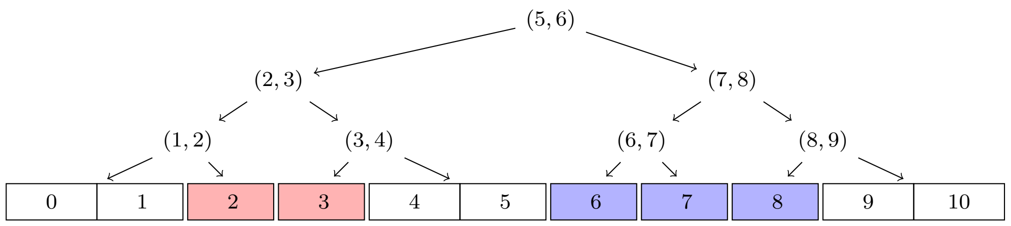
i) $u_1 \neq 1$ *and* $u_s + \ell_s - 1 \neq n-2$,

ii) $(u_{t+1} - (u_t + \ell_t)) \geq 2$, *for all* $t \in \{1, \ldots, s-1\}$ *and* $u_1 - (u_s + \ell_s - n) \geq 2$, *and*

iii) $\sum_{t=1}^{s} \ell_t = \begin{cases} c+1-s & \text{if } \{0, n-1\} \cap C = \emptyset, \\ c+2-s & \text{otherwise.} \end{cases}$



(a) Maximal covered set with two intervals, one of which intersects with vertex 0 and vertex 10. Interval $[6 \oplus 2]$ is colored red and interval $[10 \oplus 4]$ is colored blue. The total number of covered vertices is 6.



(b) Maximal covered set with 2 intervals. Interval $[2 \oplus 2]$ is colored red and interval $[6 \oplus 3]$ is colored blue. Even though the number of intervals is also 2, the number of covered vertices is 5 because neither vertex 0 nor vertex 10 is covered.

Figure 2: An example of two maximal covered sets and their respective search trees on a line of length $n = 11$ and a budget of $k = 3$ queries. Covered vertices are colored, and dashed arrows point to leaves of the search tree with two or more vertices. Figure 2a shows the first case of condition (iii) of Proposition 3.1 and Figure 2b shows the second case.

The first two conditions in the proposition encompass the fact that the vertices not covered by a search strategy always have at least one neighbor that is also not covered. The third condition says that there is a trade-off between the number of intervals needed in the description of C and the number of vertices in C. Finally, it also says that if C contains 0 or $n-1$ (or both), we gain one extra covered node. See Figure 2 for an illustration.

Of particular interest are search strategies with a single interval, i.e., with $s = 1$, as they maximize the number of covered elements. We call such strategies *efficient*. The following corollary is a direct consequence of Proposition 3.1.

Corollary 3.1. *For every $v \in V \setminus \{1\}$ there exists a search strategy T_v that covers,*

$$C(T_v) = \begin{cases} [v \oplus (c+1)] & \text{if } \{0, n-1\} \cap [v \oplus (c+1)] \neq \emptyset, \\ [v \oplus c] & \text{otherwise.} \end{cases}$$

Observe that there is no efficient strategy (of length c) that covers an interval starting at 1, as we could then obtain a strategy of length $c+1$ that also covers 0. For any $v \in V \setminus \{1\}$, we denote by T_v the search strategy that has a covering set as in the corollary.

The Seeker's Strategy. We will start by constructing a strategy $x = (x_T)_{T \in \mathcal{T}_k} \in \Delta_k$ for the seeker given by a greedy algorithm that defines the strategies of x. Given $\mathcal{X}$ which denotes the support of x, we will define the probability vector x uniformly over $\mathcal{X}$, i.e., $x_T = \frac{1}{|\mathcal{X}|}$ if $T \in \mathcal{X}$ and 0 otherwise.

Intuitively, our aim is to cover all nodes as evenly as possible with efficient strategies, that is, by the same number of strategies in $\mathcal{X}$. Consider for example the case $n = 12$ and $k = 3$ (i.e.,

$c = 6$), depicted in Figure 4. Imagine that we choose T_0 to be in $\mathcal{X}$. This alone yields an objective of 0, and hence a natural choice is to consider $\mathcal{X} = \{T_0, T_7\}$. However, this implies that nodes in $\{0, 1\}$ are covered twice while the rest is covered only once. This imbalance suggests we should add more strategies, the most natural being T_2, yielding an imbalance on nodes in $\{8, 9, 10, 11\}$. By continuing adding strategies to $\mathcal{X}$ in a greedy manner, we obtain the solution in Figure 4, yielding an objective value of $\min_v \sum_{T:v\in C(T)} x_T = 5/9$. These examples suggests Algorithm 1, that can be interpreted as a greedy rectangle packing algorithm, see Figure 3 and Figure 4.

```
1 Set $v = c + 1$ and $\mathcal{X} = \{T_0\}$;
2 while $v \notin \{0, 1\}$ do
3     $\mathcal{X} = \mathcal{X} \cup \{T_v\}$
4     $v = (v + c) \mod n - 1$.
5 end
6 Set $x_T = 1/|\mathcal{X}|$ if $T \in \mathcal{X}$ and $x_T = 0$ otherwise.
```
Algorithm 1: Greedy algorithm for the seeker's strategy

Observe that in Line 4 of Algorithm 1 we take $v + c$ modulo $n - 1$ instead of modulo n, which might be counter-intuitive. The reason for this choice is that if we take away node $n - 1$, then the cardinality (length) of each set $C(T_v) \setminus \{n - 1\}$ for every $v \notin \{0, 1\}$ is c. This avoids the case distinction of Corollary 3.1..

Given a set $\mathcal{X}$, the objective value of x is $\min_v \sum_{T:v\in C(T)} x_T$. If we reach the case during the while loop where $v = 0$, then the probability of covering vertex v^* is $|\{T \in \mathcal{X} : v^* \in C(T)\}|/|\mathcal{X}|$, which is the same for every v^* as each element is covered by the same number of search strategies in $\mathcal{X}$. This is the reason why $v = 0$ terminates the while loop. If we reach the situation where $v = 1$ in the while loop, then we should also stop since T_1 is not well defined. We observe that all nodes, except maybe for node 0, are covered the same number of times by $\mathcal{X}$, i.e., $|\{T \in \mathcal{X} : 1 \in C(T)\}| = |\{T \in \mathcal{X} : v \in C(T)\}|$ for all $v \in V \setminus \{0\}$. We define this quantity as $h := |\{T \in \mathcal{X} : 1 \in C(T)\}|$ and we say that it corresponds to the *height* of $\mathcal{X}$. Similarly, we denote the cardinality of $\mathcal{X}$ as $w = |\mathcal{X}|$. Observe that with these parameters each node, except maybe for node 0, is covered by h many sets $C(T_v)$ for $T_v \in \mathcal{X}$. Hence, the objective value of solution x is $\frac{h}{w}$. The following lemma yields an explicit relationship between w and h, and shows how to compute these values with a faster algorithm. In particular, it relates w and h with the $\gcd(c, n - 1)$, the greatest common divisor of c and $n - 1$, and Bezout's identity. Running times in the next lemma are considered in the RAM model.

Lemma 3.1. *Algorithm 1 terminates in finite time. If x denotes the output of Algorithm 1, then its objective value is $\frac{h}{w}$, where w is the cardinality and h is the height of $\mathcal{X}$. Moreover, $h, w \in \mathbb{N}$ are the numbers that satisfy*

$$h(n-1) - wc = \begin{cases} 1 & \text{if } gcd(c, n-1) = 1 \text{ and,} \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $0 < w \le n - 2$ is minimal. In particular, we can compute w, h, and the objective value of x in $O(\log n)$ time. Moreover, if c and $n - 1$ are not coprime the objective value simplifies to $\frac{h}{w} = \frac{c}{n-1}$ and $h = \frac{c}{gcd(c,n-1)}$ and $w = \frac{n-1}{gcd(c,n-1)}$.

Proof. TOPROVE 0 □

With this lemma, we observe that we can perform a search following x in logarithmic time, even without the need of running Algorithm 1: simply compute h and w with the Extended Euclidean Algorithm and sample the t-th element in $\mathcal{X}$ uniformly.

Corollary 3.2. *Let x be the output of Algorithm 1. We can sample a tree T with probability x_T in time $O(\log n)$ and using $O(\log n)$ random bits, without the need to compute x. After choosing T, each query of the tree can be determined in constant time.*

The proof of this result is provided in the Appendix. In order to show that the constructed solution x is indeed optimal, we construct a dual y with the same objective value. Hence, by weak duality this implies that the pair (x, y) is a Nash equilibrium. As suggested by the previous lemma, we should distinguish whether $n - 1$ and c are coprime or not.

Although our construction of the dual solution does not explicitly rely on complementary slackness, the intuition for our approach came from thinking about certain constraints that this concept imposed on the dual variables. In particular, observe that the solution of the seeker induces "segments" of vertices that are covered by the same subset of search strategies in $\mathcal{X}$. See, for example, vertices 3 and 4 in Figure 3 or vertices 5 and 6 in Figure 4. There are w of these segments (ignoring vertex 0 in the not coprime case), and complementary slackness implies that each of them should have a mass of $1/w$ in an optimal solution for the hider (albeit it does not imply that the mass should be uniformly distributed within each segment). The solution we construct shares this structure: the line is split into w segments, and each of them gets a probability of $1/w$ in total. How these segments are chosen and how the mass is distributed within them differs significantly depending on whether $n - 1$ and c are coprime or not. However, in both cases the solution has the property that any pair of disjoint intervals covers at most the same probability mass as an interval obtained by gluing them together according to Proposition 3.1. This then implies that the best-response strategy of the seeker is achieved with an efficient strategy, for which the covered mass will be precisely h/w.

We start with the not coprime case, which turns out to admit a simpler proof since the segments can be chosen in a more regular manner. On the other hand, the coprime case is significantly more challenging from a technical point of view as y cannot be constructed so regularly.

Optimality if $n-1$ and c are not coprime. If $n-1$ and c are not coprime, Lemma 3.1 implies that the objective value of the solution x computed by Algorithm 1 equals $\frac{c}{n-1}$. In what follows we will explicitly define a solution y for the dual [P] with the same objective function, thus implying optimality of x and y. The dual solution that we will analyze is defined as

$$y_v = \begin{cases} 0 & \text{if } v = k \cdot d \text{ for some } k \in \mathbb{N}_0, \\ \frac{1}{w(d-1)} & \text{otherwise,} \end{cases} \tag{4}$$

for all $v \in V$, where $w = (n-1)/d$ and $d = \gcd(c, n-1)$. We can interpret this solution by thinking of the vertices $\{1, \ldots, n-1\}$ as divided into w segments, each containing vertices $kd+1, \ldots, kd+d$ for $k = 0 \ldots, w-1$. Within each segment, its probability of $1/w$ is distributed uniformly among its first $d-1$ vertices (see Figure 3). Although this interpretation may seem somewhat contrived, it serves to highlight the connection to the construction of the coprime case, which relies heavily on the use of segments.

In what follows, we will show that y is indeed a probability distribution and achieves an objective value of $\frac{c}{n-1}$.

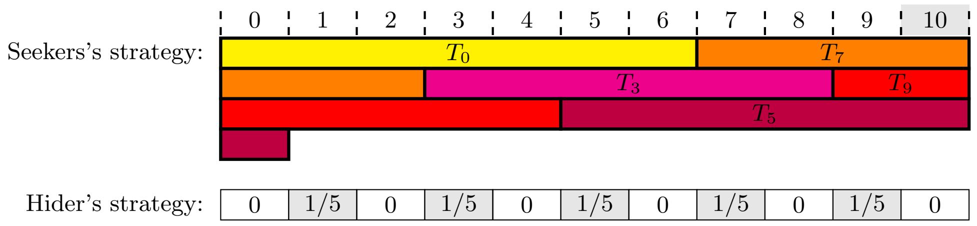


Figure 3: Example of the game on a line of length $n = 11$ with $k = 3$ queries, where $\gcd(c, n-1) = 2$. The searcher choose uniformly one out of 5 strategies. Each strategy is depicted with a distinct color marking the covered cells. Here all cells are covered with probability 3/5, except the first cell which gets more coverage. By appropriately shifting the strategies the seeker can choose to overcover exactly one of the nodes 0,2,4,6,8 or 10. These are exactly the nodes which the hider avoids, choosing uniformly among all other nodes. The value of the game is 3/5.

Theorem 3.1. *Assume that $gcd(c, n-1) = d > 1$. Then, $y := (y_v)_{v \in V}$, as defined in Equation (4), is a feasible solution for [D] and has objective value $\frac{c}{n-1}$. Hence, the pair (x, y) is a Nash equilibrium where x is the solution output by Algorithm 1.*

For a set $S \subseteq V$, we denote $y(S) = \sum_{v \in S} y_v$. Observe that by weak duality the objective value of y cannot be smaller than $\frac{c}{n-1}$. Hence it suffices to show that $y(C(T)) \leq \frac{c}{n-1}$ for all $T \in \mathcal{T}_k$. Also, observe that as d is a divisor of $n-1$, it holds that $y_{n-1} = 0$. Hence, it suffices to analyze the set $C(T) \setminus \{n-1\}$. Recall that $[u, v]_r$ denotes the interval $\{u, \ldots, v\}$ modulo r. Also, notice that $C(T_v) \setminus \{n-1\} = [v \oplus c]_{n-1}$, for all $V \setminus \{0, 1\}$ and $C(T_0) \setminus \{n-1\} = [0 \oplus (c+1)]_{n-1}$. The next lemma will be useful to compute $y(C(T))$ if T is an efficient strategy.

Lemma 3.2. *For any $v \in V$ and integer $\ell \leq n-1$ it holds that*

$$y([v \oplus \ell]_{n-1}) = \frac{\ell - \lfloor \frac{\ell}{d} \rfloor}{w(d-1)} \quad \textit{or} \quad y([v \oplus \ell]_{n-1}) = \frac{\ell - \lceil \frac{\ell}{d} \rceil}{w(d-1)}.$$

Proof. TOPROVE 1 ☐

The next technical lemma will be useful to show that $y(C(T))$ is maximized when T is an efficient strategy. In essence, it says that by gluing two intervals into one (and increasing by one the length of the interval, as suggested by Proposition 3.1), we can only increase the amount of captured probability mass.

Lemma 3.3. *Consider two disjoint intervals $[u \oplus \ell]_{n-1}$ and $[v \oplus s]_{n-1}$. Then*

$$y([u \oplus \ell]_{n-1}) + y([v \oplus s]_{n-1}) \leq y([u \oplus (\ell + s + 1)]_{n-1}).$$

Proof. TOPROVE 2 ☐

Now, we have enough tools to show Theorem 3.1.

Proof. TOPROVE 3 ☐

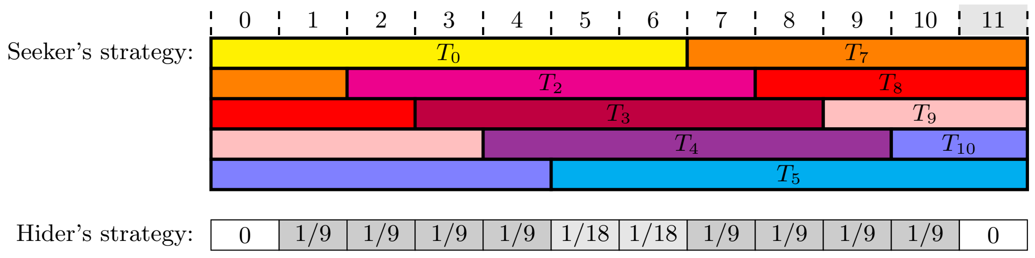


Figure 4: Example of the game on a line of length $n = 12$ with $k = 3$ queries. The searcher chooses one out of 9 strategies. Each strategy is depicted with a distinct color marking the covered cells. The seeker covers uniformly all cells. However the hider chooses a non-uniform distribution where segments have length 1 or 2. The value of the game is 5/9.

The coprime case. We now focus on instances of the game where $\gcd(c, n-1) = 1$. By Lemma 3.1, we have that the objective value of the solution x constructed by Algorithm 1 is $\frac{h}{w}$, where $h, w \geq 1$ are the natural numbers such that $h(n-1) = wc + 1$, with w minimal. As in the previous case, we define explicitly a solution y and show it is optimal. The structure of the solution is similar to the previous case but more intricate and its analysis more technical.

Ideally, we would like to assign a mass of $\frac{h}{wc}$ to each coordinate y_v. In this way, each efficient strategy $T \in \mathcal{T}_k$ of length c would capture a probability of $\frac{h}{w}$. However, this simple idea must be refined as there are issues with the boundary, where efficient strategies have length $c+1$. To handle this situation we define a function $g : V \to \mathbb{Q}$, the *ideal mass* function, that we will use as a benchmark for the cumulative distribution of the solution for the hider y. It is defined as follows:

$$g(v) = v \cdot \frac{h}{wc}.$$

The general idea is that, for any vertex $v \in [1, n-2]$, the cumulative mass of the solution y up to v remains bounded by

$$g(v) \leq y([1, v]) < g(v+1). \tag{5}$$

Such a distribution will give us an optimal solution for [D], as we will show later. To satisfy Equation (5), we partition nodes in $[1, n-2]$ into w segments of two possible lengths: $r := \lfloor \frac{c}{h} \rfloor$ or $r+1$, where length refers to the amount of nodes contained in a segment. Note that $c > h$, so r is at least 1. In each segment, a total probability mass of $\frac{1}{w}$ is distributed uniformly among its nodes. Nodes 0 and $n-1$ are assigned 0 probability mass.

The length and order of the segments is chosen in the following way: start at node $v = 1$, and choose the largest $r^* \in \{r, r+1\}$ such that

$$g(v + r^* - 1) \leq y([1, v-1]) + \frac{1}{w}, \tag{6}$$

then, r^* is the length of the segment starting at v. Set $y_i = \frac{1}{r^* w}$ for $i \in \{v, \ldots, v + r^* - 1\}$. Update $v = v + r^*$ and repeat the same selection rule until $v > n-2$. We call Equation (6) the *segment rule*. Observe that $\frac{1}{r+1} < \frac{h}{c} < \frac{1}{r}$, so the cumulative mass grows faster than g in short segments and slower than g in long segments. Hence, selecting a long segment decreases the difference $y([1, v + r^*]) - g(v + r^*)$ and a short segment increases it. See Figure 5 for an example. We start by showing that the construction is correct, that is, that for either r or $r+1$ Equation 6 is satisfied.

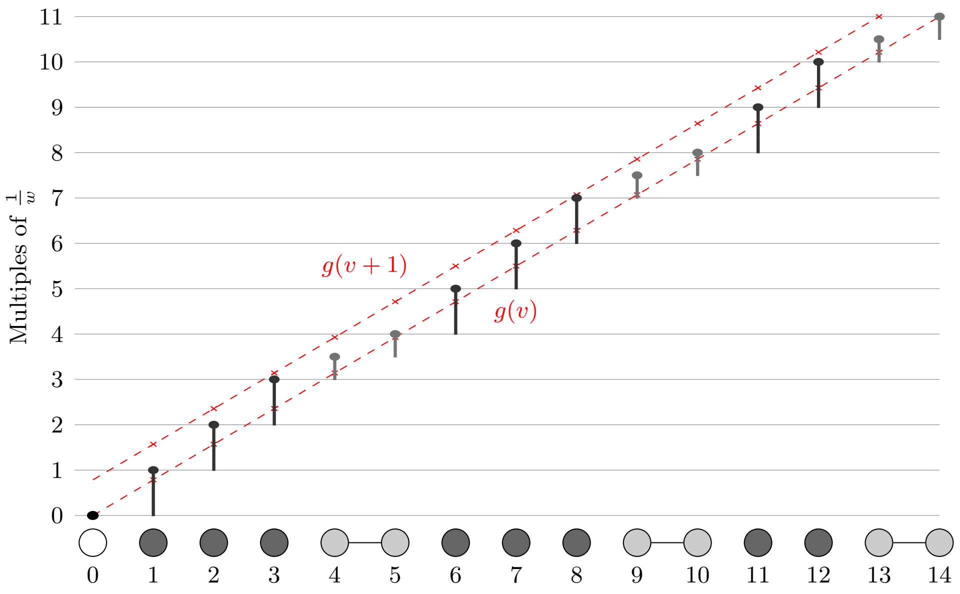


Figure 5: Cumulative distribution of the hider's solution for $n = 38$ and $k = 4$ ($c = 14$) in the first 14 vertices. In this instance, $h = 11$ and $w = 29$, which means short segments (resp. long) have length 1 (resp. 2). In this figure, connected vertices are in the same segment. The probability assigned to each vertex is illustrated by its shade. Notice that $y([1, v])$ always remains within $g(v)$ and $g(v + 1)$.

Lemma 3.4. *Let v be the first node of a segment. Then $g(v - 1) \leq y([1, v - 1])$ and Equation (6) holds for $r^* = r$.*

Proof. TOPROVE 4 □

The next proposition gives several useful properties of our construction of y.

Lemma 3.5. *The constructed solution y satisfies the following properties.*

a) For all $v \leq n - 2$, it holds that $g(v) \leq y([1, v]) < g(v + 1)$.

b) Node $v \in [1, n - 2]$ belongs to the s-th segment where $s = \left\lceil \frac{vh}{c} \right\rceil$.

c) $y([1, v]) = g(v)$ if and only if v is a multiple of c. If v is a multiple of c then it is the last node of a segment.

d) $y_v = y_{v+c}$ for all $v \in [1, n - 2 - c]$.

e) The construction of y finishes with w segments. The last node of the last segment is $n - 2$.

Proof. TOPROVE 5 □

In what follows, we introduce two fundamental results that are crucial for our construction. The two of them combined will make it possible to use the gluing argument utilized in the not coprime case.

Lemma 3.6. *The following equalities hold,*

i) $y([v \oplus \ell]) \leq y([1 \oplus \ell])$ *for all* $\ell, v \in [1, c]$, *and*

ii) $y([v \oplus \ell]) \geq y([t \oplus \ell])$, *for all* $\ell, v \in [1, c]$ *where* $t := (n-1) \mod c$.

Moreover, the segment starting at node 1 and the one ending at node $t-1$ *have length* r, *and the segment starting at* t *and the one ending at* c *have length* $r+1$.

Proof. TOPROVE 6 □

Lemma 3.7. *Let* $t = n - 1 \mod c$. *Then for all* $\ell \in [1, c]$, *we have that*

$$y([1 \oplus \ell]) \leq y([t \oplus (\ell + 1)]).$$

Proof. TOPROVE 7 □

With these properties we can bound the probability mass captured by an efficient strategy. The derived properties are analogous to the ones obtained in Lemmas 3.2 and 3.3 for the not coprime case.

Lemma 3.8. *For all* $v \in V \setminus \{1\}$ *it holds that* $y(C(T_v)) \leq h/w$.

Proof. TOPROVE 8 □

Lemma 3.9. *Consider two disjoint intervals* $[u \oplus \ell]_{n-1}$ *and* $[v \oplus s]_{n-1}$ *for* $\ell + s \leq c$. *Then*

$$y([u \oplus \ell]_{n-1}) + y([v \oplus s]_{n-1}) \leq y([u \oplus (\ell + s + 1)]_{n-1}).$$

Proof. TOPROVE 9 □

Now we have all tools needed to conclude our main theorem.

Theorem 3.2. *The value of the game is* $\frac{h}{w}$. *In particular Algorithm 1 yields an optimal solution for the seeker and (4) and (6) yield an optimal solution for the hider for the coprime and not coprime case, respectively.*

Proof. TOPROVE 10 □

4 A Dynamic Programming Algorithm for the General Case

In this section we consider problems [P] and [D] for the case when G is a general tree and the profit p is an arbitrary non-decreasing function.

A natural approach for computing the optimal profit is using the Ellipsoid method on the dual. For this, we need an algorithm to separate the first set of inequalities of [D], i.e., we need to solve $\max_{T \in \mathcal{T}_k} \sum_{v \in V} y_v \cdot p(h_T(v))$ for a given $y \in \Delta_V$. In other words, we need to solve the *best-response* problem for the seeker. This problem turns out to be NP-hard.

Theorem 4.1. *Computing the best-response for the seeker is NP-hard, even if* G *has constant diameter or constant degree, and if* $p(t) = n - t$ *for all* $t \in \{0, \ldots, n\}$.

Proof. TOPROVE 11 □

In the remaining of this section we show that the best-response problem can be solved in time $O(n^2 2^{2k})$ for an arbitrary tree G. With the Ellipsoid algorithm [18], this implies an algorithm with time complexity $\text{poly}(n)2^{O(k)}$, which is fixed-parameter tractable on k. In order to use the Ellipsoid method, we are interested in separating the first set of inequalities of linear program [D]. This poses the following problem: given a tree $G = (V, E)$, a mixed strategy of the hider $y \in \Delta_V$ and a non-increasing profit function $p : \{1, \dots, k+1\} \to \mathbb{N}_0$ with $p(k+1) = 0$, what is the search strategy in $\mathcal{T}_k$ that maximizes the expected profit of the seeker? In this section, we introduce a dynamic program to solve this question, based on a characterization of search strategies using edge labelings.

4.1 Edge Labelings to describe Search Strategies

Edge labelings are functions that map edges of a graph to numbers. In what follows, we specify the conditions an edge labeling must obey in order to properly encode a search strategy in $\mathcal{T}_k$. Such a labeling will be called *valid*. Subsequently, we formulate the objective function of the best-response problem in terms of valid labelings. Finally, we provide a characterization of valid labelings that will be used by the dynamic program defined in the next subsection.

The relationship between edge labelings and (unrestricted) search strategies was established independently in [14, 24]. We give a modified definition that captures the limited height of the search strategies in $\mathcal{T}_k$.

Definition 4.1 (Valid Labeling)**.** *A function $f : E \to \{0, \dots, k\}$ is a* valid labeling *for tree $G = (V, E)$ if for each pair of distinct edges $e_1, e_2 \in E$ such that $f(e_1) = f(e_2) > 0$, there is an edge e_3 on the simple path from e_1 to e_2 for which $f(e_3) > f(e_1)$. The set of valid labelings with range $\{0, \dots, k\}$ is denoted by $\mathcal{F}_k$*

We remark that every valid labeling reaches its maximum at a unique edge trivially, except for the degenerate all zero labeling. Intuitively, a valid labeling maps an edge e to the remaining budget of a search strategy right before it queries e (see Figure 1 for an example). Observe that a search strategy finds a vertex the moment all of its incident edges are queried. This suggests the following notation: for a vertex $v \in V$ and a valid labeling $f \in \mathcal{F}_k$, let $h_f(v) = k + 1 - \min\{f(e) : e \in \delta(v)\}$. Consequently, the *expected profit* of a valid labeling is given by

$$\sum_{v \in V} y_v \cdot p\left(h_f(v)\right). \tag{7}$$

Then, we can show that the best-response problem can be solved by maximizing this equation.

Proposition 4.1. *The maximum expected profit of a valid labeling is equal to the objective value of y, that is,*

$$\max_{f \in \mathcal{F}_k} \sum_{v \in V} y_v \cdot p(h_f(v)) = \max_{T \in \mathcal{T}_k} \sum_{v \in V} y_v \cdot p(h_T(v)).$$

We leave the proof of this proposition for the appendix. With this equivalence established, we turn our attention to obtaining a "local" description of valid labelings. Such a description will give us a recipe for constructing valid labelings in an algorithmic fashion. Let $G = (V, E)$ be a rooted out-tree and let $f : E \to \{0, \dots, k\}$ be an arbitrary labeling (not necessarily valid). In this setting, an edge e that points from vertex u to v is denoted $e = (u, v)$. We say that an edge e' is *visible* from edge e if on the directed path from e ending in e', there is no other edge e'' such that $f(e'') > f(e')$. In other words, the edges visible from e are those that are not "screened" by greater values of f. The *visibility sequence* of e, denoted $L(e)$, is the enumeration in ascending order of the labels of edges visible from e. We remark that the first label of $L(e)$ equals $f(e)$. We extend the definition of visibility sequence to vertices. The visibility sequence of a vertex u, $S(u)$, is the union of the visibility sequences of its outgoing edges (see Figure 6). The following result gives us the conditions that we need to impose on visibility sets to obtain a valid labeling.

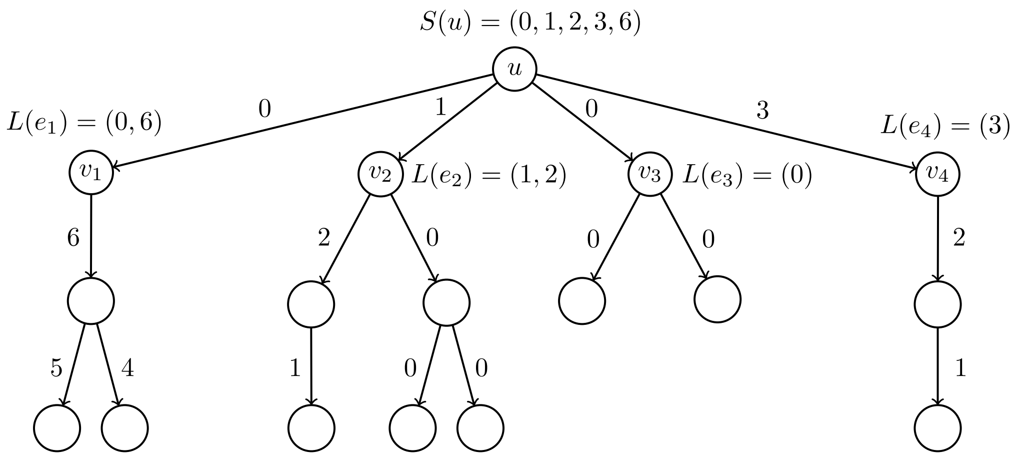


Figure 6: An out-tree rooted at u with an edge labeling. For $i = 1, \ldots, 4$, edge (u, v_i) is denoted by e_i.

Proposition 4.2. *A labeling $f : E \to \{0, \ldots, k\}$ of tree $G = (V, E)$ is valid if and only if for an arbitrary orientation of the tree it holds that*

1. *for every $u \in V$, if $e = (u', u) \in E$ then $f(e) \notin S(u)$ or $f(e) = 0$ (or both), and*
2. *for every $u \in V$, for distinct edges $e, e' \in \delta^+(u)$, the sets $L(e)$ and $L(e')$ are disjoint, except possibly for label 0.*

Proof. TOPROVE 12 □

4.2 Dynamic Program

The idea of the dynamic program is to compute an optimal valid labeling from the bottom up, following an arbitrary orientation of the tree. For every edge and vertex, we need to keep track of their visibility sets and make sure that the conditions of Proposition 4.2 are satisfied.

The dynamic program consists of two tables B and C, with recurrences interleaving one another. For an edge $e = (u, v)$, consider the sub-tree consisting of this edge and the sub-tree rooted at v. By $B[e, L]$ we denote the maximum expected profit of a valid labeling of this sub-tree with visibility sequence L for e. Vertex u is not counted in the profit. Note that the label of e is $\min L$.

Now, for some vertex v with outgoing edges $e_1, \ldots, e_d$ and an integer $1 \leq i \leq d$, consider the subtree consisting of the edges $e_1, \ldots, e_i$ together with the sub-trees attached to the endpoints of these edges. By $C[v, i, S]$ we denote the maximum profit of a valid labeling of this sub-tree with visibility sequence S at vertex v. Again vertex v is not counted in the profit. For convenience we include the degenerate case $i = 0$ corresponding to an empty sub-tree, which includes the case when v is a leaf.

To introduce the recursion for B, let $e = (u, v)$ be an edge and let L be a non-empty visibility sequence and denote $l_1 = \min L$. Let d be the degree of v, which is 0 in case v is a leaf. If $l_1 = 0$,

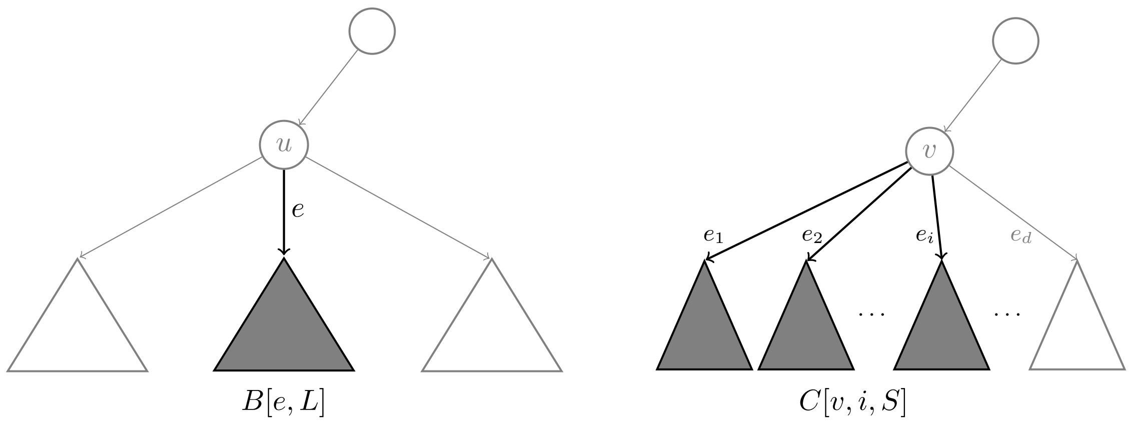


Figure 7: Illustration of table values B and C. The shaded areas correspond to the portion of the tree that is considered by the table.

vertex v is not covered and we have the recursion

$$B[e, L] = \max\{C[v, d, L], C[v, d, L \setminus (0)]\}. \tag{8}$$

If $l_1 > 0$, we have the recursion

$$B[e, L] = \max_{S' \subseteq [l_1]} y(v) \cdot p(k + 1 - \min\{S' \cup (l_1)\}) + C[v, d, (S' \cup L) \setminus (l_1)], \tag{9}$$

where $[l_1]$ denotes the set $\{0, 1, \dots, l_1 - 1\}$. For the second case, S' corresponds to the set of labels screened by l_1, but present in the visibility sequences of outgoing edges from v. The label l_1 is positive, so it must be excluded from visibility sequence of v to satisfy Property 1 of Proposition 4.2. To prevent e to be unlabeled, we define $B[e, \emptyset] = -\infty$. Next, we focus on table C. We define

For the special case $i = 0$ of table C we define

$$C[v, 0, S] = \begin{cases} 0 & \text{if } S = \emptyset \\ -\infty & \text{if } S \neq \emptyset. \end{cases} \tag{10}$$

For an index $1 \leq i \leq d$ we define

$$C[v, i, S] = \max_{L_i \subseteq S} B[e_i, L_i] + \max\{C[v, i-1, S \setminus L_i], C[v, i-1, S \setminus (L_i \setminus (0))]\}. \tag{11}$$

The idea is to decompose the visibility sequence S of vertex v in almost disjoint visibility sequences L_i for each edge e_i, with the potential exception of label 0. This is done in order to preserve the second property of Proposition 4.2. We achieve that by assigning $L_i \subseteq S$ to edge e_i and calling table C again on the remaining edges with updated visibility sequence $S \setminus (L_i \setminus (0))$. The maximum of two choices translates the fact that e_i could be the only edge among $e_1, \dots, e_i$ to be labeled 0, or not the only one. In both cases L_i would contain 0, but in the second case determines whether the recursion in $C[v, i-1, \cdot]$ continues also with a sequence starting with 0 or not.

The maximum value of Equation (7) over all valid labelings can be computed by solving

$$\max_S y(r) \cdot p(k + 1 - \min S) + C[r, d, S], \tag{12}$$

where d is the degree of the fixed root r. Notice, however, that the resulting labeling might not be maximal. This is easily solved by making the algorithm break ties in favor of lexicographically larger visibility sequences. This, together with the fact that p is non-increasing, results in a maximal valid labeling. By following the dynamic program constructed through equations 8, 9 and 10, we obtain the following theorem :

Theorem 4.2. *The best-response problem can be solved in $\mathcal{O}(n^2 2^{2k})$.*

Proof. TOPROVE 13 □

A Missing Proofs of Section 3

A.1 Proof of Proposition 3.1

The following lemmas will be useful to show Proposition 3.1.

Lemma A.1. *Let $G = (V, E)$ be a path graph with n vertices. We say that $p_0, p_1, \dots, p_j \in \mathbb{N}_0$ with $0 = p_0 < p_1 < \dots < p_{j-1} < p_j = n$ define a* partition into j intervals *of V. Then, $\{V(\ell) : \lambda \in L\} = \{[p_i, p_{i+1} - 1] : 0 \le i \le j - 1\}$ for some search strategy $T \in \mathcal{T}_k$ with leaves L if and only if $j \le 2^k$.*

Proof. TOPROVE 14 □

As the proof of the lemma suggests, the partition into intervals of a given search strategy is simply a representation of the edge queries performed by the strategy on the graph in a compact format.

Lemma A.2. *Let $G = (V, E)$ be a path graph with n vertices and let $T \in \mathcal{T}_k$ be some search strategy, with $2^k < n$. For a vertex $v \in V$, denote the unique leaf λ of T such that $V(\lambda) \ni v$ as $\lambda(v)$. Then, $C(T)$ is a maximal covered set if and only if*

a) T has exactly 2^k leaves, and

b) for any vertex $v \in V$ such that $v - 1, v \notin C(T)$, $\lambda(v - 1) = \lambda(v)$.

Proof. TOPROVE 15 □

Now we have all the tools to prove Proposition 3.1.

Proof. TOPROVE 16 □

A.2 Proof of Corollary 3.2

As stated in Lemma 3.1, the values h and w can be computed in time $O(\log n)$ by using the Extended Euclidean algorithm. Take a number t uniformly at random in $\{0, \dots, w - 1\}$ (using $O(\log w) \le O(\log n)$ bits), which represents the $(t + 1)$-th tree inserted to $\mathcal{X}$ in the algorithm. If $t = 0$ this tree corresponds to T_0, otherwise, it corresponds to the efficient strategy T_v with $v = (t \cdot c \mod n - 1) + 1$. For a given v, the queries implied by T_v can be easily determined as in the proof of Proposition 3.1.

B Proof of Proposition 4.1

We first provide an auxiliary lemma.

Lemma B.1. *Let $f \in \mathcal{F}_k$ be a valid labeling. There is a search strategy $T \in \mathcal{T}_k$ such that for every edge $v \in V$, we have $h_T(v) \geq h_f(v)$.*

Proof. TOPROVE 17 □

Notice that the proof of the lemma shows how to recover a search strategy from a valid labeling. Now we continue with the proof of the proposition.

Proof. TOPROVE 18 □

References

[1] Explosive material production (hidden) agile search and intelligence system, HORIZON 2020. https://cordis.europa.eu/project/id/261381. Accessed: 2024-04-21.

[2] W. Ahmed, N. Angel, J. Edson, K. Bibby, A. Bivins, J. W. O'Brien, P. M. Choi, M. Kitajima, S. L. Simpson, J. Li, B. Tscharke, R. Verhagen, W. J.M. Smith, J. Zaugg, L. Dierens, P. Hugenholtz, K. V. Thomas, and J. F. Mueller. First confirmed detection of SARS-CoV-2 in untreated wastewater in Australia: A proof of concept for the wastewater surveillance of COVID-19 in the community. *Science of The Total Environment*, 728:138764, 2020.

[3] J. Baboun, I. S. Beaudry, L. M. Castro, F. Gutierrez, A. Jara, B. Rubio, and J. Verschae. Identifying outbreaks in sewer networks: An adaptive sampling scheme under network's uncertainty. *Proceedings of the National Academy of Sciences (PNAS)*, 121(14):e2316616121, April 2024.

[4] Y. Ben-Asher and E. Farchi. The cost of searching in general trees versus complete binary trees. Technical report, 1997.

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

[6] B. A. Berendsohn, I. Golinsky, H. Kaplan, and L. Kozma. Fast Approximation of Search Trees on Trees with Centroid Trees. *LIPIcs, Volume 261, ICALP 2023*, 261:19:1–19:20, 2023.

[7] B. A. Berendsohn and L. Kozma. Splay trees on trees. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 1875–1900. Society for Industrial and Applied Mathematics, January 2022.

[8] E. Bézout. *Théorie générale des équations algébrique*. De l'imprimerie de Ph. D. Pierres, 1779.

[9] Renato Carmo, Jair Donadelli, Yoshiharu Kohayakawa, and E Laber. Searching in random partially ordered sets. *Theoretical Computer Science*, 321(1):41–57, 2004.

[10] F. Cicalese, T. Jacobs, E. Laber, and M. Molinaro. On the complexity of searching in trees and partially ordered structures. *Theoretical Computer Science*, 412:6879–6896, 2011.

[11] F. Cicalese, T. Jacobs, E. Laber, and M. Molinaro. Improved Approximation Algorithms for the Average-Case Tree Searching Problem. *Algorithmica*, 68:1045–1074, April 2014.

[12] F. Cicalese, T. Jacobs, E. Laber, and C. Valentim. The binary identification problem for weighted trees. *Theoretical Computer Science*, 459:100–112, 2012.

[13] T. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 3rd edition, 2009.

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

[15] C. Di Cristo and A. Leopardi. Pollution Source Identification of Accidental Contamination in Water Distribution Networks. *Journal of Water Resources Planning and Management*, 134(2):197–202, 2008.

[16] E. Domokos, V. Sebestyén, V. Somogyi, A. J. Trájer, R. Gerencsér-Berta, B. Oláné H., E. G. Tóth, F. Jakab, G. Kemenesi, and J. Abonyi. Identification of sampling points for the detection of SARS-CoV-2 in the sewage system. *Sustainable Cities and Society*, 76:103422, 2022.

[17] E. Emamjomeh-Zadeh, D. Kempe, and V. Singhal. Deterministic and probabilistic binary search in graphs. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, STOC '16, pages 519–532, 2016.

[18] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, June 1981.

[19] D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971.

[20] T. W. Lam and F. L. Yue. Optimal Edge Ranking of Trees in Linear Time. *Algorithmica*, 30:12–33, May 2001.

[21] R. C. Larson, O. Berman, and M. Nourinejad. Sampling manholes to home in on SARS-CoV-2 infections. *PLOS ONE*, 15(10):e0240007, 2020.

[22] S. Mozes, K. Onak, and O. Weimann. Finding an optimal tree searching strategy in linear time. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 8 of *SODA '08*, pages 1096–1105, 2008.

[23] M. Nourinejad, O. Berman, and R. C. Larson. Placing sensors in sewer networks: A system to pinpoint new cases of coronavirus. *PLOS ONE*, 16:e0248893, 2021.

[24] K. Onak and P. Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, pages 379–388. IEEE, 2006.

[25] J. O'Keeffe. Wastewater-based epidemiology: current uses and future opportunities as a public health surveillance tool. *Environmental Health Review*, 64:44–52, 2021.

[26] A. A. Schäffer. Optimal node ranking of trees in linear time. *Information Processing Letters*, 33:91–96, 1989.

[27] A. Sulej-Suchomska, A. Klupczynska, P. Derezinski, J. Matysiak, P. Przybylowski, and Z. Kokot. Urban wastewater analysis as an effective tool for monitoring illegal drugs, including new psychoactive substances, in the Eastern European region. *Scientific Reports*, 10(4885), 2020.

[28] J. Von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100:295–320, 1928.