Randomized Binary and Tree Search Under Pressure

Agustín Caracci
Institute for Mathematical and Computational Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile

Christoph Dürr
Sorbonne University, CNRS, LIP6, Paris, France

José Verschae
Institute for Mathematical and Computational Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile

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 time-constrained applications, we might only have time to perform a sub-logarithmic number of queries. 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's position) probability of finding the target. Such a strategy provides a natural solution when there is no a priori (stochastic) information about 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 zero-sum game and exploiting an underlying number theoretical structure.

For the more general case on trees, querying an edge returns the edge's endpoint closest to the target. Given a bound k on the number of queries, 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 (that is, 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 us 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 in general.

2012 ACM Subject Classification Theory of computation → Sorting and searching; Theory of computation → Algorithmic game theory; Theory of computation → Dynamic programming

Keywords and phrases Binary Search, Search Trees on Trees, Nash Equilibrium

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.41

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2406.06468` [8]

Funding *Agustín Caracci*: Partially supported by Fondecyt-ANID Nr. 1221460.
Christoph Dürr: Work partially conducted while C.D. was affiliated with Universidad de Chile, CMM. Partially supported by Centro de Modelamiento Matemático (CMM) BASAL fund FB210005, ANID-Chile, and the grants ANR-19-CE48-0016, ANR-23-CE48-0010 from the French National Research Agency (ANR).
José Verschae: Partially supported by Fondecyt-ANID Nr. 1221460 and by Centro de Modelamiento Matemático (CMM) BASAL fund FB210005, ANID-Chile.

Acknowledgements We thank several anonymous reviewers. Their insightful comments helped to improve the presentation of this document significantly.

52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025).
Editors: Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis
Article No. 41; pp. 41:1–41:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

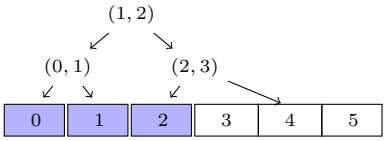
Searching for an object on a line (or, equivalently, an ordered list) with n vertices is one of the most fundamental tasks in computer science. Binary search, one of the most basic discrete algorithms, produces a search strategy that finds the target item with $O(\log(n))$ comparisons, regardless of its position. However, when we encounter tight time constraints, we face the problem of searching under a potentially sub-logarithmic budget of k comparisons. In this paper, we study search problems on paths and, more generally, trees, where the given number of queries does not allow us to search the complete set of nodes.

A prominent recent example arises in the search for infections on a sewer network, a strategy that attracted significant attention during the COVID-19 pandemic [1, 2, 17, 23, 26, 28]. In this context, we can query a node of the sewer network – via a PCR test – and ask whether traces of the virus are found in the sampled water. Since a sewer network can (usually) be modeled as an in-tree [6] – which carries water from different locations to the water-treatment plant, i.e., the root node – a query on a node v tells us if the infection lies in the subtree rooted at v, or its complement. As the water flows towards the root, querying v is equivalent to querying the edge uv, where u is the parent of v. Hence, we can equivalently state the problem for an undirected tree G, where querying edge e indicates the connected component of $G - e$ that contains the target node.

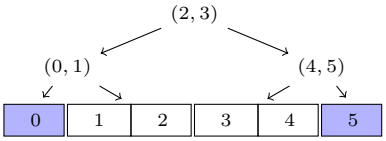
A core but poorly understood question considers the search for the first infection in a particular locality. Unlike other models found in the literature, after the infection is first detected at the water-treatment plant, the system designer faces intense pressure to find the source of the infection within a fixed time constraint: If the infected node is found within a pre-specified number of days, the spread of the infection could be prevented; otherwise, the infection will spread uncontrollably. Observe that this strategy is attractive because the virus can be detected in the wastewater before symptoms appear [20].

Before stating our model in general, consider the example shown in Figure 1, where the network is just a path $G = (V, E)$ with nodes $V = \{0, \dots, 5\}$. A target (infected) node $v^* \in V$ is hidden in the network. We can devise a search strategy that queries edges. Each query answers whether the target is to the left or right of the queried edge. We aim to find (and certify) the target's position within a given number of queries k. In this example, $k = 2$. In our COVID motivation, k is the number of queries we can perform between the infection is detected in the network, and the infected person starts spreading the virus. As one can imagine, the number of queries might not be enough to search the entire path, that is, k is sub-logarithmic. In Figure 1a, we depict a search strategy (i.e., a binary tree) and the positions where the target would be found, which is, for this search strategy, a subpath. If we do not have a priori information on the position of v^*, selecting a deterministic strategy will leave blind spots (i.e., the white boxes). Choosing a randomized strategy allows us to guarantee a non-zero probability of finding the target, regardless of its position. We study the problem of devising a randomized strategy that maximizes this guaranteed probability, i.e., that maximizes the worst-case (with respect to the position of the target) probability of finding the target. Uniformly randomizing the strategy in Figure 1a with the mirrored strategy (that finds the target if $v^* = 3, 4$ or 5) yields the optimal solution in this case, where the target is found with a probability of at least 0.5, regardless of its position. In contrast, the solution in Figure 1b, yields a strategy that is never in the support of an optimal solution.

The described setting can be equivalently posed as a two-player zero-sum game on a path $G = (V, E)$. 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



(a) Search strategy that captures the target if it belongs to the left half of the path. Uniformly randomizing with the mirrored strategy (that captures the target on the right half of the path), we can guarantee that the target is found with probability 0.5, regardless of its position. This is the optimal randomized strategy.



(b) Solution that performs the usual binary search strategy but stops after 2 queries. As nodes 1 and 4 are not captured, and no strategy with budget $k = 2$ can capture both nodes simultaneously, this strategy can never belong to the support of an optimal solution.

Figure 1 Example of an instance of our search problem with a budget of $k = 2$. Nodes of the path are represented by boxes, numbered from 0 to 5. For each search strategy, blue boxes represent positions where the target node is found.

edges to query to find the hidden node v^*. If edge e is queried, the seeker learns whether v^* is to the left or to the right of e. 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 players get a profit of 0. We want to find a mixed (i.e., randomized) strategy for each player that yields a Nash equilibrium. That is, the seeker (respectively, the hider) aims to maximize (resp., minimize) the probability that the target is found within k queries. For the example in Figure 1, a Nash equilibrium is the mentioned optimal strategy for the seeker (randomizing between Figure 1a and the mirror solution) and a solution that picks a random vertex within $\{1, \dots, 4\}$ for the hider. For a given randomized search strategy, let $P(v)$ be the probability of finding the target if $v^* = v$. If the randomized solution is part of a Nash equilibrium, the hider's best response is a probability distribution supported within $\arg\min_{v \in V} P(v)$, which yields a profit of $\min_{v \in V} P(v)$ for the seeker. Therefore, in a Nash equilibrium, the seeker maximizes $\min_{v \in V} P(v)$, which was our original goal.

The advantage of interpreting our problem as a zero-sum game is twofold: *(i)* it allows to interpret a hider's Nash strategy as a dual certificate of optimality of the seeker's strategy (by exploiting von Neummans' minimax theorem or, equivalently, LP-duality; see Section 2 for details), and *(ii)* the Nash equilibrium explicitly captures situations when both the hider and seeker act strategically, in particular, when the target is hidden adversarially. This last point is particularly relevant in other applications. For example, similar search problems have been proposed to detect the presence of illicit drugs [30], pollution [16] and even illicit explosives [13] in water networks.

Inspired by these applications, we extend our model in two directions. First, we consider the case in which the graph $G = (V, E)$ is not only a path, but an arbitrary tree. As before, the seeker queries edges. If $e \in E$ is queried, she obtains as 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 $p(t) = 0$ if $t > k$. If the seeker finds the hider with t queries, she obtains a profit of $p(t)$, and the hider perceives a cost of $p(t)$ (or, equivalently, a profit of $-p(t)$). We obtain the same problem as above if $p(t) = 1$ for $t \leq k$, which we call the *unit-profit* case. As before, the objective is to find a mixed Nash equilibrium efficiently.

Model discussion

Our paper studies a models where the seeker must certify exactly the position of the target at the end of the procedure. Guessing the target at the end – without spending queries to certify it – yields zero profit. This feature is motivated by cases where the seeker is accountable for actions taken after the target is found, which include, e.g., forcing quarantines or other extreme measures. It is interesting to consider extensions, where some profit might be gained depending on the size of the target's range after the k queries. We leave such models for future work.

Related Literature

Previous literature mostly focuses on minimizing the worst-case number of queries needed to find the target. When the target is hidden in a tree, Lam and Yue [22] (see also [15]) provide 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. [3], Onak and Parys [27], and Mozes et al. [25]. 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 approximation factor.

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 [29] provides a linear-time algorithm generating an optimal search tree. 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 dependency is necessary, as there is no $O(m^{(1-\epsilon)\log n})$-time algorithm under the Strong Exponential-Time Hypothesis (SETH) [18]. 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 [21] shows that dynamic programming can solve this problem on the line with running time $O(n^2)$, while Mehlhorn [24] shows that bisecting the probability mass in each iteration yields a constant-factor approximation algorithm. For the more complicated problem on trees in the edge query model, Cicalese et al. [10, 11] shows that finding an optimal search strategy is NP-hard. Moreover, it admits a 1.62-approximation algorithm and even an FPTAS on trees of bounded degree. On the other hand, for node queries, the complexity of the problem is still open. Recently, Berendsohn and Kozma [5] show that the problem admits a PTAS, and Berendsohn et al. [4] analyze 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 on a 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 edge to query takes constant time. The solution picks a random interval to perform binary search on it. The interval is chosen uniformly at random from a set constructed with a greedy algorithm based on modular arithmetics (see Figure 4). 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 5).

For *(ii)*, where $G = (V, E)$ 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, 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* [22, 27, 15], 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. 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 [3, 18]. 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 and discuss the connection with zero-sum games. Our main result is 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 details are deferred to the appendix.

2 Problem Definition

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

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 $e(\nu) \in E$; see Figure 2 for an example. By definition, the root $\rho \in N$ is labeled, unless $|N| = 1$. The

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

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 the 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 $|N| = 1$.

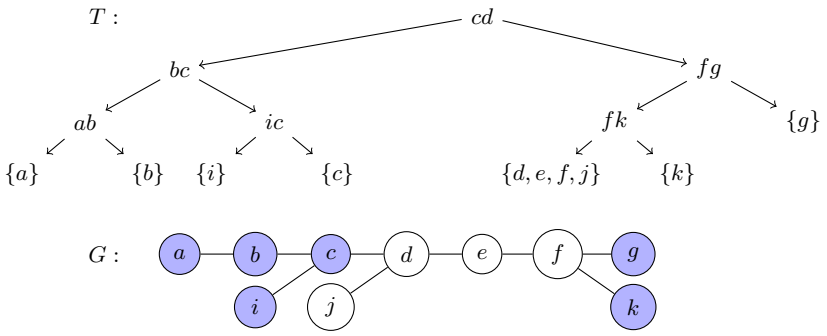


Figure 2 Example of a search tree T over a tree G. Leafs ν of T are labeled with $V(\nu)$. Covered vertices in G are colored blue.

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 of V into connected components. 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 represent with $\mathcal{T}_k$ 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,

$$\Delta_k := \left\{ x : x \geq 0 \text{ and } \sum_{T \in \mathcal{T}_k} x_T = 1 \right\}.$$

For the *unit-profit* case, 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 [31], a fundamental 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 deterministic search strategy, namely 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 arise 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 $C(T)$ represents the set of covered nodes. Much of our technical work is dedicated to show that an optimal randomized solution selects only search strategies where $C(T)$ forms a set of consecutive nodes modulo n. For $u, v \in \mathbb{N}$, we define $[u, v]_r$ as $\{w \mod r : u \leq w \leq v\}$ if $u \leq v$, and $[u, v]_r = \emptyset$ otherwise. For example, $[3, 7]_5$ is $\{3, 4, 0, 1, 2\}$. We refer to $[u, v]_r$ as an *interval modulo* r, or simply an *interval*. To simplify notation, we omit the subscript r when $r = n$, writing $[u, v]$ instead of $[u, v]_n$. Additionally, for an integer $\ell \leq r$, we define $[v \oplus \ell]_r = [v, v + \ell - 1]_r$, which we refer to as *the interval that starts at v of length ℓ* or simply *an interval of length ℓ starting at v*[2]. Observe that this notation describes intervals that "wrap around" the path in a concise manner.

It will be particularly useful to understand the sets that appear as a cover set of a search strategy $T \in \mathcal{T}_k$. For a fixed value of k, for the rest of this section we define $c := 2^k - 2$. Observe that a search strategy that aims to cover a single interval in $[1, n-2]$ (i.e., an interval that does not wrap around) is able to cover c nodes: just apply binary search restricted to a given interval of length c. On the other hand, if the strategy covers a single (wrap-around) interval that contains either 0 or $n-1$ (or both), it will 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 $[n-c-1, n-2]$ and node $n-1$. 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 $c-1$; see Figure 3b. 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 1.** *propositionNonDominated Let $c = 2^k - 2$. Let $C = [u_1 \oplus \ell_1] \cup \cdots \cup [u_s \oplus \ell_s] \subseteq V$ be a set where $0 \leq u_1 < \cdots < u_s \leq n-1$ (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)** $\sum_{t=1}^{s} \ell_t = \begin{cases} c + 1 - s & \text{if } \{0, n-1\} \cap C = \emptyset, \\ c + 2 - s & \text{otherwise.} \end{cases}$

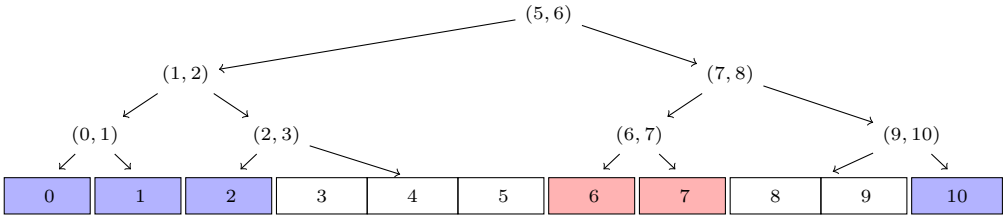
The first two conditions in the proposition encompass the fact that the vertices not covered by a search strategy always have at least one neighbor that is also not covered. The third condition states a trade-off between s, the number of intervals in C, and the cardinality of C. Finally, it also says that if C contains 0 or $n-1$ (or both), the strategy gains an extra covered node. See Figure 3 for an illustration.

Of particular interest are search strategies that cover 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 1.

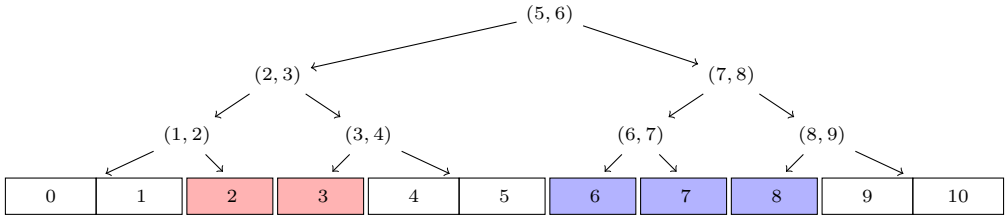
▶ **Corollary 2.** *Let $c = 2^k - 2$. For every $v \in V \setminus \{1\}$ there exists an efficient search strategy $T_v \in \mathcal{T}_k$ that covers*

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

[2] The length of an interval is not to be confused with the length of a path. The former is given by the number of vertices and the latter by the number of edges in the path.



(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 3 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. Figure 3a shows the first case of condition (iii) of Proposition 1 and Figure 3b shows the second case.

Due to the first condition of Proposition 1, no efficient strategy covers an interval starting at 1. For $v \in V \setminus \{1\}$, let T_v be the efficient search strategy that covers $C(T_v)$, 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 support of x. Given $\mathcal{X} := \text{supp}(x)$, 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 5. 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 are covered only once. This imbalance suggests we should add more strategies, the most natural being T_2, which produces an imbalance for nodes in $\{8, 9, 10, 11\}$. By continually adding strategies to $\mathcal{X}$ in a greedy manner, we obtain the solution in Figure 5, yielding an objective value of $\min_v \sum_{T: v \in C(T)} x_T = 5/9$. This example suggests Algorithm 1, which can be interpreted as a greedy rectangle packing algorithm, see Figure 4 and Figure 5.

Observe that in Line 4 of Algorithm 1 we take $v + c$ modulo $n - 1$ instead of modulo n, which might be counterintuitive. The reason for this choice is that if we remove node $n - 1$, then for every $v \notin \{0, 1\}$ the cardinality (length) of each set $C(T_v) \setminus \{n - 1\}$ is c. This avoids the case distinction of Corollary 2.

Given a set $\mathcal{X}$, the objective value of x is $\min_v \sum_{T: v \in C(T)} x_T$. If in the while loop we reach the case $v = 0$, 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

Algorithm 1 Greedy algorithm for the seeker's strategy.

```
1 Set v = c + 1 and 𝒳 = {T_0};
2 while v ∉ {0, 1} do
3 |   𝒳 = 𝒳 ∪ {T_v};
4 |   v = (v + c) mod n − 1;
5 end
6 Set x_T = 1/|𝒳| if T ∈ 𝒳 and x_T = 0 otherwise.
```

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, let $w = |\mathcal{X}|$ denote the cardinality of $\mathcal{X}$. With these parameters, 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 to $\gcd(c, n-1)$, the greatest common divisor of c and $n-1$, and Bezout's identity. The proof of the lemma is based on the fact that the algorithm indeed finishes and that the values of h and w satisfies the Bezout's identity $h(n-1) - wc = \gcd(n-1, c) = 1$ if $n-1$ and c are coprime. The running times in the next lemma are considered in the RAM model.

▶ **Lemma 3.** *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 $0 < w \leq n-2$ is minimal. In particular, we can compute w, h, and hence the objective value of x, in $O(\log n)$ time. Moreover, if c and $n-1$ are not coprime the objective value simplifies to $\frac{h}{w} = \frac{c}{n-1}$, where $h = \frac{c}{gcd(c,n-1)}$ and $w = \frac{n-1}{gcd(c,n-1)}$.

Proof. We have already argued about the objective value of x.

We now give a proof for Equation (3) and argue that the algorithm terminates. For a given iteration, consider the state of the algorithm right after running line 4, in particular we have a set $\mathcal{X}$ together with an updated value of v. Let w be the cardinality of $\mathcal{X}$ and h its height, defined as $h = \min_{v \in V} |\{T \in \mathcal{X} : v \in C(T)\}|$.

We use a volume argument. Define the volume of $\mathcal{X}$ as $\text{vol}(\mathcal{X}) = \sum_{T \in \mathcal{X}} |C(T) \setminus \{n-1\}|$. Observe that the length of all intervals $C(T) \setminus \{n-1\}$ for $T \in \mathcal{X} \setminus \{T_0\}$ is c. Thus, $\text{vol}(\mathcal{X}) = 1 + wc$. On the other hand, $\text{vol}(\mathcal{X}) = (n-1)h + v$. Notice that v can also be seen as $\text{vol}(\mathcal{X}) \mod n-1$, and that the algorithm terminates the first time that $v \in \{0, 1\}$. Hence, if $\mathcal{X}$ is the final set, then w is the smallest number such that, for some $v \in \{0, 1\}$, it holds that

$$h(n-1) - wc = 1 - v.$$

We now show that the algorithm finishes for $w \leq n-2$ and that $v = 0$ if and only if $\gcd(c, n-1) = 1$ at termination.

Recall that the basic properties of modular arithmetic and Bezout's identity [7] imply that all numbers of the form $s(n-1)+tc$ for $s,t \in \mathbb{Z}$ are multiples of $\gcd(c, n-1)$. Moreover, there exists a pair $(s,t) \in \mathbb{Z}^2$ that satisfies equation $s(n-1)+tc = \gcd(c, n-1)$, and any pair $(s',t') \in \mathbb{Z}^2$ that satisfies this equation is of the form $(s', t') = (s - r\frac{c}{\gcd(c,n-1)}, t + r\frac{n-1}{\gcd(c,n-1)})$ for some $r \in \mathbb{Z}$. Therefore, the pair (s,t) that satisfies this equation with the largest value $t < 0$ satisfies $|t| \leq \frac{n-1}{\gcd(c,n-1)}$, where equality holds if and only if c divides $n-1$.

Assume that $\gcd(c, n-1) = 1$. With the previous discussion, the smallest w such that $h(n-1) - wc = 1$, satisfies that $w < n-1$. This in particular implies that the algorithm terminates after at most $n-1$ iterations. Moreover, the pair (h', w') with the smallest $w' > 0$ such that $h'(n-1) - w'c = 0$ implies that $w' = n-1$ (and $h' = c$), as otherwise c and $n-1$ would share a common divisor. Hence, the smallest value of w such that $h(n-1) - wc \in \{0, 1\}$ is attained when $h(n-1) - wc = 1$, and thus Equation (3) holds.

If $\gcd(c, n-1) > 1$, then there are no values of $h, w \in \mathbb{Z}$ such that $h(n-1) - wc = 1$, and hence w is the smallest number such that $h(n-1) - wc = 0$ by construction. This implies Equation (3) and that the algorithm terminates after $w = \frac{n-1}{\gcd(c,n-1)}$ many iterations.

To obtain a fast algorithm to compute h and w, use the Extended Euclidean Algorithm to compute values of $s, t \in \mathbb{N}$ such that $s(n-1) - tc = \gcd(c, n-1)$ and $t > 0$ is minimal. This takes $O(\log n)$ time (in the RAM model) [14, Chapter 31]. If $\gcd(c, n-1) = 1$ then we are done as $h = s$ and $w = t$. Otherwise, $h = \frac{c}{\gcd(c,n-1)}$ and $w = \frac{n-1}{\gcd(c,n-1)}$. ◀

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 4.** *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 4 or vertices 5 and 6 in Figure 5. 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 contains at most the same probability mass as an interval obtained by gluing them together according to Proposition 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.

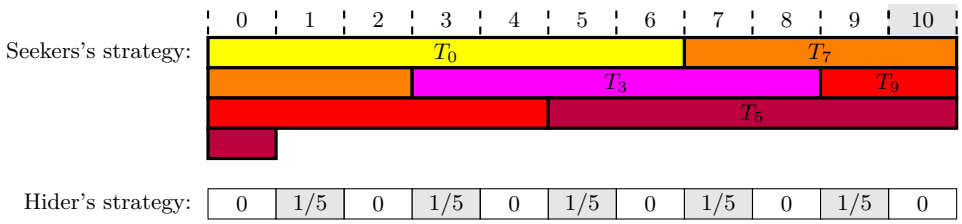


Figure 4 Example of the game on a line of length $n = 11$ with $k = 3$ queries, where $\gcd(c, n-1) = 2$. The searcher choose uniformly one out of 5 strategies. Each strategy is depicted with a distinct color marking the covered cells. Here all cells are covered with probability 3/5, except the first 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.

We start with the non-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 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 [D] 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, the probability mass of $1/w$ is uniformly distributed among its first $d-1$ vertices (see Figure 4). 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 show that y is indeed a probability distribution with objective value $\frac{c}{n-1}$.

▶ **Theorem 5.** *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 $S \subseteq V$, let $y(S) = \sum_{v \in S} y_v$. To prove the theorem, 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 6.** *For any $v \in V$ and positive integer $\ell \leq n-1$ it holds that*

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

Proof. The definition of y implies that for any v and ℓ it holds that $y([v \oplus \ell]_{n-1}) = (\ell - m)/(w(d-1))$, where m is the number of elements in $[v \oplus \ell]_{n-1}$ that are multiples of d. It is easy to see that $m = \lfloor \ell/d \rfloor$ or $m = \lceil \ell/d \rceil$. The lemma follows. ◀

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 1), we can only increase the amount of captured probability mass. With this lemma we will have enough tools to show Theorem 5.

▶ **Lemma 7.** *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. Observe that

$$y([u \oplus (\ell + s + 1)]_{n-1}) = y([u \oplus \ell]_{n-1}) + y([(u + \ell) \oplus (s + 1)]_{n-1}),$$

and hence it suffices to show that $y([(u + \ell) \oplus (s + 1)]_{n-1}) \geq y([v \oplus s]_{n-1})$. Indeed,

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

where the first inequality follows by Lemma 6. ◀

Proof (Theorem 5). First we check that y is indeed a probability distribution. To do this, note that there are $(n-1)/d = w$ multiples of d in $V \setminus \{n-1\}$. Hence, we have that

$$y(V) = \frac{n - 1 - w}{w(d-1)} = \frac{d(n-1) - (n-1)}{wd(d-1)} = \frac{(n-1)}{wd} = 1,$$

and hence y defines a probability distribution over V.

In order to show that y has an objective value of $\frac{c}{n-1}$, by weak duality it suffices to show that $y(C(T)) \leq \frac{c}{n-1}$ for all $T \in \mathcal{T}_k$. First of all, observe that for any $v \in V$ we have that

$$y([v \oplus c]_{n-1}) \leq \frac{c - \lfloor \frac{c}{d} \rfloor}{w(d-1)} = \frac{c - \frac{c}{d}}{w(d-1)} = \frac{c(d-1)}{(n-1)(d-1)} = \frac{c}{n-1},$$

where the second equality follows as d divides c. Hence, if $v \in V \setminus \{0, 1\}$ we have that $y(C(T_v)) = y(C(T_v) \setminus \{n-1\}) = y([v \oplus c]_{n-1}) \leq \frac{c}{n-1}$. Finally, if $v = 0$, as $y_0 = 0$, then $y(T_0) = y([0 \oplus (c+1)]_{n-1}) = y([1 \oplus c]_{n-1}) \leq \frac{c}{n-1}$. We conclude that $y(C(T)) \leq \frac{c}{n-1}$ for any efficient search strategy T.

Let $T \in \mathcal{T}_k$ be any tree with $C(T)$ maximal. We must show that $y(C(T)) \leq c/(n-1)$. By Proposition 1 we know that $C(T) = [u_1 \oplus \ell_1] \cup \cdots \cup [u_s \oplus \ell_s]$, where $0 \leq u_1 < \ldots < u_s \leq n-1$ and $u_1 \neq 1$. Assume that $s \geq 2$, as otherwise T is an efficient strategy and we are done. Observe that $C(T) \setminus \{n-1\} = [u_1 \oplus \ell'_1]_{n-1} \cup \cdots \cup [u_s \oplus \ell'_s]_{n-1}$, where $\ell'_t = \ell_t$ for all $t \leq s-1$ and $\ell'_s = \ell_s$ if $n-1 \notin [u_s \oplus \ell_s]$ and $\ell'_s = \ell_s - 1$ otherwise. Regardless, reinterpreting condition (iii) in Proposition 1, and recalling that intervals in $C(T)$ are maximal, it holds that if $u_1 \neq 0$ then $\sum_{t=1}^{s} \ell'_t = c + 1 - s$ and $\sum_{t=1}^{s} \ell'_t = c + 2 - s$ if $u_1 = 0$.

Also by Proposition 1, there exists a search strategy $T' \in \mathcal{T}$ such that

$$C(T') \setminus \{n-1\} = [u_1 \oplus (\ell'_1 + \ell'_2 - 1)]_{n-1} \cup [u_3 \oplus \ell'_3]_{n-1} \cup \cdots \cup [u_s \oplus \ell'_s]_{n-1},$$

that is, we merged the first and second interval, gaining one unit in the total length. Observe that regardless of weather $u_1 = 0$ or not, the new intervals satisfy the conditions of Proposition 1, guaranteeing the existence of T'. Moreover, by Lemma 7, we have that

$$y(C(T)) = y(C(T) \setminus \{n-1\}) \leq y(C(T') \setminus \{n-1\}) = y(C(T')).$$

Iterating this argument we obtain that $y(C(T)) \leq y(C(T_{v_1}))$. Weak duality implies the theorem. ◀

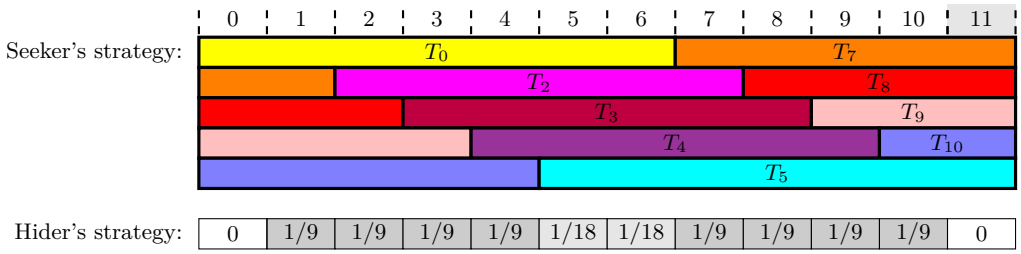


Figure 5 Example of the game on a line of length $n = 12$ with $k = 3$ queries. The searcher chooses one out of 9 strategies. Each strategy is depicted with a distinct color marking the covered cells. The seeker covers uniformly all cells. However the hider chooses a non-uniform distribution where segments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

The coprime case

It remains to show optimality of the solution x in the case $\gcd(c, n-1) = 1$. We define a solution y which assigns probability 0 to the border cells 0 and $n-1$. The remaining cells are partitioned into w segments, and a probability mass $1/w$ is uniformly distributed among the cells of each segment, see Figure 5.

The idea for the definition of the segments is as follows. 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, $g(v) := v \cdot \frac{h}{wc}$. 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 $g(v) \leq y([1, v]) < g(v+1)$. To satisfy this property, 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 number of nodes in a segment).

To define the segments start at node $v = 1$, and choose the largest $r^* \in \{r, r+1\}$ such that

$$g(v + r^* - 1) \leq y([1, v-1]) + \frac{1}{w}, \tag{5}$$

then, r^* is the length of the segment starting at v. Set $y_i = \frac{1}{r^* w}$ for $i \in [v \oplus r^*]$. Update $v = v + r^*$ and repeat the same selection rule until $v > n-2$. We call Equation (5) the *segment rule*.

The constructed y can be used to establish the optimality of x. To do so, we need to show that the seeker cannot capture more than h/w probability mass from y with any strategy in $\mathcal{T}_k$. The proof is given in the full version [8]. It follows a similar structure as for the non-coprime case. However, as the segments are not of uniform length, the analysis is significantly more technical. Combining both cases, coprime and non-coprime, allows to conclude our main theorem.

▶ **Theorem 8.** *The value of the game is* $\frac{h}{w}$*. In particular, Algorithm 1 yields an optimal solution for the seeker, while* (4) *and the segment rule* (5) *yield an optimal solution for the hider for the non-coprime and coprime cases, respectively.*

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 [19] 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: 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? This problem turns out to be NP-hard; see the full version [8] for the proof.

▶ **Theorem 9.** *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 $t \in \{0, \dots, n\}$.*

Next, we show that, for an arbitrary tree G, the best-response problem can be solved in time $O(n^2 2^{2k})$ via a dynamic program based on characterizing search strategies using edge labelings. With the Ellipsoid method, this implies an optimal algorithm with time complexity $\text{poly}(n)2^{O(k)}$.

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 [15, 27]. We give a modified definition that captures the limited height of the search strategies in $\mathcal{T}_k$, by allowing 0 labels (representing that the edge is never queried).

▶ **Definition 10** (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 2 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 $\sum_{v \in V} y_v \cdot p\,(h_f(v))$, and hence we can show that the best-response problem can be solved by maximizing this expression.

▶ **Proposition 11.** *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 us orient G by rooting it arbitrarily and directing the edges away from the root. Let $f : E \to \{0, \ldots, k\}$ 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. Its proof can be found in the full version [8].

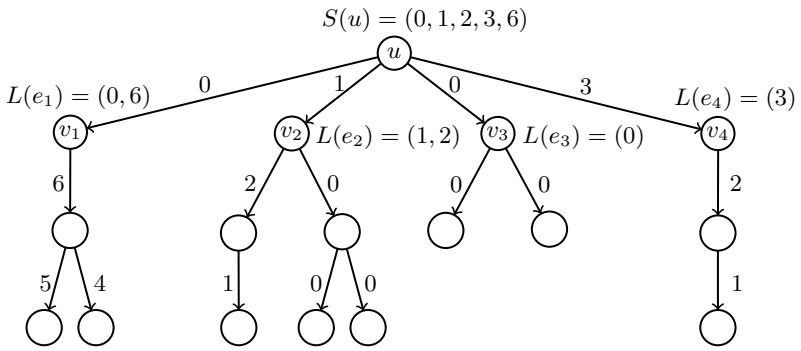


Figure 6 An out-tree rooted at u with an edge labeling. For $i = 1, \ldots, 4$, we use $e_i := (u, v_i)$.

▶ **Proposition 12.** *A labeling $f : E \to \{0, \ldots, k\}$ of tree $G = (V, E)$ is valid if and only if, for every possible rooting of the tree with edges directed away from the root, the following hold: (i) for every $u \in V$, if $e = (u', u) \in E$ then $f(e) \notin S(u)$ or $f(e) = 0$ (or both); and (ii) 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.*

4.2 Dynamic Program

The idea of the dynamic program is to compute an optimal valid labeling with a bottom-up traversal, following an arbitrary rooting of the tree, with edges directed away from the root. A similar technique was used by Cicalese et al. [12, Proposition 1]. For every edge

and vertex, we need to keep track of their visibility sets and make sure that the conditions of Proposition 12 are satisfied. This is done using 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. Let $B[e, L]$ denote the maximum expected profit of a valid labeling of this sub-tree with visibility sequence L for e.

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. Let $C[v, i, S]$ denote the maximum profit of a valid labeling of this sub-tree with visibility sequence S at vertex v. The Bellman equations for tables B, C are defined in a natural way, with special care for border cases. The overall running time is $\mathcal{O}(n^2 2^{2k})$. Details are given in the full version [8].

▶ **Theorem 13.** *The best-response problem can be solved with a running time of* $\mathcal{O}(n^2 2^{2k})$.

Together with the Ellipsoid method, we can directly conclude the following.

▶ **Corollary 14.** *An optimal solution for the problem of searching on a tree with a budget* k *and an arbitrary non-decreasing profit function* p *can be computed in polynomial time if* $k \in O(\log(n))$. *In particular, if the input tree* G *has a constant maximum degree,* $d = O(1)$, *then the optimal solution can be found in polynomial time regardless of the value of* k.

5 Conclusions and Open Problems

In this article, we have introduced a model for search problems under pressure, with a strict budget constraint on the number of queries needed to find the target.

Several open questions remain. Arguably, the most fundamental corresponds to search on trees to minimize the worst-case number of queries needed to find the target. As shown in Corollary 14, if $k = O(\log(n))$, then the problem is polynomially solvable. Although for arbitrary profit functions, the problem is NP-hard (Theorem 9), for the unit-profit case the status of the problem is left open for larger values of k and if the maximum degree is super constant. Similarly, we leave the problem of efficiently approximating the optimal solution for general profit functions.

In our model, a profit is only obtained if the target is localized with complete certainty. An exciting future direction is to consider generalized models where a partial profit is obtained if the target is localized to a small enough subset of vertices.

References

1 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.

2 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, 2024.

3 Y. Ben-Asher, E. Farchi, and I. Newman. Optimal search in trees. *SIAM Journal on Computing*, 28(6):2090–2102, 1999. doi:10.1137/S009753979731858X.

4 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. doi:10.4230/LIPICS.ICALP.2023.19.

5 B. A. Berendsohn and L. Kozma. Splay trees on trees. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '22, pages 1875–1900. Society for Industrial and Applied Mathematics, 2022.

6 A. Berko, V. Zhuk, and I. Sereda. Modeling of sewer networks by means of directed graphs. *Environmental Problems*, 2:97–100, 2017.

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

8 A. Caracci, C. Dürr, and J. Verschae. Randomized binary and tree search under pressure, 2024. doi:10.48550/arXiv.2406.06468.

9 R. Carmo, J. Donadelli, Y. Kohayakawa, and E. Laber. Searching in random partially ordered sets. *Theoretical Computer Science*, 321(1):41–57, 2004. doi:10.1016/J.TCS.2003.06.001.

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. doi:10.1016/J.TCS.2011.08.042.

11 F. Cicalese, T. Jacobs, E. Laber, and M. Molinaro. Improved Approximation Algorithms for the Average-Case Tree Searching Problem. *Algorithmica*, 68:1045–1074, 2014. doi:10.1007/S00453-012-9715-6.

12 F. Cicalese, T. Jacobs, E. Laber, and C. Valentim. The binary identification problem for weighted trees. *Theoretical Computer Science*, 459:100–112, 2012. doi:10.1016/J.TCS.2012.06.023.

13 CORDIS. Explosive material production (hidden) agile search and intelligence system, HORIZON 2020. https://cordis.europa.eu/project/id/261381. Accessed: 2025-04-21.

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

15 D. Dereniowski. Edge ranking and searching in partial orders. *Discrete Applied Mathematics*, 156(13):2493–2500, 2008. doi:10.1016/J.DAM.2008.03.007.

16 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.

17 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.

18 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.

19 M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981. doi:10.1007/BF02579273.

20 D. L. Jones, M. Baluja, D. W. Graham, A. Corbishley, J. E. McDonald, S. K. Malham, L. S. Hillary, T. R. Connor, W. H. Gaze, I. B. Moura, M. H. Wilcox, and K. Farkas. Shedding of SARS-CoV-2 in feces and urine and its potential role in person-to-person transmission and the environment-based spread of COVID-19. *Science of The Total Environment*, 749:141364, 2020.

21 D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971. doi:10.1007/BF00264289.

22 T. W. Lam and F. L. Yue. Optimal Edge Ranking of Trees in Linear Time. *Algorithmica*, 30:12–33, 2001. doi:10.1007/S004530010076.

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

24 K. Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5(4):287–295, 1975. doi:10.1007/BF00264563.

25 S. Mozes, K. Onak, and O. Weimann. Finding an optimal tree searching strategy in linear time. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 1096–1105, 2008.

26 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.

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

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

29 A. A. Schäffer. Optimal node ranking of trees in linear time. *Information Processing Letters*, 33:91–96, 1989. doi:10.1016/0020-0190(89)90161-0.

30 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.

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