Finding Hidden Independent Sets in Interval Graphs

Therese Biedl[1], Broňa Brejová[1], Erik D. Demaine[2], Angèle M. Hamel[3], Alejandro López-Ortiz[1], Tomáš Vinař[1]

[1] School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada, {biedl,bbrejova,alopez-ortiz,tvinar}@uwaterloo.ca

[2] MIT Laboratory for Computer Science, 200 Technology Square, Cambridge, MA 02139, USA, edemaine@mit.edu

[3] Department of Physics and Computing, Wilfrid Laurier University, Waterloo, ON, N2L 3C5, Canada, ahamel@wlu.ca

Abstract

We design efficient competitive algorithms for discovering hidden information using few queries. Specifically, consider a game in a given set of intervals (and their implied interval graph G) in which our goal is to discover an (unknown) independent set X by making the fewest queries of the form "Is point p covered by an interval in X?" Our interest in this problem stems from two applications: experimental gene discovery with PCR technology and the game of Battleship (in a 1-dimensional setting). We provide adaptive algorithms for both the verification scenario (given an independent set, is it X?) and the discovery scenario (find X without any information). Under some assumptions, these algorithms use an asymptotically optimal number of queries in every instance.

Keywords: games, interval graphs, independent set, adaptive algorithms, gene finding, Battleship.

1 Introduction

An *interval graph* is an intersection graph of intervals on the real line, i.e. vertices are represented by intervals and there is an edge between two vertices if and only if their corresponding intervals intersect. Interval graphs have a number of applications, for example in genetics, archeology and developmental psychology (see e.g. [22]). Their geometric structure makes it easy to solve various optimization problems, among them finding the maximum independent set or a clique cover (see e.g. [12]). An *independent set* in a graph G is a set of vertices such that no two vertices share an edge.

In this paper we study how to determine, given a set of intervals (with their implied interval graph G), an unknown (hidden) independent set X in G. One could think of this independent set as having been chosen by a player in an interactive game. The other player then tries to determine X using queries of the following type: "Is a point p on the real line covered by an interval in X?" The first player always answers the query truthfully. Our goal is to use the smallest possible number of queries to determine set X. Our problem is motivated by two applications: recovering gene structure with PCR techniques and the game of Battleship. We explain the connections to our problem after stating it precisely.

While there is a wide literature regarding games in graphs (e.g., game coloring [3], the Ramsey graph game [9], and node search [14]), our problem appears to be new in this area.

Several games involving finding a hidden object using queries have also been studied in the bioinformatics literature. Xu et al. [27] discuss the problem of locating hidden exon boundaries in cDNA. This leads to a game in which the hidden object is a subset $A \subseteq \{1, \ldots, n\}$ and the queries are of the type "Given an interval I, does it contain an element of A?". In a certain sense their problem is the dual of ours: they use intervals to locate and identify points; we use points to locate and identify intervals. Beigel et al. [2] discuss the problem of closing gaps in DNA sequencing data. This problem can be formulated as a search for a hidden perfect matching in a complete graph using queries "Given an induced subgraph, does it contain at least one matching edge?" McConnell and Spinrad [18] consider the tangentially related problem of reconstructing an interval graph given probes about the neighbors of only a partial set of vertices.

1.1 Terminology

An interval graph may have a number of different representations by intervals. In what follows, when we say "interval graph," we presume that one representation has been fixed. Without loss of generality, we may assume that in this representation all intervals are closed, have length at least one, and their end points are integers between 1 and $2n$, where n is the number of intervals.[1] We denote the interval of the ith vertex by $I_i = [s_i, f_i]$, where $s_i < f_i$ are integers. An edge (i, j) thus exists if $I_i \cap I_j \neq \emptyset$.

The complement $\overline{G}$ of an interval graph G has a special structure. Assume that (i, j) is not an edge in G, i.e., $I_i \cap I_j = \emptyset$. Then either $f_i < s_j$ or $f_j < s_i$, and thus we can orient the edge in $\overline{G}$ as $i \to j$ or $j \to i$. Thus, $\overline{G}$ has a natural orientation of the edges, and this orientation is well-known to be acyclic and transitive. For this and other results about interval graphs, see e.g. [12].

We will deal with discovering an initially unknown independent set in G chosen by an adversary, and will refer to this set as the *hidden independent set*. If V' is an independent set in G, then it is a clique in the complement graph $\overline{G}$. If G is an interval graph, then any clique in $\overline{G}$ has a unique topological order consistent with orientation of its edges. We can thus consider V' as a (directed) path π in $\overline{G}$, and will speak of a *hidden (directed) path* instead of a hidden independent set. We will generally omit the word "directed" as we will not be talking about any other kind of path.

We determine the hidden independent set through *probes* and *queries*. A *probe* is a unit open interval $(a, a + 1)$ where a is integer. A *query* is the use of a probe to determine information about the hidden independent set. Specifically, a query is a statement of the form: "Is there some vertex in the hidden independent set whose interval intersects the probe?" A query can be answered either "yes" or "no."[2]

Note that no such query can ever distinguish between two identical intervals. For this reason, we will assume that the input graph has no two identical intervals. On the other

[1] It is well-known that every interval graph can be represented in such a way. Moreover, one can easily verify that such a modification does not change the set of allowed queries in the graph (see definition of query below).

[2] Note that since intervals begin and end at integers, probing with a unit interval is equivalent to probing at a non-integral point. Probing with intervals arises naturally in our applications.

Figure 1: Different graphs may require different number of queries.

hand, intervals are allowed to have the same start point or the same end point.[3]

1.2 Our Results

Suppose we are given an interval graph with a fixed interval representation and want to determine a hidden independent set X in that graph. We study two versions of the problem:

1. Given an independent set Y, use queries to verify whether X is Y. We call this the *verification problem* and study it in Section 2.

2. Use queries to discover X without any other information. We call this the *discovery problem* and study it in Section 3.

Our results are summarized as follows. For the verification problem, we give a protocol to determine whether $X = Y$ using the exact optimal number of queries for that specific instance.

For the discovery problem, we give a linear-time algorithm for discovering X. Different graphs may require different number of queries to discover the hidden independent set. For example, both instances in Figure 1 are of the same size, but we can find the hidden independent set in instance (a) in $O(\log n)$ queries, while instance (b) requires $\Omega(n)$ queries. If at most a constant number of intervals start at a common point, then our protocol is within a constant factor of the optimal number of queries for that specific graph. That is, our algorithm is instance-optimal in the sense of Fagin et al. [11] and optimally adaptive in the sense of [7]. If this assumption is not satisfied, then the number of queries may be larger than the information-theoretic lower bound; however, we also prove stronger lower bounds to show that the number of queries must be larger in some of these cases.

1.3 Application to Gene Finding

In this section, we explain how our game of finding hidden independent sets in interval graphs relates to a problem in computational biology.

Recent technologies in molecular biology allowed researchers to determine genomic sequences of several organisms. These sequences need to be annotated. That is, biological meaning needs to be assigned to particular regions of the sequence. An important step in

[3]Every interval graph has a representation by intervals with distinct end points. However, modifying the graph to such a representation changes the set of allowed queries and hence the problem.

the annotation process is the identification of genes, which are the portions of the genome producing the organism's proteins.

A gene is a sequence of disjoint regions—called exons—of the genomic sequence. Exons are cut out and spliced together in the process of protein production. Thus each exon is an interval of the DNA sequence and a gene is a set of non-overlapping intervals.

There are a number of computational tools for gene prediction (e.g., [4, 23]); however, experimental studies (e.g., [19, 6]) show that the best of them predicts, on average, only about 50% of the entire genes correctly. It is therefore important to have alternative methods that can produce or verify such predictions experimentally.

While genes cannot be reliably predicted by purely computational means, these methods can provide a set of candidate exons. Algorithms for gene prediction have to balance sensitivity (how many real exons they discover) with specificity (how many false exons they predict). Usually it is possible to increase sensitivity at the expense of a decrease in specificity. Using a highly sensitive method, we may generate a candidate set containing many false exons with small probability of excluding a real exon.

To apply our algorithms, we may view the set of candidate exons as the set of intervals defining an interval graph. The gene we want to discover then corresponds to a hidden independent set in this interval graph (since a gene is a set of non-overlapping intervals from the candidate set). Queries in our algorithms correspond to the question: "Is a given short region of DNA sequence contained in a real exon?" To use our method for finding genes, we need to answer this question by appropriate biological experiments.

Thousands of such queries can be answered simultaneously by an expression array experiment [25]. Shoemaker et al. [26] have used expression arrays to verify gene predictions in annotation of human chromosome 22 [8]. In their approach, they probed DNA sequence at short regular intervals (every 10 nucleotides). Using our algorithm for independent set verification (Section 2), we can design a smaller set of queries that can verify the gene prediction, thus reducing the cost of such an experiment.

Queries similar to ours can be also implemented using polymerase chain reaction (PCR) technology [24]. A PCR experiment can answer a query of the following form: "Given two short regions of the DNA sequence, do both of them occur in the same gene (possibly in two different exons)?" The answer to our query can be obtained by PCR, provided that we already know at least one short region of exonic DNA which occurs in our gene. Our algorithm for the independent set discovery (Section 3) then yields an experimental protocol for finding genes.

However, many aspects of the real experimental domain further restrict the set of possible queries and would need to be addressed to apply this technique in practice (see e.g., [6]). This application of PCR technology was inspired by open problem 12.94 in [20]. PCR queries have also been used in similar way to determine exon boundaries in cDNA clones [27].

1.4 Application to 1-dimensional Battleship

The game of Battleship (also known as Convoy and Sinking-Ships or in a solitaire variant, FathomIt) is a well-known two-person game. Both players have an $n \times n$ grid and a fixed set of ships, where each ship is a $1 \times k$ rectangle for some $k \leq n$. Each player arranges the ships

on his/her grid in such a way that no two ships intersect. Then players take turns shooting at each other's ships by calling the coordinates of a grid position. The player that first sinks all ships (by hitting all grid positions that contain a ship) wins.

There are many variants of Battleship (see e.g., [1]) involving other ship shapes or higher dimensions. In an *offline* variation of the problem, the collection of shots must cover the d-dimensional lattice in order to hit all rectangles with at least a given volume [5, 17].

We can rephrase Battleship as a graph problem as follows. Define a graph G with one vertex for every possible ship position. Two vertices in the graph are adjacent if and only if the corresponding ship positions intersect or touch. The positions that the adversary chooses for his/her ships then correspond to a hidden independent set in graph G. The only operation allowed for discovering a ship is choosing a point of the grid and asking whether it is covered by a ship, which corresponds to querying a set of vertices in the graph.

For the standard Battleship game, the graph G is what is known as a *boxicity-2* graph, i.e., it is the intersection graph of two-dimensional axis-aligned rectangles (see e.g., [21]). In fact, it is an even more specialized graph since all rectangles have unit width or unit height. Such graphs are the same as intersection graphs of horizontal or vertical line segments. Recognizing such graphs is NP-hard [16], and one can easily show that finding a maximum independent set in them is NP-hard as well. We are not aware of any results concerning finding hidden independent sets in these graphs.

Graph G becomes an interval graph if we study a simplified version of Battleship that operates in 1-dimensional space. Here the ships are intervals with integral end points, and, as before, no two intersecting ship positions may be taken. The allowed operations are now exactly our queries: given an open unit interval $(a, a+1)$, does one ship overlap this interval?

2 Independent Set Verification

In this section, we will give a polynomial-time algorithm for the verification problem: given an interval graph G and an independent set Y in G, find the minimum number of probes that can determine whether $X = Y$, where X is the hidden independent set chosen by the adversary.

There are two types of queries: the ones for which the probe intersects some interval in Y (we call this a *positive probe*) and the ones for which it does not (we call this a *negative probe*). For a probe the *expected answer* is the answer that is consistent with $X = Y$. Thus, a positive probe has expected answer "yes," while a negative probe has expected answer "no."

Consider an algorithm to solve the verification problem. If for some query it does not get the expected answer, then $X \neq Y$ and the algorithm can terminate. Otherwise the algorithm must continue until enough queries are asked to determine that $X = Y$. Thus the worst case for any optimal verification algorithm is when $X = Y$ (i.e., all answers are as expected).

This observation implies that we can rephrase the verification problem as follows: for a given graph G and an independent set Y, produce a set of queries U such that Y is the only independent set in G consistent with the expected answers to all queries in U. Any algorithm that creates queries interactively based on answers to the previous questions can

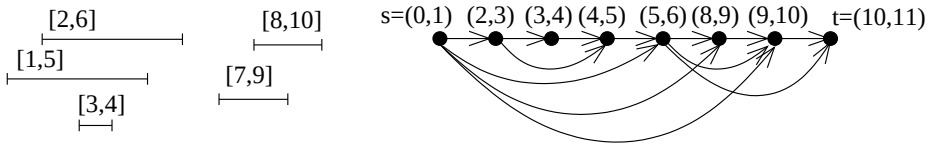

Figure 2: An interval graph and its corresponding graph H for $Y = \{[2,6],[8,10]\}$. For example, edge $(5,6) \to (9,10)$ exists because the independent set $\{[7,9]\} \in G[6,9]$ intersects all positive probes between 6 and 9.

be transformed to an algorithm solving the rephrased problem without changing the worst-case number of queries (we simply simulate the algorithm by providing the expected answer for each query and gather all queries produced in this way). We say that a set of queries U *verifies* that $X = Y$ if every independent set $Z \neq Y$ is inconsistent with the expected answer of at least one query in U; we say that this query *eliminates* Z.

In this section we give a polynomial-time algorithm that discovers the minimum set of queries needed to verify that $X = Y$. First we will study a special case in which only queries with positive probes are allowed. This case is then used as a subroutine for the general case.

2.1 Finding a Minimum Set of Positive Probes

We first study the special case where only positive probes are allowed. Note that for some inputs it is impossible to verify $X = Y$ using only positive queries.

Sometimes we will consider only intervals inside some region $[a,b]$. Let $G[a,b]$ denote the subgraph of G induced by intervals completely contained in the region $[a,b]$. Similarly, for any independent set Z, let $Z[a,b]$ denote the subset of Z of intervals completely contained in the region $[a,b]$.

The minimum set of positive probes for a graph G will be computed using a directed acyclic graph H defined as follows. Graph H contains one vertex for every positive probe. Let a_{min} be the smallest start point and a_{max} be the largest end point of an interval in G. Two additional vertices s and t are added, where s corresponds to probe $(a_{min}-1, a_{min})$ and t corresponds to probe $(a_{max}, a_{max}+1)$. Note that these probes are negative for G.

Intuitively, H contains a directed edge from one probe to another if no positive probe between them can distinguish Y from some other independent set. More precisely, for any $a < b$, graph H contains an edge $e_{a,b}$ from $(a, a+1)$ to $(b, b+1)$ if and only if there is an independent set $Z_{a,b}$ in $G[a+1,b]$ that intersects all positive probes $(c, c+1)$ with $a < c < b$ and that is different from Y. See Figure 2 for an example of graph H.

Graph H has $O(n)$ vertices and $O(n^2)$ edges, where n is the number of intervals. Using dynamic programming, it can be constructed in $O(n^2)$ time, see Section 2.3. The following two lemmas show the connection between graph H and the optimal set of positive queries.

Lemma 1. *It is possible to verify that $X = Y$ by a set of positive probes if and only if vertices s and t are not connected by an edge in H.*

Proof. Edge $e_{s,t}$ exists if and only if there is an independent set $Z_{s,t}$ in graph G that intersects all positive probes and that is different from Y. But this means that $Z_{s,t}$ and Y cannot be distinguished by positive probes. □

Lemma 2. *A set of positive probes U verifies that $X = Y$ if and only if vertices s and t become disconnected in graph H after removal of all vertices in U.*

Proof. On the one hand, suppose that U is a set of positive probes verifying that $X = Y$. Let π be a path in H from s to t. We will prove that π must contain a vertex from U.

Define the set of intervals Z_π *corresponding to path* π as the union of the independent sets $Z_{a,b}$ over all edges $e_{a,b} \in \pi$. Note that Z_π is an independent set because for any edge $e_{a,b}$ in π, the independent set $Z_{a,b}$ has intervals with points between $a+1$ and b. Graph H does not contain edge $e_{s,t}$; otherwise $X = Y$ could not be verified by Lemma 1. So π contains at least one vertex $(u, u+1) \neq s, t$. Let $e_{a,u}$ and $e_{u,b}$ be the incoming and outgoing edge of $(u, u+1)$ in π. Then $Z_{a,u}$ is in $G[a+1, u]$ and $Z_{u,b}$ is in $G[u+1, b]$. So neither independent set intersects the positive probe $(u, u+1)$. Therefore Z_π cannot intersect the positive probe $(u, u+1)$, and thus $Z_\pi \neq Y$.

Because $Z_\pi \neq Y$, there must be a probe $(v, v+1) \in U$ inconsistent with Z_π. Suppose for contradiction that $(v, v+1) \notin \pi$. Thus π "jumps" over this vertex using edge $e_{a,b}$, where $a < v < b$. However, set $Z_{a,b} \subseteq Z_\pi$ must then contain an interval intersecting probe $(v, v+1)$, contradicting that Z_π is inconsistent with $(v, v+1)$. Therefore, $(v, v+1) \in \pi$, which means that removing U interrupts all paths from s to t as desired, and π contains a vertex in U.

On the other hand, suppose that set U disconnects vertices s and t in H. Let $Z \neq Y$ be an independent set in H. We will prove that Z is inconsistent with at least one probe from U.

Let $S = \{(s_1, s_1+1), (s_2, s_2+1), \ldots, (s_k, s_k+1)\}$ be the set of all positive probes inconsistent with Z. Without loss of generality let $s_1 < s_2 < \cdots < s_k$; let $s_0 = s$ and $s_{k+1} = t$. Note that for $0 \leq i \leq k$, the independent set $Z[s_i + 1, s_{i+1}]$ defines edge $e_{s_i, s_{i+1}}$. Thus we can form a path π in graph H from the edges $e_{s_i, s_{i+1}}$ over all $0 \leq i \leq k$.

Path π connects vertices s and t in H, so in particular π contains at least one vertex $(u, u+1) \in U$. By the definition of π, we must have $(u, u+1) \in S$ and thus Z is inconsistent with probe $(u, u+1)$. □

Thus the minimal set of positive probes to verify $X = Y$ corresponds to the smallest set of vertices in H that disconnect s and t. This vertex-connectivity problem can be solved in $O(n^{8/3})$ time using network flows. Since we want to use this as a subroutine in the general case, we expand the result to any subgraph $G[a, b]$ of G. On such a subgraph we need to verify that $X[a, b] = Y[a, b]$. The following lemma shows the details of the algorithm.

Definition 1. *Let $A_+[a, b]$ be the smallest number of positive probes needed to verify that $X[a, b] = Y[a, b]$ in $G[a, b]$, or $A_+[a, b] = \infty$ if this is not possible.*

Lemma 3. *Value of $A_+[a, b]$ can be computed in $O(n^{8/3})$ time.*

Proof. Consider a directed acyclic graph H for graph $G[a, b]$ defined as in Lemma 2. We will show how to compute such a graph efficiently in Section 2.3. First transform the graph H into a graph H' by replacing each vertex $i \in H \setminus \{s, t\}$ by a directed edge (i', i''). All edges entering i in H will go to i' in H' and all edges leaving i in H leave from i''. Instead of finding the smallest set of vertices disconnecting s from t in H (vertex cut), we will search for the smallest set of edges disconnecting s from t in H' (edge cut). Obviously, any vertex

cut in H is an edge cut in H'. On the other hand, if an edge cut in H' contains some edge (i'', j'), we can instead cut either (i', i'') or (j', j'') (at least one of i, j is neither s nor t). Therefore we can obtain a minimum edge cut with only edges of the type (i', i''), and these clearly correspond to a vertex cut in H. The minimum edge cut separating s from t can be found using a unit-capacity maximum-flow algorithm for directed graphs, in $O(n^{8/3})$ time [13, 10]. □

2.2 Finding a Minimum Set of Probes in the General Case

The general case, in which both positive and negative probes are allowed, is solved by a dynamic programming algorithm that has the result of Lemma 3 as a base case.

Definition 2. *Let $A[a]$ be the smallest number of queries needed to verify that $X[1, a] = Y[1, a]$ in the interval graph $G[1, a]$.*

Lemma 4. *$A[a]$ satisfies the following recursive formula:*

$$A[a] = \min \begin{cases} A_+[1, a], \\ \min_b A[b] + A_+[b + 1, a] + 1, \end{cases}$$

where the inner minimum is taken over all probes $(b, b + 1)$ that are negative probes intersecting $[1, a]$.

Proof. If the optimal solution of subproblem $A[a]$ contains only positive queries, then $A[a] = A_+[1, a]$. Otherwise let $(b, b + 1)$ be the rightmost negative probe in it. All probes to the right of b are positive and they comprise a solution of $A_+[b + 1, a]$. Probes to the left of b comprise a solution of $A[b]$. Therefore in this case we have $A[a] = A[b] + A_+[b+1, a] + 1$. □

2.3 Algorithm Details

Lemma 4 gives a recursive formula for computing $A[1, a]$ using the values $A_+[a, b]$. These values can be computed using the result of Lemma 3, but a method is still needed for finding the edges of H. First we define an auxiliary table $E_{a,b}$ and show how to compute its values. Then we show how to use this table to obtain the edges of $H[a, b]$ corresponding to $G[a, b]$.

Definition 3. *Let $E_{a,b}$ be the number of independent sets in graph $G[a + 1, b]$ that intersect every positive probe $(c, c + 1)$ inside $[a + 1, b]$ (i.e. $a < c < b$).*

Lemma 5. *The values of $E_{a,b}$ can be computed in $O(n^2)$ time for all $a \leq b$.*

Proof. Let $S_{a,b}$ be the set of all intervals $[c, b]$ in graph $G[a + 1, b]$ such that $(c - 1, c)$ is a negative probe or it is equal to $(a, a + 1)$. The values $E_{a,b}$ can then be computed using the following recursive formula.

$$E_{a,b} = \begin{cases} 1 & \text{if } a = b \text{ or } a + 1 = b \\ \displaystyle\sum_{[c,b] \in S_{a,b}} E_{a,c-1} & \text{if } a + 1 < b, \ (b - 1, b) \text{ positive} \\ E_{a,b-1} + \displaystyle\sum_{[c,b] \in S_{a,b}} E_{a,c-1} & \text{if } a + 1 < b, \ (b - 1, b) \text{ negative} \end{cases}$$

The base case happens if $a = b$ or $a + 1 = b$. Then the only independent set satisfying the criteria is the empty one. Let us assume now that $a + 1 < b$. There are two cases. If the probe $(b - 1, b)$ is positive, then it must intersect an interval in the independent set. This interval must end in b. Thus we go through all such intervals and sum up the possibilities. However, if the interval $[c, b]$ is in an independent set, then this set does not intersect $(c-1, c)$. Therefore $[c, b]$ can be used only if $(c - 1, c)$ is a negative probe or it is equal to $(a, a + 1)$. If the probe $(b-1, b)$ is negative, all the possibilities from the case with positive probe $(b-1, b)$ are valid, but we also need to add independent sets that do not intersect $(b - 1, b)$. These are stored in $E_{a,b-1}$.

Let S_b be the set of intervals ending in b. The time needed to compute $E_{a,b}$ is $O(1) + O(|S_b|)$ (because $S_{a,b} \subseteq S_b$). Therefore total time to compute all $E_{a,b}$ is $O(n^2) + n \sum_b |S_b|$. However, every interval can be only in one set S_b, therefore $\sum_b |S_b| = n$, and total time is $O(n^2)$.

□

Lemma 6. *Let $H[a, b]$ be the directed acyclic graph from Lemma 2 corresponding to the graph $G[a, b]$ and a given path $Y[a, b]$. Then the edges of $H[a, b]$ can be computed in $O(n^2)$ time.*

Proof. For any two positive probes $(u, u + 1)$ and $(v, v + 1)$ inside $[a + 1, b]$, we know by definition that $e_{u,v} \in H[a, b]$ if and only if $E_{u,v} > 0$. The only issue is that the existence of edge $e_{a,b}$ requires that the independent set $Z_{a,b}$ is different from $Y[a, b]$, but that is true because it does not intersect positive probes $(u, u + 1)$ and $(v, v + 1)$.

We also need to consider edges incident to s and t. Vertex s corresponds to probe $(a-1, a)$. Notice that the value $E_{a-1,b}$ is influenced only by the intervals of G that are inside $G[a, b]$. Therefore, there is an edge from s to a positive probe $(u, u + 1)$ inside $[a + 1, b]$ if and only if $E_{a-1,u} > 0$. Similarly, vertex t corresponds to probe $(b, b + 1)$ and there is an edge from $(u, u + 1)$ to t if and only if $E_{u,b} > 0$.

Edge $e_{s,t}$ is different, because s and t are both negative probes in $G[a, b]$ and thus $Y[a, b]$ is included in the count $E_{a-1,b}$. Therefore $e_{s,t} \in H[a, b]$ if and only if $E_{a-1,b} > 1$.

Because graph H has $O(n)$ vertices and for each two vertices their adjacency can be obtained by a simple lookup in $O(1)$ time, we have the required bound. □

The overall computation can be organized as follows. First, table $E_{a,b}$ is computed in $O(n^2)$ time (Lemma 5). Then we run the dynamic program according to Lemma 4. Each time a value $A_+[a, b]$ is required, we construct graph $H[a, b]$ in $O(n^2)$ time according to Lemma 6. If edge $e_{s,t}$ does not exist, we compute the smallest number of vertices separating s and t according to Lemma 3. This number is equal to $A_+[a, b]$. If edge $e_{s,t}$ exists, $A_+[a, b] = \infty$. Notice that each $A_+[a, b]$ is used at most once, so it is unnecessary to store them. The overall time is dominated by the computation of $A_+[a, b]$ for all $a < b$. Thus the overall time is $O(n^4 + n^2 T)$ where T is the time to find the smallest (s, t)-cut in a network ($T \in O(n^{8/3})$, see Lemma 3). This yields the following result:

Theorem 1. *Given an n-vertex interval graph G and an independent set Y in G, we can find in $O(n^{14/3})$ time the minimum set of queries that verifies whether Y is the hidden independent set chosen by an adversary.*

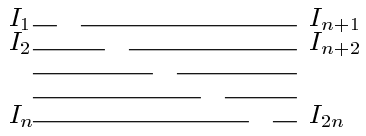

Figure 3: The staircase needs $n-1$ queries.

3 Independent Set Discovery

In this section, we study the discovery problem. In it, we are given an interval graph G, and we want to find some hidden independent set X with queries of the form $(a, a+1)$. We will give an interactive protocol to find X, i.e., the next query depends on the outcome of the previous query. The protocol uses an asymptotically optimal number of queries if at most constant number of intervals start at a common point.

We start by giving lower bounds for how many queries are needed. A simple information-theoretic argument yields the following lower bound, which holds for any graph and any type of query.

Theorem 2. *Assume that G is a graph that contains p independent sets. Regardless of the types of yes/no queries allowed, we need at least $\lceil \log_2 p \rceil$ queries to find a hidden independent set X in the worst case.*

Proof. We use a decision tree argument. Build a decision tree with the posed queries at each interior node, and the resulting independent set at the leaves. Each query yields a yes/no answer, so each interior node has at most two children. Since the decision tree has at least p leaves, it must have a leaf of depth at least $\lceil \log_2 p \rceil$. Since X is hidden, the adversary can choose exactly the independent set at this leaf for X, resulting in $\lceil \log_2 p \rceil$ queries to find X. □

We do not always get a tight bound, even for an interval graph. Consider the so-called *staircase* depicted on Figure 3. It consists of $2n$ intervals, with interval $I_i = [0, 2i-1]$ for $i = 1, \ldots, n$ and $I_i = [2(i-n), 2n+1]$ for $i = n+1, \ldots, 2n$. In this case we have $n(n+1)/2 + 2n + 1$ independent sets, which gives a lower bound of $2\log_2 n + O(1)$ queries. A stronger lower bound can be shown as follows.

Theorem 3. *The staircase with $2n$ intervals requires $n-1$ queries in the worst case.*

Proof. The adversary decides that the hidden independent set X will be $\{I_j, I_{n+j}\}$ for some j, i.e., one of the n pairs of intervals with the same y-coordinate in Figure 3.

Assume that the algorithm uses only $k \leq n-2$ queries and the adversary answers each of these queries "yes". So let $(a, a+1)$ be an arbitrary probe for a query, where $0 \leq a \leq 2n$ is an integer. If a is even, probe $(a, a+1)$ intersects all independent sets of the form $\{I_j, I_{n+j}\}$. If a is odd, say $a = 2i-1$, then it intersects all such independent sets except $\{I_i, I_{n+i}\}$.

Since the algorithm used $k \leq n-2$ queries and with each query there was at most one pair $\{I_j, I_{n+j}\}$ not intersecting the query, there are at least two such pairs that intersect all queries. Each of them can be a correct answer. □

The lower bound of $\lceil \log_2 p \rceil$ queries from Theorem 2 can be matched (asymptotically) under some assumptions. To show this, we will give a protocol that discovers a hidden independent set in $O(\log p)$ queries, where p is the number of independent sets, under the assumption that at most a constant number of intervals start at the same point. This is not a contradiction to Theorem 3, because in the staircase example, many intervals start at the same point. For this protocol, we will adopt the point of view of the complement graph, and, as explained in Section 1.1, search for a hidden (directed) path in a directed acyclic transitive graph.

3.1 Overview of the Algorithm

The algorithm to detect the hidden path is recursive. The crucial idea is that with a constant number of queries we eliminate at least a constant fraction of the remaining paths. Therefore, after $O(\log p)$ queries, we know the correct path.

To explain how to find the next query at any given time, we need some notations. The following terms are defined relative to all those intervals that have not yet been eliminated. Note that these values, and in particular the value of f_i to be defined, will change as more intervals are eliminated.

So fix one moment of time, and let $I_1, \ldots, I_n$ be the set of intervals that have not yet been eliminated. We assume that the intervals $I_1, \ldots, I_n$ are sorted by increasing start point, breaking ties arbitrarily. Let I_i be the interval that ends first, i.e., $f_i \leq f_j$ for all $j = 1, \ldots, n$, breaking ties arbitrarily. Our next query will happen at or near interval I_i, and thus affect all those intervals that intersect I_i. We call these intervals the *clique intervals*; more precisely, the clique intervals are the intervals $I_1, \ldots, I_k$ with k such that $s_k \leq f_i$ and $s_{k+1} > f_i$. Note that all clique intervals intersect point f_i; hence, as the term suggests, they form a clique in G, and at most one of them is in any path.

Our algorithm operates under two different scenarios. Let a *legal path* be a path in the graph that could be the solution even under the following additional restrictions. In the *unrestricted scenario*, any path is a legal path; this is the scenario at the beginning of the algorithm. In the *restricted scenario*, only a path that intersects (f_{i-1}, f_i) is legal (we will have obtained this information through previous queries). Any legal path thus uses a clique interval that starts strictly before f_i, and we can eliminate all clique intervals that start at f_i.

3.2 Effects of Queries

The algorithm uses only one kind of query: we always query at $(a, a+1)$ for some $a \leq f_i$. Only clique intervals can intersect the probe (though not all of them necessarily do).

After each query we eliminate all legal paths that are not consistent with the answer to the query. More precisely, if the answer to a query at $(a, a+1)$ is "no", then we eliminate all clique intervals that intersect $(a, a+1)$. If the original scenario was unrestricted, then all remaining paths are consistent with this query and we can solve the problem recursively in the unrestricted scenario.

If the original scenario was restricted, we already know that one of the clique intervals

$I_1, \ldots, I_k$ is in the hidden path X. Elimination of some clique intervals may increase the value of f_i and therefore add some more intervals to the clique intervals. None of these new clique intervals can be in X, and thus they can also be eliminated. Then we solve the restricted scenario recursively on the new graph.

Assume now that the answer to a query with some probe $(a, a+1)$ is "yes". Since X contains at most one clique interval, all clique intervals not intersecting $(a, a+1)$ can be eliminated. One of the remaining clique intervals will be part of the solution, so the next scenario will be restricted. We also can eliminate all intervals that become clique intervals due to an increase in f_i.

If in the new situation we are now in the restricted scenario with only one clique interval I_1, then I_1 belongs to X. Therefore, interval I_1 can be eliminated from the graph and we solve the unrestricted scenario on the resulting graph recursively. Afterwards we add I_1 to get the hidden path X.

3.3 Some Definitions and Observations

Before specifying how we actually choose the queries, we need some definitions and useful observations. Fix one point of time when we want to find the next query.

Let P_{legal} be the set of all legal paths. Since every legal path contains at most one clique interval, we can partition P_{legal} as $P_{legal} = P_1 \cup \cdots \cup P_k \cup P_{rest}$, where P_j is the set of legal paths that use clique interval I_j, and P_{rest} denotes the legal paths that do not use a clique interval. (P_{rest} is empty in the restricted scenario.) Define $p_\beta = |P_\beta|$ for all subscripts β.

Claim 1. *In the unrestricted scenario, $p_i = p_{rest}$.*

Proof. For every path π in P_i, we can obtain a path π' by deleting the first interval (which is I_i) in π. Note that any path contains at most one clique interval, and P_{rest} includes the empty path, so π' is a path in P_{rest} and $p_i \leq p_{rest}$.

For the other direction, let π be a path in P_{rest}. Since π does not contain a clique interval, none of its intervals intersects I_i (by definition of a clique interval). Hence we can obtain a path π' in P_i by adding I_i to π, and $p_{rest} \leq p_i$. □

Claim 2. $p_{rest} \leq \frac{1}{2} p_{legal}$.

Proof. This holds trivially in the restricted scenario since $p_{rest} = 0$. In the unrestricted scenario, we have one path in P_i for every path in P_{rest} by Claim 1, hence P_{rest} contains at most half of all paths. □

Claim 3. *If I_{j_1} and I_{j_2} are clique intervals with $f_{j_1} \leq f_{j_2}$ then $p_{j_1} \geq p_{j_2}$.*

Proof. For any path $\pi \in P_{j_2}$, we can obtain a path $\pi' \in P_{j_1}$ by removing the first element of π and inserting I_{j_1} instead. This is a legal path because the first element of π must be I_{j_2} (since I_{j_2} is a clique interval), and I_{j_1} does not end after I_{j_2}. □

Now we can also refine the analysis of the effects of some queries.

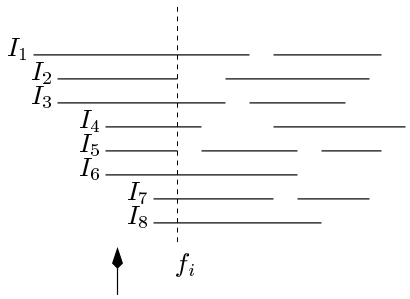

Figure 4: A query at $(s_5, s_5 + 1)$ eliminates $p_1 + \ldots + p_6$ paths if the answer is "no", or $p_7 + p_8 + p_{rest}$ paths if the answer is "yes".

Lemma 7. *If we query at $(s_j, s_j + 1)$ for some j with $s_j < f_i$, then we can eliminate either $p_1 + \cdots + p_{j'}$ paths or $p_{j'+1} + \cdots + p_k + p_{rest}$ paths, where $j' \geq j$ is the largest index with $s_{j'} = s_j$.*

Proof. If the answer to the query is "no", then we can eliminate all clique intervals that intersect $(s_j, s_j + 1)$; since $s_j < f_i$ these are the intervals $I_1, \ldots, I_{j'}$ and we eliminate $p_1 + \cdots + p_{j'}$ paths.

If the answer to the query is "yes", then the solution contains an interval that intersects (s_j, s_j+1); since $s_j < f_i$ this must be a clique interval and all paths in P_{rest} can be eliminated. Furthermore, by choice of j' the clique intervals $I_{j'+1}, \ldots, I_k$ do not intersect $(s_j, s_j + 1)$ and can be eliminated as well. □

3.4 Choosing Queries

In light of Lemma 7 we will try to find a j such that both sets of possibly eliminated paths contain a constant fraction of the paths. To find such a j, define $1 \leq \ell \leq k$ to be the index such that

$$p_1 + \cdots + p_{\ell-1} < \tfrac{1}{2} p_{legal} \quad \text{and} \quad p_1 + \cdots + p_{\ell-1} + p_\ell \geq \tfrac{1}{2} p_{legal}; \tag{1}$$

this is well-defined because $p_1 + \cdots + p_k \geq \frac{1}{2} p_{legal}$ by Claim 2. Define ℓ^- and ℓ^+ to be the smallest/largest index such that $s_{\ell^-} = s_\ell = s_{\ell^+}$. Thus $\ell^- \leq \ell \leq \ell^+$. We distinguish three cases:

C1: $p_1 + \cdots + p_{\ell^- - 1} \geq \frac{1}{4} p_{legal}$ and $p_{\ell^-} + \cdots + p_k + p_{rest} \geq \frac{1}{4} p_{legal}$:

In this case, query at the beginning of $I_{\ell^- - 1}$, i.e., at $(s_{\ell^- - 1}, s_{\ell^- - 1} + 1)$. By definition of ℓ^-, intervals $I_{\ell^- - 1}$ and I_{ℓ^-} have distinct starting points, so by Lemma 7 this eliminates at least $\frac{1}{4} p_{legal}$ paths.

C2: $p_1 + \cdots + p_{\ell^+} \geq \frac{1}{4} p_{legal}$ and $p_{\ell^+ + 1} + \cdots + p_k + p_{rest} \geq \frac{1}{4} p_{legal}$:

In this case, query at $(s_{\ell^+}, s_{\ell^+} + 1)$. By Lemma 7 this eliminates at least $\frac{1}{4} p_{legal}$ paths.

C3: Neither (C1) nor (C2) holds:

In this case, we query with probe $(f_i, f_i + 1)$. Note that this query is not covered by Lemma 7, and we will analyze its effects separately.

In both cases (C1) and (C2) we eliminate at least a constant fraction of the legal paths, and hence the number of such queries is at most $O(\log p)$. The analysis is more intricate in case (C3). We need a few observations.

Lemma 8. *If cases (C1) and (C2) do not hold, then* $p_{\ell^-} + \cdots + p_{\ell^+} > \frac{1}{2}p_{legal}$.

Proof. By definition of ℓ, we have $p_\ell + \ldots + p_k + p_{rest} = p_{legal} - (p_1 + \ldots + p_{\ell-1}) > \frac{1}{2}p_{legal}$. Since $\ell^- \leq \ell$ therefore $p_{\ell^-} + \ldots + p_k + p_{rest} > \frac{1}{2}p_{legal}$. So if (C1) does not hold, then

$$p_1 + \ldots + p_{\ell^- - 1} < \frac{1}{4}p_{legal}.$$

Also by definition of ℓ, we have $p_1 + \ldots + p_\ell \geq \frac{1}{2}p_{legal}$. Since $\ell \leq \ell^+$ therefore $p_1 + \ldots + p_{\ell^+} \geq \frac{1}{2}p_{legal}$. So if (C2) does not hold, then

$$p_{\ell^+ + 1} + \ldots + p_k + p_{rest} < \frac{1}{4}p_{legal}.$$

Thus there are more than $\frac{1}{2}p_{legal}$ paths left that are not covered in either equation, and these must belong to $P_{\ell^-}, \ldots, P_{\ell^+}$. □

Denote by θ the maximum number of intervals that have a common start point (i.e., $l^+ - l^- + 1 \leq \theta$).

Lemma 9. *A positive answer to a query in case (C3) eliminates at least* $p_i \geq \frac{1}{2\theta}p_{legal}$ *paths.*

Proof. Since we obtain a positive answer at a query $(f_i, f_i + 1)$, none of the clique intervals that end at f_i can be in the hidden path. So we can eliminate these intervals, and in particular eliminate interval I_i and p_i paths.

By Claim 3 we have $p_i \geq p_{\ell^-}, \ldots, p_{\ell^+}$. By Lemma 8 furthermore $p_{\ell^-} + \cdots + p_{\ell^+} > \frac{1}{2}p_{legal}$. The intervals $I_{\ell^-}, \ldots, I_{\ell^+}$ all start at s_ℓ, therefore there are at most θ of them, and

$$p_i \geq \max\{p_{\ell^-}, \ldots, p_{\ell^+}\} \geq \frac{1}{\theta}(p_{\ell^-} + \cdots + p_{\ell^+}) \geq \frac{1}{\theta}\frac{1}{2}p_{legal}.$$

□

Now we turn to the case when the query in (C3) yields a negative answer. This is the only case where possibly less than a constant fraction of paths is eliminated, but we account for this query in a different way. First we observe the following.

Lemma 10. *In case (C3) at least one clique interval intersects* $(f_i, f_i + 1)$.

Proof. Assume that no clique interval intersects $(f_i, f_i + 1)$, thus all clique intervals end at f_i by definition of i. Therefore all clique intervals have distinct starting points (recall that all intervals are distinct), and $\ell^- = \ell = \ell^+$. By Lemma 8 therefore $p_\ell > \frac{1}{2}p_{legal}$.

Note that $\ell = i$, because otherwise by $p_i \geq p_\ell$ (Claim 3) and $p_\ell > \frac{1}{2}p_{legal}$ we would have $p_i + p_\ell > p_{legal}$, which is impossible. Furthermore, no interval other than I_i ends at f_i, because otherwise both would be contained in equally many paths (Claim 3), which

contradicts $p_i > \frac{1}{2}p_{legal}$. So there is only one clique interval, I_i. Finally, note that $p_i > \frac{1}{2}p_{legal}$ implies that we are in the restricted scenario by Claim 1.

So we have only one clique interval I_i and we are in the restricted scenario, which means that necessarily I_i belongs to X. Since we detect this beforehand (see Section 3.2), the algorithm does not search for a query in this case. □

Now we are ready to analyze the situation for a negative answer in case (C3).

Lemma 11. *During all recursive calls, we have at most* $\log_2 p$ *times a negative answer in case (C3), where* p *is the number of paths in the original graph.*

Proof. Let s be the number of such queries. We will show that the original graph contains an independent set of size s. Since every subset of it is also an independent set, we have $p \geq 2^s$, which yields the result.

Note first that we never do the same query with a negative answer twice in case (C3), for once we have obtained a negative answer at $(f_i, f_i + 1)$, we eliminate all intervals that intersect the probe. Hence by Lemma 10, we will not return to case (C3) until the value of f_i has changed. Thus for each negative answer in case (C3), we have a different value of f_i. Let $f_{i_1} < \cdots < f_{i_s}$ be these values, and for $1 \leq j \leq s$ let I_{i_j} be a clique interval that ends at f_{i_j} and was not eliminated when we queried at $(f_{i_j}, f_{i_j} + 1)$.

We claim that $I_{i_1}, \ldots, I_{i_s}$ is an independent set. For if two of them intersected, then they would have different end points since the f_{i_j}'s are distinct, and the query at the earlier-ending interval would eliminate the later-ending interval. Thus, we indeed have an independent set of size s, as desired. □

Now we are ready to establish the effectiveness of our querying scheme.

Lemma 12. *Assume we are given a set of* n *intervals that define* p *paths, and at most* θ *intervals start at the same point. Then any hidden path* X *can be found with at most* $\log_2 p + \max\{\log_{2\theta/(2\theta-1)} p, \log_{4/3} p\}$ *queries.*

Proof. Compute the queries as described above until we have found the hidden path, say with m queries. Some number s of these queries give a negative answer in case (C3); we know that $s \leq \log_2 p$. The remaining $m - s$ queries each eliminate at least $\frac{1}{4}p_{legal}$ or $\frac{1}{2\theta}p_{legal}$ paths at that time. Since we are done when only one path is left, we have $m - s \leq \log_{4/3} p$ (for $\theta \leq 2$) or $m - s \leq \log_{2\theta/(2\theta-1)} p$ (for $\theta > 2$). □

Note that for $\theta \leq 2$, the number of queries is at most $\log_2 p + \log_{4/3} p \approx 3.41 \log_2 p$, thus we are within a factor of 3.41 of the minimum number of queries. As long as θ is a constant, we use $O(\log_2 p)$ queries, which is asymptotically optimal. Assuming that θ is constant is quite realistic for 1D-battleships where typically there is only a limited number of types of ships.

3.5 Time Complexity

We now show how to implement the above algorithm such that finding all queries takes $O(n+m)$ time, where m is the number of edges in the complement of the interval graph. This time complexity holds if unit cost arithmetic is used, i.e., arbitrarily large numbers can be handled in $O(1)$ time. If we take the time for adding such numbers into account, the time complexity increases to $O((m+n)\log p)$, where p is the number of paths in the complement graph. Note that p may be exponential in n.

For easier maintenance, we group the intervals into bundles. Here, a bundle is a maximal set of intervals that all have the same start point, or a maximal set of intervals that all have the same end point. Each interval hence belongs to two bundles.

We maintain the following data structures:

- We store a list $\mathcal{S}$ of bundles of intervals with the same start point, and a list $\mathcal{E}$ of bundles of intervals with the same end point. Recall that all start and end points of intervals are integers between 1 and $2n$; we can therefore initialize $\mathcal{S}$ and $\mathcal{E}$ with two bucket sorts in $O(n)$ time.

- Within each bundle, the intervals are sorted by increasing value of the end point that is not equal. Each interval stores cross-references to the bundles that contain it and where it is stored in these bundles, so that it can be deleted from the structures in constant time. Each interval I_j also stores p'_j which is the number of paths that start at I_j. Note that $p_j = p'_j$ if I_j is a clique interval. This can be computed initially for all intervals with a reverse topological order in $O(m+n)$ time, since $p'_j = 1 + \sum_{I_j \to I_k} p'_k$.

- We store the current scenario in a flag.

- We store the current total number of paths p, and the current number p'_{rest} of paths that do not use a clique interval. Then p is simply the sum of all p'_j; p'_{rest} is initialized to p and will be updated later. We can compute p_{legal} and p_{rest} from p, p'_{rest} and the scenario-flag in constant time.

- Each bundle B stores a list of its intervals and also the number of paths $p(B)$ that start at an interval in this bundle. This can be computed initially in $O(n)$ total time by summing the p'_j over all intervals in the bundle.

- We store the clique intervals implicitly, by maintaining a reference to the first bundle B^* in $\mathcal{S}$ that does not contain clique intervals. We initialize B^* to be the first bundle in $\mathcal{S}$ and will update it during the algorithm.

All lists in our data structure are doubly-linked lists for easier deletion. Now each round of the algorithm proceeds as follows:

- Find the first bundle in $\mathcal{E}$. The first interval in this bundle is I_i, and its end point is f_i.

- For as long as the start point s of intervals in B^* satisfied $s \leq f_i$, advance B^* to be the next bundle in $\mathcal{S}$. With every advancement of B^*, subtract $p(B^*)$ from p_{rest}, since

these paths now start in clique intervals. If we are in the restricted scenario, all newly added clique intervals can be eliminated as discussed in Section 3.2.

The time required to do this is proportional to the number of bundles that we have advanced. We will study below what needs to be done to eliminate an interval.

- If there is only one clique interval I_i, and if we are in the restricted scenario, then add I_i to X, eliminate I_i, and move to the unrestricted scenario. This ends the round.
- Otherwise, find the second bundle in $\mathcal{E}$. If the start point s_j of the first interval in this interval satisfies $s_j > f_i$, then all clique intervals end at f_i.
- Check whether $p_i > \frac{1}{2\theta} p_{legal}$. If this is true, and if not all clique intervals end at f_i, then the next query is $(f_i, f_i + 1)$. This is case (C3).[4]
- If $p_i < \frac{1}{2\theta} p_{legal}$ or if all clique intervals end at f_i, then we are in case (C1) or (C2) (by Lemma 9 and Lemma 10). Thus, we now must search for ℓ, and do this as follows:

  For $\alpha = 1, 2, 3, \ldots$

  - Compute the number n_1 of paths starting in an interval in the first α bundles of $\mathcal{S}$. (Thus, $n_1 = p_1 + \cdots + p_{j_1}$ for some j_1.)
  - Compute the number n_2 of paths starting in an interval in the α bundles before B^* in $\mathcal{S}$. (Thus, $n_2 = p_{j_2} + \cdots + p_k$ for some j_2.)
  - Compute $n_3 = p_{legal} - n_2$, thus $n_3 = p_1 + \cdots + p_{j_2 - 1} + p_{rest}$.
  - Stop as soon as $n_1 \geq \frac{1}{2} p_{legal}$ or $n_3 < \frac{1}{2} p_{legal}$. The last bundle that has been added is the bundle containing $I_{\ell^-}, \ldots, I_{\ell^+}$.

  Note that we can compute the value of n_1, n_2, n_3 by adding to the values of the previous round. Since we search for ℓ in parallel from both ends, starting at the bundles containing I_1 and I_k, this search takes at most $O(1 + \min\{\ell^-, k - \ell^+\})$ time.

- Compute $p_1 + \cdots + p_{\ell^-}$ and $p_{\ell^+ + 1} + \cdots + p_k$, determine whether case (C1) or (C2) applies, and find the appropriate query. These values can be computed in $O(1)$ from n_1 or n_3 computed in the previous step, by adding/subtracting the number of paths in the bundle containing $I_{\ell^-}, \ldots, I_{\ell^+}$.

Once we have done the query, the data structures must be updated. The crucial observation for doing so is that p_j (the number of paths starting at interval I_j) does not change, since we always delete clique intervals. Also, f_i and I_k are updated dynamically during the algorithm. All that remains to do is to eliminate an interval I_j. To do so, we first decrease p by p_j. Then we remove all references to I_j in the bundles that contain it. If the bundle is now empty we delete it as well. This takes constant time per deleted interval.

Finding the next query to perform thus takes constant time per query, with two exceptions: advancing I_k takes time proportional to the number of steps that are advanced, and

[4]Note that occasionally we will apply case (C3) even if case (C1) or (C2) was possible; this is necessary because we cannot test whether (C1) or (C2) applies in constant time.

finding the bundle containing I_ℓ takes time proportional to the number of bundles that had to be searched. However, both these operations take constant amortized time. To see this, note that once an interval is a clique interval, it stays a clique interval until it is eliminated, because being a clique interval only depends on the location of the first end point f_i, and f_i increases throughout the algorithm. Hence, B^* advanced only once per bundle, or $O(n)$ time total.

As for the time to find the bundle containing ℓ, this is proportional to the minimum of ℓ or $k - \ell$. However, if we do this search, then we end in case (C1) or (C2) and eliminate at least $\min\{\ell, k - \ell\} - 1$ intervals. Thus, the time spent on finding ℓ is proportional to the number of eliminated intervals, hence the overall time for this is also $O(n)$.

We conclude:

Theorem 4. *Given an n-vertex interval graph G with m edges in its complement, we can find the hidden independent set in G using q queries, where q is asymptotically optimal if only a constant number of intervals start in any one point. The overall computation time and space is $O(n + m)$.*

4 Conclusions and Future Work

In this paper we studied a problem motivated by applications in bioinformatics and game playing: given an interval graph, how can we find an independent set chosen by an adversary with as few queries as possible? We gave polynomial-time algorithms both for verifying whether some independent set is the one chosen by the adversary, and for discovering what set the adversary has chosen. The algorithm for verification gives the optimal number of queries for all instances. The algorithm for independent set discovery gives a number of queries that is optimal to within constant factor, provided that no more than a constant number of intervals start at the same point. This algorithm is optimal in the adaptive sense as well as in the worst case sense. We also proved a stronger lower bound than the one implied by a simple information theory argument. Several related questions deserve further study:

- The main open problem is whether our adaptive algorithm can extend to instances in which many intervals may start at a common point, and still achieve a number of queries that is within a constant factor of optimal. The staircase example (Figure 3) requires $\Omega(\sqrt{p})$, showing that the information-theoretic lower bound of $\Theta(\log p)$ becomes unachievable in this setting. Is this the worst example, i.e., can all instances be solved using $O(\sqrt{p})$ queries? A positive answer to this question would not complete the adaptive algorithm, which must be within a constant factor of optimal for every instance.

- One of the problems that motivated this work is gene finding using PCR techniques. Here we need to consider that obtaining probing material is often done via an external provider, and the turnaround time between each request might dominate the total time. We might thus consider performing several probes in parallel rounds. What is

the minimum number of queries required if the entire computation must be done in a given number of rounds?[5]

- In the application to gene finding, we might also be able to eliminate certain edges of $\overline{G}$ using biological background information. Can we adapt our algorithm to take advantage of this, i.e., use an optimal number of queries subject to knowing this information? (Note that G is now no longer necessarily an interval graph.)

- The conventional 2-dimensional Battleship game is a natural candidate for further study. Can the algorithms be extended to boxicity-2 graphs? What about intersection graphs of other shapes, such as ships on a diagonal or the tetromino shapes of Tetris fame? Also, the number of ships and their shapes are known a priori in the board game, and not every independent set can be a placement of ships. Can this information be used to our advantage? Finally, in some variants, "yes" queries are rewarded by being allowed to fire again. What are good strategies in this scenario?

- From both applications, and out of general interest, the problem on arbitrary graphs also deserves study. More precisely, assume that we are given a graph $G = (V, E)$. The queries are of the form "Given a clique K in G, is $X \cap K \neq \emptyset$?" Under what type of conditions can we successfully identify an independent set using clique queries? Can we generalize the queries to subsets of vertices other than cliques?

Acknowledgements. We thank Dan Brown and the participants at the Bioinformatics problem sessions at the University of Waterloo for many useful comments on this problem. All authors were partially supported by NSERC.

References

[1] Battleships variations. Mountain Vista Software. Web page. See `http://www.mountainvistasoft.com/variations.htm`.

[2] R. Beigel, N. Alon, M. S. Apydin, and L. Fortnow. An optimal procedure for gap closing in whole genome shotgun sequencing. In *5th Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 22–30, 2001.

[3] H. L. Bodlaender and D. Kratsch. The complexity of coloring games on perfect graphs. *Theoretical Computer Science*, 106(2):309–326, 1992.

[4] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268(1):78–94, 1997.

[5] This is similar to network sorting of a set of numbers. In this problem, given an integer n, the goal is to produce a predetermined sequence of comparison/exchanges, called a sorting network, such that the sequence of comparison/exchanges sorts any given set of n numbers. The quality of the sorting network is measured by both the number of comparators (probes) and the depth (rounds) of the network of comparators (see e.g. [15]).

[5] L. S. Chandran. A high girth graph construction and a lower bound for hitting set size for combinatorial rectangles. In *19th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 283–290, 1999.

[6] M. Das, C. B. Burge, E. Park, J. Colinas, and J. Pelletier. Assessment of the total number of human transcription units. *Genomics*, 77(1-2):71–78, 2001.

[7] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.

[8] I. Dunham, N. Shimizu, B. A. Roe, S. Chissoe, et al. The DNA sequence of human chromosome 22. *Nature*, 402(6761):489–495, 1999.

[9] P. Erdős and J. L. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory – Series A*, 14:298–301, 1973.

[10] S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4:507–518, 1975.

[11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *20th ACM Symposium on Principle of Database Systems (PODS)*, pages 102–113, 2001.

[12] M. C. Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, New York, 1980.

[13] A. V. Karzanov. On finding maximum flows with special structure and some applications (in Russian). In *Matematicheskie Voprosy Upravleniya Proizvodstvom*, volume 5. Moscow State University Press, 1973.

[14] L. M. Kirousis and C. H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2):205–218, 1986.

[15] D. E. Knuth. *The art of computer programming. Volume 3, Sorting and searching.* Addison-Wesley Publishing Co., 1973.

[16] J. Kratochvíl and J. Matoušek. Intersection graphs of segments. *Journal of Combinatorial Theory. Series B*, 62(2):289–315, 1994.

[17] N. Linial, M. Luby, M. Saks, and D. Zuckerman. Efficient construction of a small hitting set for combinatorial rectangles in high dimension. *Combinatorica*, 17(2):215–234, 1997.

[18] R. M. McConnell and J. P. Spinrad. Construction of probe interval models. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 866–875, 2002.

[19] N. Pavy, S. Rombauts, P. Dehais, C. Mathe, D. V. Ramana, P. Leroy, and P. Rouze. Evaluation of gene prediction software using a genomic data set: application to *Arabidopsis thaliana* sequences. *Bioinformatics*, 15(11):887–889, 1999.

[20] P. A. Pevzner. *Computational molecular biology: an algorithmic approach*. MIT Press, 2000.

[21] F. S. Roberts. On the boxicity and cubicity of a graph. In *Recent Progress in Combinatorics (3rd Waterloo Conference on Combinatorics, 1968)*, pages 301–310. Academic Press, New York, 1969.

[22] F. S. Roberts. *Discrete mathematical models with application to social, biological and ecological problems*. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[23] A. A. Salamov and V. V. Solovyev. Ab initio gene finding in *Drosophila* genomic DNA. *Genome Research*, 10(4):516–522, 2000.

[24] S. J. Scharf, G. T. Horn, and H. A. Erlich. Direct cloning and sequence analysis of enzymatically amplified genomic sequences. *Science*, 233(4768):1076–1078, 1986.

[25] M. Schena, D. Shalon, R. W. Davis, and P. O. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270(5235):467–470, 1995.

[26] D. D. Shoemaker, E. E. Schadt, et al. Experimental annotation of the human genome using microarray technology. *Nature*, 409(6822):922–927, 2001.

[27] G. Xu, S. H. Sze, C. P. Liu, P. A. Pevzner, and N. Arnheim. Gene hunting without sequencing genomic clones: finding exon boundaries in cDNAs. *Genomics*, 47(2):171–179, 1998.