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

Technical report CS-2001-26
Dept. of Computer Science, University of Waterloo
December 2001

Abstract

We design efficient competitive algorithms for discovering information hidden by an adversary. Specifically, consider a game in a given interval graph G in which the adversary chooses an independent set X in G. Our goal is to discover this hidden 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 on the interval graph, these algorithms use an asymptotically optimal number of queries in every instance.

1 Introduction

An independent set (or *stable set*) in a graph G is a set of vertices such that no two vertices share an edge. The problem of finding an independent set of maximum size in a graph is a well-known optimization problem with many applications; for example in game theory and communication networks. It was among the first problems to be shown to be NP-complete [Kar72], although it is solvable for some special graph classes.

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. [Rob76]). 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. [Gol80]).

In this paper we study how to determine, given an interval graph, an unknown independent set X chosen by an adversary. We refer to X as a *hidden independent set*. Note that X need not be maximal. We determine X by playing an interactive game against an

adversary using queries of the following type: "Is a point p on the real line covered by an interval in X?" The adversary 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 (see Section 1.3) and the game of Battleship (see Section 1.4).

While there is a wide literature regarding games in graphs (e.g. game coloring [BK92], the Ramsey graph game [ES73], and node search [KP86]), 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. [XSL+98] 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?". Beigel et al. [BAAF01] 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?" 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. McConnell and Spinrad [MS02] consider the tangentially related problem of reconstructing an interval graph given probes about the neighbours of only a partial set of vertices.

We now give some terminology before stating the precise problems and our results.

1.1 Terminology

As defined above, an *interval graph* is an intersection graph of intervals on the real line. Without loss of generality 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. It is well-known that every interval graph can be represented in such a way. We denote the interval of the ith vertex of an interval graph 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, depending on this, 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. [Gol80].

Formally, an *independent set* of vertices of a graph $G = (V, E)$ is a subset V' of V such that no two vertices in V' are adjacent. We will deal with discovering an initially unknown independent set 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 $\overline{G}$. If G is an interval graph, then $\overline{G}$ has a "natural" acyclic orientation (direct each edge from the left interval to the right interval) and any clique in $\overline{G}$ hence has a unique topological order. We can therefore consider V' as a (directed) path π in $\overline{G}$, and will sometimes speak of a *hidden path* instead of a hidden independent set. We will omit the word "directed" from now on 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 simply a unit open interval $(a, a+1)$, where a is an 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."[1]

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 hand, intervals are allowed to have the same start point or the same end point.[2]

1.2 Our results

Suppose we are given an interval graph 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 an adaptive algorithm for discovering X. If no two intervals at a common point, then this protocol is within a constant factor of the optimal number of queries for that specific instance. That is, the algorithm is optimal in an adaptive sense (see [DLOM00, EW92, Meh84] for details on adaptive algorithms). 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 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 scientists to obtain genomic sequences of several organisms. These sequences need to be annotated, i.e., biological meaning needs to be assigned to particular regions of the sequence. An important step in the annotation process is the identification of genes. For our purposes a gene is a sequence of disjoint regions of the genomic sequence. These regions are called exons.

The discovery of genes in DNA sequences is a well-studied problem. There are a number of automatic tools for gene prediction; however, experimental studies (e.g. [PRD+99]) show that even the best of them predicts, on average, only about 50% of the entire genes correctly.

[1] Note that since intervals begin and end at integers, probing with a unit interval is equivalent to probing at a non-integral point.

[2] 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.

It is therefore important to have alternative methods that can produce or verify such predictions by using experimental data. Our approach is based on polymerase chain reaction (PCR) technology and it was inspired by open problem 12.94 in [Pev00]. Without going into further details, we note that PCR technology can be used to perform the following query: given two short sequences called primers, do both of them occur in some exon of the gene?

Assume for now that we are given a set which contains all real exons belonging to a gene as well as some false exons. Each exon is an interval of the DNA sequence; therefore, the set of candidate exons is a set of intervals in a corresponding interval graph. The gene is a collection of disjoint intervals; therefore, it corresponds to the hidden independent set in the interval graph of all exon candidates. We can try either to discover this hidden independent set or to check whether the prediction of a gene finding program (another independent set) is correct.

Our setup assumes that the candidate exons are given; they can be determined using computer tools for gene prediction. These algorithms have to balance sensitivity (i.e., how many real exons they discover) with specificity (i.e., how many false exons they predict), and usually it is possible to increase sensitivity at the expense of a decrease in specificity. To create a good set of candidates for our set–up, we should use a highly sensitive method that may generate many false exons but has only a small probability of excluding a real exon. The queries in our game can be implemented as PCR experiments. Note that many real-life constraints would need to be addressed to apply this technique in practice.

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. [MVS]) involving other ship shapes or higher dimensions. In theoretical computer science, a *sampling* variation of the problem has been considered in which the collection of shots must cover the d-dimensional lattice in order to hit all rectangles with at least a given volume [Cha99, LLSZ97].

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, 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. [Rob69]). In fact, it is an even more specialized graph since all rectangles are forced to have one unit dimension.

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 of length k 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

The verification variant of the problem can be formulated as follows: given an interval graph G and an independent set Y in G, 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: 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 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 query; 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 $Y = X$. 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

Let us consider the case in which only positive probes, i.e., queries with expected answer "yes", are allowed. Note that for some inputs it is impossible to verify $Y = X$ using only positive queries.

Sometimes we will consider only intervals inside some region $[a, b]$. To simplify the discussion in such cases we use the following notation. For any graph W, let $W[a, b]$ denote the subgraph of W 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 region $[a, b]$.

The minimum set of positive probes for a graph $G[a_{min}, a_{max}]$ will be computed using a directed acyclic graph H defined as follows. Graph H contains one vertex for every

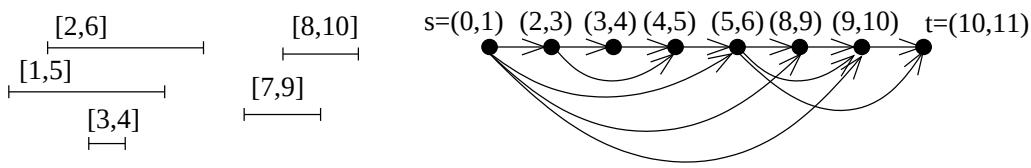

Figure 1: Example of interval graph G and its corresponding graph H. for $Y = \{[2,6],[8,10]\}$.

positive probe. 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 $G[a_{min}, a_{max}]$.

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 $Z_{a,b}$ in $G[a+1, b]$ that intersects all positive probe $(c, c+1)$ with $a < c < b$ and that is different from Y. See Figure 1 for an example of graph H.

Graph H has $O(n)$ vertices and $O(n^2)$ edges, where n is the number of intervals. We will give an algorithm for constructing H in 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 $G[a_{min}, a_{max}] = G$ that intersects all positive probes and that is different from Y. But this means that $Z_{s,t}$ and Y can not 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 to be the union of the sets $Z_{a,b}$ over all edges $e_{a,b} \in \pi$. Notice that Z_π is an independent set in G, because for any edge $e_{a,b}$ in π, the independent set $Z_{a,b}$ has intervals with points between $a+1$ and b.

We will show that $Z_\pi \neq Y$. Graph H does not contain edge $e_{s,t}$; otherwise $X = Y$ could not be verified by Lemma 1. Thus path 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 also cannot intersect the positive probe $(u, u+1)$, which proves $Z_\pi \neq Y$.

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 $(v, v+1) \in \pi$.

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 all i such that $0 \leq i \leq k$, the 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 therefore independent set Z is inconsistent with probe $(u, u+1)$. □

The best general algorithm we know to solve this form of the vertex connectivity problem is based on network flows:

Lemma 3. *There is an $O(n^{8/3})$-time algorithm that, given a graph H and vertices s and t, finds the smallest set of vertices whose removal from H disconnects s from t.*

Proof. 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 [Kar73, ET75]. □

2.2 Finding the 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. The special case studied in Section 2.1, in which we do not allow any negative queries, turns out to be a base case in the dynamic program.

Lemma 4. *Let $A[a, b]$ be the smallest number of queries needed to verify that $X[a, b] = Y[a, b]$ in the interval graph $G[a, b]$. 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. Then*

$$A[1, a] = \min \begin{cases} A_+[1, a], & \\ \min_b A[1, b-1] + A_+[b+1, a] + 1, & \textit{where } (b, b+1) \textit{ is a negative} \\ & \textit{probe intersecting } [1, a] \end{cases}$$

Proof. Consider the optimal solution of subproblem $A[1, a]$. If this solution contains only positive queries, then $A[1, a] = A_+[1, a]$. If the solution contains a negative probe, then let b be the rightmost negative probe. 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 $A[1, b-1]$. Therefore in this case we have $A[1, a] = A[1, b-1] + A_+[b+1, a] + 1$. □

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 2. However, we still need to provide the method for finding the edges of the graph 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 $G[a,b]$.

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

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

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 $(a-1,a)$. The values $E_{a,b}$ can then be computed using the following recursive formula.

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

The base case happens if $a=b$ or $a+1=b$. In this case the only independent set satisfying the criteria is the empty independent set. 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 probes are valid, but we also need to add independent sets that do not intersect $(b-1,b)$. These are stored in $E_{a,b-1}$.

Each interval $[c,b]\in G$ is checked $O(n)$ times, once for each $a<c$. All other computation is $O(1)$ time per one $E_{a,b}$. Therefore all values of $E_{a,b}$ can be computed in $O(n^2)$ time. □

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 $O(n^4 + n^2T)$ 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.*

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 asymptotially optimal number of queries if no two intervals start at a common point.

3.1 Lower bounds

We have the following simple information-theoretic lower bound.

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. □

Note that this theorem holds for any graph, and any type of queries. However, we do not always get a tight bound, not even for an interval graph. As an example, consider the so-called *staircase*, which 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 $i = n+1, \ldots, 2n$. See Figure 2.

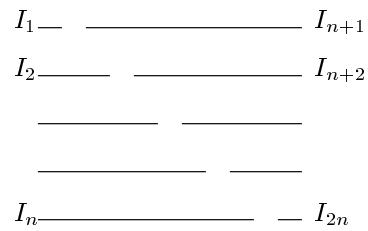

Figure 2: The staircase needs $n-1$ queries.

In this case we have $n(n+1)/2 + 2n + 1$ independent sets, which gives lower bound of $2\log_2 n + O(1)$ queries, but in fact any algorithm requires at least $n-1$ queries.

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 2.

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. □

3.2 An adaptive protocol

Now we show that the lower bound from Theorem 2 can be matched (asymptotically) under some assumptions. Thus, we will give an adaptive 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 search for a hidden path. Denote by P the set of directed paths in the complement $\overline{G}$ of the interval graph; as discussed before $p = |P|$.

3.2.1 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.

For ease of notation, 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 first 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 name suggests, they form a clique in G, and at most one of them is in any path.

For our recursive algorithm to work properly, we must allow additional restrictions, and thus operate under two different scenarios. Let a *legal path* be a path in the graph that could be the solution even under the following added restrictions. In the first scenario, any path is a legal path. We will call this the *unrestricted scenario*. This is the scenario at the beginning of the algorithm. In the second scenario, only a path that intersects (f_{i-1}, f_i) is legal. (We will have obtained this information through previous queries.) We will call this the *restricted scenario*. Therefore any legal path uses a clique interval. Furthermore, no legal path can use a clique interval that starts at f_i, and such intervals can be eliminated. Thus, in the restricted scenario all clique intervals start strictly before f_i.

3.2.2 Effects of a query

The algorithm uses only one kind of query: we always query at $(a, a+1)$ for some $a \leq f_i$. Now we will study the effect of such a query. Note first that since $a \leq f_i$, only clique intervals can intersect the probe (though not all of them necessarily do). Through our choice of a, we will ensure that at least one clique interval always intersects the probe.

After each query we reduce the graph by eliminating several intervals so that the legal paths of the reduced graph are exactly the legal paths of the original graph that were consistent with the answer to the query. The intervals to be eliminated and the resulting scenario can be determined as follows. If the answer to a query at $(a, a+1)$ is "no", then clearly none of the clique intervals that intersect $(a, a+1)$ are in the hidden path and we eliminate them from the input graph. If the original scenario was unrestricted, by this elimination we obtain a graph which contains exactly all paths that are consistent with this query and we can solve the problem recursively by solving the unrestricted scenario on the reduced graph.

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. Some of the clique intervals were eliminated by the query, which 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, since they intersect the original clique interval that is in X. Thus, if we are in the restricted scenario, then intervals that would become clique intervals due to an increase in f_i can be eliminated as well. Hence the new clique intervals are exactly those original clique intervals that do not intersect $(a, a+1)$. One of them is in the hidden path, and they all intersect $(f_i - 1, f_i)$ for the new value of f_i, so again we want to solve the restricted scenario 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. Therefore, regardless of the current scenario, the next scenario will be restricted. We also can eliminate all intervals that become clique intervals (due to an increase in f_i), because they do not intersect $(a, a+1)$ by $a \leq f_i$ (for the old value of 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.

3.2.3 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 $P_{legal} = P_1 \cup \ldots \cup P_k \cup P_{rest}$, where P_j is the set of legal paths that use 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 by $p_{rest} = 0$. 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} ends not after I_{j_2}. □

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

Lemma 7. *If we query at $(s_j, s_j + 1)$ for some j with $s_j < f_i$, then we can eliminate either $p_1 + \ldots + p_{j'}$ paths or $p_{j'+1} + \ldots + p_k + p_{rest}$ 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 $p_1 + \ldots + p_{j'}$ 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 is necessarily a clique interval and all paths in P_{rest} can be eliminated. Furthermore, the clique intervals $I_{j'+1}, \ldots, I_k$ do not intersect $(s_j, s_j + 1)$ (by choice of j'), so these intervals can be eliminated as well. □

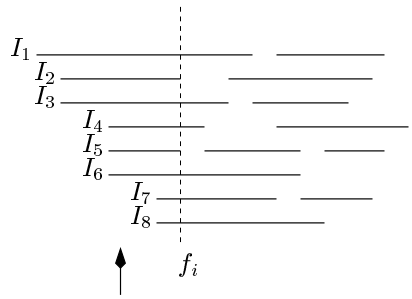

Figure 3: A query at $(s_5, s_5 + 1)$ eliminates $p_1 + \ldots + p_6$ paths (if the answer is "no") and $p_7 + p_8 + p_{rest}$ paths (if the answer is "yes").

3.2.4 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

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

this is well-defined because $p_1 + \ldots + p_k \geq \frac{1}{2} p_{legal}$ 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 cases:

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

In this case, query at the beginning of interval $I_{\ell^- - 1}$, 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: $p_1 + \ldots + p_{\ell^+} \geq \frac{1}{4} p_{legal}$ and $p_{\ell^+ + 1} + \ldots + p_k + p_{rest} \geq \frac{1}{4} p_{legal}$:

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

C3: All remaining cases.

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 case (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. *Assume that cases (C1) and (C2) do not hold. Then* $p_{\ell^-} + \ldots + p_{\ell^+} > \frac{1}{2} p_{legal}$.

Proof. By definition of ℓ, we have $p_\ell + \ldots + p_k + p_{rest} = p_{legal} - (p_1 + \ldots + p_{\ell-1}) > \frac{1}{2} p_{legal}$, and by $\ell^- \leq \ell$ 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 $p_1 + \ldots + p_\ell \geq \frac{1}{2} p_{legal}$, and by $\ell \leq \ell^+$ therefore $p_1 + \ldots + p_{\ell^+} \geq \frac{1}{2} p_{legal}$. So if (C2) does not hold, then

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

There are thus 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 $p_{\ell^-} + \ldots + p_{\ell^+} > \frac{1}{2} p_{legal}$. The intervals $I_{\ell^-}, \ldots, I_{\ell^+}$ all start at s_ℓ, therefore there are at most θ of them, and

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

as desired. □

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. We need an observation:

Lemma 10. *If we are in case (C3), then 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. 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), contradicting $p_i > \frac{1}{2} p_{legal}$. 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 3.2.2), we would not have tried to find 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 query with a negative answer in case (C3), we have a different value of f_i. Let $f_{i_1} < \ldots < f_{i_s}$ 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 intersect, then they either have the same end point (which contradicts the choice of the f_{i_j}'s), or 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 state the main result.

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* $\log_2 p + \max\{\log_{\theta/(\theta-\frac{1}{2})} p, \log_{4/3} p\}$ *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 $m - s \leq \log_{\theta/(\theta-\frac{1}{2})} p$ (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.

3.2.5 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.

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. The bundles in $\mathcal{S}$ are sorted by the start point of the intervals in it. Recall that all start points are integers between 1 and $2n$; we can therefore initialize $\mathcal{S}$ with a bucket sort in $O(n)$ time.

- Similarly we store a list $\mathcal{E}$ of bundles of intervals with the same end points, sorted by end points.

- 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, i.e., the number of paths that start at I_j. 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. The p is simply the sum of all p_j; p_{rest} is initialized to p and will be updated later. We do not change p_{rest} when we move to a different scenario, but depending on the current scenario we use $p_{legal} = p$ or $p_{legal} = p - p_{rest}$.

- 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 list 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 3.2.2. We will study below what needs to be done to eliminate an interval.

  The time required to do this is proportional to the number of bundles that we have advanced.

- 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).[3]

[3]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.

- 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, $n_1 = p_1 + \ldots + p_{j_1}$ 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, $n_2 = p_{j_2} + \ldots + p_k$ for some j_2.)
  - Compute $n_3 = p_{legal} - n_2$, thus $n_3 = p_1 + \ldots + p_{j_2-1} + p_{rest}$.
  - 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 $p_1 + \ldots + p_{\ell^-}$ and $p_{\ell^+ + 1} + \ldots + p_k$, 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 finding the bundle containing I_ℓ takes time proportional to the number of bundles that had to be searched. However, both these operations are 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 at most 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 work 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 adaptive 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 of an independent set 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. Both algorithms are 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 questions which remain unanswered deserve further study:

- The staircase example (Figure 2) shows that the information-theoretic lower bound is not always tight. In this case $\Omega(\sqrt{p})$ queries are necessary, where p is the number of independent sets in the graph. Is this the worst possible or is there perhaps a graph requiring an even higher number of queries? Alternatively, can we prove a matching upper bound; that is, an algorithm that finds any hidden independent set with at most $\Omega(\sqrt{p})$ queries?

- One of the problems that motivated this work is gene finding using PCR techniques. In this setting we need to consider not only the number of probes but also the fact that obtaining primers (probing material which corresponds in our setting to unit intervals) is often done via an external provider. Thus, the turnaround time between each primer request might dominate the entire computation. To speed up this process we might consider performing several probes in parallel rounds. In each round we decide upon a set of queries and obtain answers for them performing as many as possible at the same time. What is the minimum number of queries required if the entire computation is done in a given number of rounds?[4]

- Alternatively, using biological background information, we might be able to eliminate certain edges of the directed complement G' a priori and thus reduce the number of probes needed. (Note that G is now no longer necessarily an interval graph.)

- Another motivation for our work is the game *Battleship*. In this paper we considered a 1-dimensional variant of Battleships. The conventional 2-dimensional game seems a natural candidate for further study. Can the algorithms be extended to an intersection graph of rectangles of height 1 or width 1? What about other shapes, such as ships on a diagonal or the tetromino shapes of Tetris fame?

[4] 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 [Knu73]).

In the board version, the number of ships and their shapes are known a priori. Hence, not every independent set can be a possible hidden independent set. 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 scenerio?

- 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 generalized the queries to subsets of vertices other than cliques?

Acknowledgements. We thank 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

[BAAF01] Richard Beigel, Noga Alon, Mehmet S. Apydin, and Lance Fortnow. An optimal procedure for gap closing in whole genome shotgun sequencing. In Thomas Langauer, David Sankoff, Sorin Istrail, Pavel Pevzner, and Michael Waterman, editors, *Proceedings of the 5th Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 22–30, Montreal, Canada, April 22–25 2001. ACM Press.

[BK92] Hans L. Bodlaender and Dieter Kratsch. The complexity of coloring games on perfect graphs. *Theoretical Computer Science*, 106(2):309–326, 1992.

[Cha99] L. Sunil Chandran. A high girth graph construction and a lower bound for hitting set size for combinatorial rectangles. In *Proceedings of the 19th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 283–290, Chennai, India, December 13–15 1999.

[DLOM00] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 743–752, 2000.

[ES73] P. Erdős and J. L. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory – Series A*, 14:298–301, 1973.

[ET75] S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4:507–518, 1975.

[EW92] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, December 1992.

[Gol80] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, New York, 1980.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[Kar73] A. V. Karzanov. O nakhozhdenii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh (On finding maximum flows in networks with special structure and some applications). In *Matematicheskie Voprosy Upravleniya Proizvodstvom*, volume 5. Moscow State University Press, 1973.

[Knu73] Donald E. Knuth. *The art of computer programming. Volume 3.* Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont., 1973. Sorting and searching, Addison-Wesley Series in Computer Science and Information Processing.

[KP86] Lefteris M. Kirousis and Christos H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2):205–218, 1986.

[LLSZ97] Nathan Linial, Michael Luby, Michael Saks, and David Zuckerman. Efficient construction of a small hitting set for combinatorial rectangles in high dimension. *Combinatorica*, 17(2):215–234, 1997.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching.* Springer-Verlag, 1984.

[MS02] 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.

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

[Pev00] Pavel A. Pevzner. *Computational molecular biology: an algorithmic approach.* The MIT Press, 2000.

[PRD+99] 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.

[Rob69] Fred S. Roberts. On the boxicity and cubicity of a graph. In *Recent Progress in Combinatorics (Proc. Third Waterloo Conf. on Combinatorics, 1968)*, pages 301–310. Academic Press, New York, 1969.

[Rob76] Fred S. Roberts. *Discrete mathematical models with application to social, biological and ecological problems.* Prentice-Hall, Englewood Cliffs, NJ, 1976.

[XSL+98] 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.