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

Consider a game in a given set of intervals (and their implied interval graph G) in which the adversary chooses an independent set X in G. The 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, these algorithms use an asymptotically optimal number of queries in every instance.

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. An *independent set* in 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 chosen by an adversary. 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. The goal is to use the smallest possible number of queries to determine set X. This 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., [3, 9, 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. [25] 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. McConnell and Spinrad [16] consider the tangentially related problem of reconstructing an interval graph given probes about the neighbors of only a partial set of vertices.

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 refer to the initially unknown independent set in G chosen by an adversary, and 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". We 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.

Our Results We study two versions of the problem. First, the verification problem consisting of verifing. via probe queries, that a purported independent set Y is the hidden independent set. Second, the discovery problem, consisting of indentifying the set X.

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. 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.

Applications to Gene Finding Recent advances in molecular biology have resulted in 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, 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, 21]); however, experimental studies (e.g., [17, 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 by using experimental data.

While genes cannot be reliably predicted by purely computational means, we can use these methods to provide us with a set of candidate exons. Algorithms for gene prediction have to

[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).

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. By using a highly sensitive method, we may generate a candidate set that contains many false exons but has only a very 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 given short region of DNA sequence contained in a real exon?" In order 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 [23]. Shoemaker et al. [24] 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 the independent set verification (Section 2), we can design a smaller set of queries which 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 [22]. The PCR can answer the 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)?" Answer to our query can be obtained by PCR provided that we already know at least one short region of 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 [18]. PCR queries were also used in similar way to determine the exon boundaries in cDNA clones [25].

Applications to 1-dimensional Battleship The game of Battleship (also known as Convoy and Sinking-Ships) 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. The Battleship becomes an interval graph game in the 1-dimensional version. 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

Let Y be the candidate set to be verified. 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."

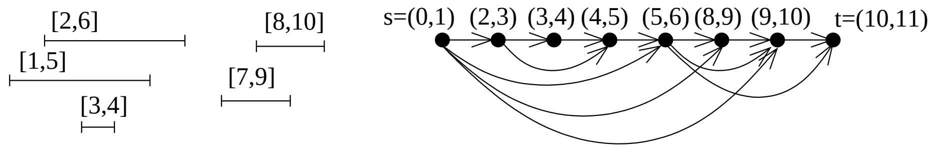

Figure 1: An interval graph and its corresponding graph H for $Y = \{[2,6],[8,10]\}$. For example, edge $(5,6) \to (9,10)$ exists because the independent set $\{[7,9]\} \in G[6,9]$ intersects all positive probes between 6 and 9.

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 implies that we can rephrase the verification problem as follows: given a 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. 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 this query *eliminates* Z.

Finding a Minimum Set of Positive Probes We first 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. Note that for some inputs it is impossible to verify $Y = X$ using only positive queries.

Let $G[a,b]$ denote the subgraph of G induced by intervals completely contained in the region $[a,b]$ and for any independent set Z, let $Z[a,b]$ denote the subset of Z of intervals completely contained in $[a,b]$. The minimum set of positive probes for a graph G will be computed using a directed acyclic graph H. 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. I.e. 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 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. Using dynamic programming, it can be constructed in $O(n^2)$ time. 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 that intersects all positive probes and that is different from Y. But then $Z_{s,t}$ 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. Details are omitted due to space. 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 $X[a, b] = Y[a, b]$.

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

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.

Lemma 4. *Let $A[a]$ be the smallest number of queries needed to verify that $X[1, a] = Y[1, a]$ in the interval graph $G[1, a]$. Then*

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

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 $A[a] = A[b] + A_+[b+1, a] + 1$. □

Thus we can find the optimum solution with dynamic programming. We have at most n^2 subproblems $A_1[a, b]$ to solve, and hence obtain the overall result.

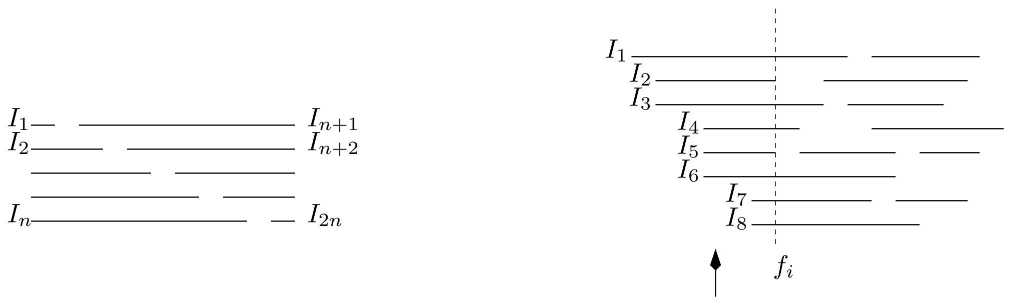

Figure 2: (a) Staircase (b) Paths eliminated: $p_1 + \ldots + p_6$ ("no" answer) or $p_7 + p_8 + p_{rest}$ ("yes").

Theorem 1. *Given an n-vertx interval grpah 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 give an interactive protocol to find an independent set X. In this case 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. A simple information-theoretic argument yields the following 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.*

We do not always get a tight bound, even for an interval graph. Consider the so-called *staircase* depicted on Figure 2. 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. Each query in the worst case eliminates at most one pair $\{I_i, I_{n+i}\}$, and with anything less than $n-1$ queries, we cannot determine the hidden independent set in the worst case.

The lower bound of $\lceil \log_2 p \rceil$ queries from Theorem 2 can be matched (asymptotically) under the assumption that at most a constant number of intervals start at the same point.

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. This would be straightforward if the set of paths always had a central element allowing us to readily eliminate the desired constant fraction of paths. However there are configurations with no such element. In this case the crucial observation is that, surprisingly, such configurations appear unfrequently. 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 with lefmost right end point. 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*. Note that as the name 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 added 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.

Effects of Queries The algorithm always queries at $(a, a+1)$ for some $a \leq f_i$. Only clique intervals can intersect the probe. If the answer to the query 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. 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. Eliminating 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 the query 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, 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.

Some Definitions and Observations Consider a specific point in 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. (Sketch) For every path π in P_i, we can obtain a path π' by deleting the first interval (which is I_i) in π. Conversely, for every path π in P_{rest} we can obtain a path π' in P_i by adding I_i to π. □

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

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

Lemma 5. *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 intersecting (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, the clique intervals $I_{j'+1}, \ldots, I_k$ do not intersect $(s_j, s_j + 1)$ and can be eliminated as well. □

Choosing Queries In light of Lemma 5 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 $p_{\ell^-} + \cdots + p_k + p_{rest} \geq \frac{1}{4}p_{legal}$: The algorithm queries 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 5 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 5 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 5, 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).

Lemma 6. *If cases (C1) and (C2) do not hold, then* $p_{\ell^-} + \cdots + p_{\ell^+} > \frac{1}{2}p_{legal}$.

Proof. Note that $\ell^- \leq \ell \leq \ell^+$, and hence by Equation 1 we have $p_{\ell^-} + \ldots + p_k + p_{rest} > \frac{1}{2}p_{legal}$ and $p_1 + \ldots + p_\ell \geq \frac{1}{2}p_{legal}$. The result now follows from the definition of cases (C1) and (C2). □

Lemma 7. *Let θ denote the maximum number of intervals that have a common start point (i.e., $l^+ - l^- + 1 \leq \theta$). 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 6 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.

Lemma 8. *In case (C3) at least one clique interval intersects* (f_i, f_i+1).

Proof. (Sketch) If this is not true, then all clique intervals end at f_i. This implies $\ell^- = \ell = \ell^+$, so $p_\ell > \frac{1}{2}p_{legal}$ by Lemma 6. Using the claim, one can show that there is only one clique interval and we must be in the restricted scenario. This clique interval must be in the hidden independent set, and we would not have queried. □

Lemma 9. *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 that we never repeat a query with a negative answer at (f_i, f_i+1) as it eliminates all intervals that intersect the probe. Hence by Lemma 8, 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 $1 \le j \le s$ 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. $\square$

Lemma 10. *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 $s \le \log_2 p$. 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 $m - s \le \log_{4/3} p$ (for $\theta \le 2$) or $m - s \le \log_{2\theta/(2\theta-1)} p$ (for $\theta > 2$). $\square$

Note that as long as θ is a constant, we use $O(\log_2 p)$ queries, which is asymptotically optimal. This can easily be implemented in polynomial time and indeed, with the right data structure can be accomplished in time $O(n + m)$, where m is the number of edges in the complement of the interval graph. Details are omitted due to space limitations.

Theorem 3. *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 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 question 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. 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? 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.)

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. *5th Int. Conf. on Comp. Molecular Biol. (RECOMB)*, pp. 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] L. S. Chandran. A high girth graph construction and a lower bound for hitting set size for combinatorial rectangles. *19th Conf. Found. of Soft. Tech. and Theor. Comp. Sci.* , *LNCS 1738*, pp. 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 ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 743–752, 2000.

[8] I. Dunham 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 J.Comp.* 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, 1980.

[13] A. V. Karzanov. On finding maximum flows with special structure and some applications (in Russian). In *Matematicheskie Voprosy Upravleniya Proizvodstvom*, vol. 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] 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.

[16] 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.

[17] 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.

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

[19] 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, 1969.

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

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

[22] 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.

[23] 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.

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

[25] 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.