The Berlin Safe House Puzzle: Spycraft via Interval Graphs

Gennaro Cordasco[1]
Dipartimento di Informatica, Università degli Studi di Salerno, Fisciano, Italy

Luisa Gargano
Dipartimento di Informatica, Università degli Studi di Salerno, Fisciano, Italy

Adele Anna Rescigno
Dipartimento di Informatica, Università degli Studi di Salerno, Fisciano, Italy

Abstract

We propose a gamified application of the Identifying Code problem on Interval Graphs, framed as a high-stakes Cold War counter-intelligence operation. We present a polynomial-time algorithm to assign "Listening Devices" (bugs) to "Safe Houses" (intervals) so that every safe house is uniquely identifiable by its bug signature. While the problem is NP-hard on several graph classes, including chordal and bipartite graphs, the interval-graph structure allows us to compute a 2-approximate solution efficiently.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Approximation algorithms analysis

Keywords and phrases Interval Graphs, Watching-System, Approximate Algorithms

Digital Object Identifier 10.4230/LIPIcs.FUN.2026.13

Funding This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU.

1 Introduction: The Mission

Welcome, *Agent X*. The year is 196X. You are a spy handler operating in the shadowed streets of Cold War Berlin.

Your network consists of m *Safe Houses*. Due to rigorous security protocols, each safe house is active only during a specific time window. However, a vulnerability exists: Two safe houses are connected if their active times overlap, meaning that agents can move between them without being exposed. Your mission is to secure the network against leaks. To do this, you must plant *Listening Devices (Bugs)* within the safe houses. These devices are sophisticated; a bug placed in one house can be tuned to listen to any chosen subset of the overlapping (connected) houses.

The Constraint

To succeed, your deployment must satisfy a critical *Unique Identification* directive: generate a unique "bug signature" – the specific set of bugs that can hear it. If a message is intercepted, you must be able to identify exactly which safe house it came from. If two houses share the same signature, the network is compromised.

[1] Corresponding author

The Strategy

The problem challenges us to find the *minimum number of bugs* required to make the entire network identifiable.

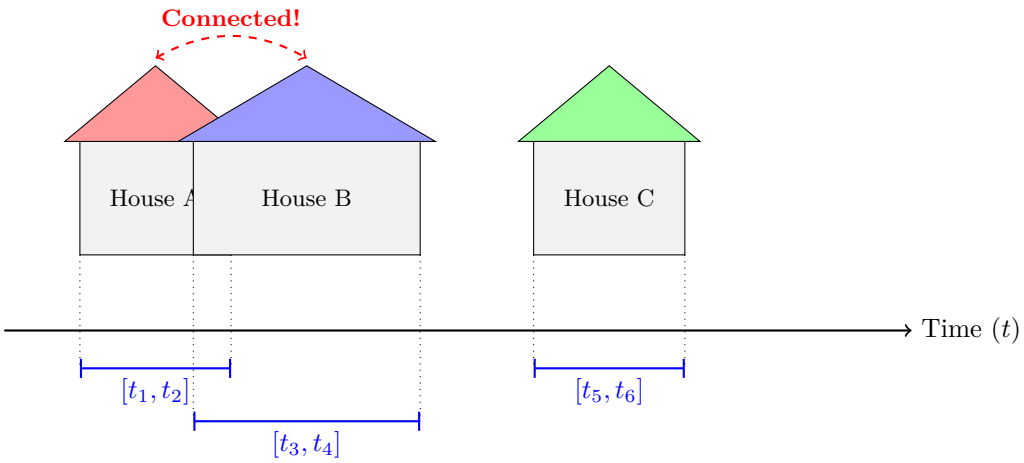

Figure 1 The Safe House Network. Safe Houses correspond to active time intervals. House A and House B overlap in time, creating a covert connection. House C is isolated.

In the following, we declassify the mathematical dossier for this operation. We map the "Safe Houses" to intervals on a real line and the "Spy Network" to an *Interval Graph*. We then present an algorithm that utilizes a "Timeline Strategy" (Interval Ordering) and a "Suspicion Shuffle" mechanism to secure the network.

2 The Network Architecture

We formalize our spy network using the language of Interval Graphs and establish the following correspondence between our problem and mathematical structures. This correspondence allows us to transfer algorithmic tools, complexity results, and structural properties from the general theory of Watching Systems to the specialized interval-graph surveillance setting developed in this work. The used definitions and the notation are given in sections 2.1-2.3.

Spycraft Term	Symbol	Mathematical Concept
Safe House	$I \in \mathcal{I}$	Interval
Connection (Overlap)	$I \cap J \neq \emptyset$	Adjacency / Intersection
Timeline Order	$\prec$	Interval Ordering
Listening Device (Bug)	b	Watcher
Bug Signature	$S(I)$	Code
Surveillance System	B	Watching System

2.1 Interval graphs

An *interval graph* is a graph in which each vertex is associated with an interval on the real line (representing the active time window of a safe house) such that two vertices are adjacent if and only if their associated intervals have a nonempty intersection (i.e., two safe houses are connected if and only if their active times overlap).

From now on, each safe house will be identified with the time interval I corresponding to its activation window. Let $\mathcal{I}$ be a set of intervals on the time line representing our network of safe houses. We now introduce a timeline ordering relation on $\mathcal{I}$. For any interval $I \in \mathcal{I}$, we denote its leftmost (minimum) and rightmost (maximum) points – corresponding to the start and end times of the safe house activity – by

$$\ell(I) = \min\{t \mid t \in I\}, \qquad r(I) = \max\{t \mid t \in I\}.$$

▶ **Definition 1** (Timeline Ordering). *For all $I, J \in \mathcal{I}$,*

$$I \prec J \iff \big(r(I) < r(J)\big) \quad or \quad \big(r(I) = r(J) \text{ and } \ell(I) > \ell(J)\big). \tag{1}$$

If $\mathcal{I} = \{I_1, \ldots, I_m\}$ is ordered so that $I_1 \prec I_2 \prec \cdots \prec I_m$, then I_i is called the i-th safe house.

2.2 Bug Signatures and Surveillance Systems

According to (1), we define an ordering on the bugs. Bugs located in different safe houses follow the order of their intervals, whereas bugs in the same house may appear in any order.

Let $B = \{b_1, b_2, \ldots, b_k\}$ be a set of listening devices on $\mathcal{I}$. For bugs $b_p, b_q \in B$ located in intervals I and J, respectively, we set

$$b_p \prec b_q \qquad \text{whenever } I \prec J. \tag{2}$$

In what follows, every set of bugs is assumed to be ordered according to (2), with ties broken arbitrarily.

Given a safe house J and a listening device (bug) b located in safe house I, we say that

- J is *in the range of* b (equivalently b is *able to listen to* J) if $J \cap I \neq \emptyset$ (the houses's operation times overlap);
- J is *monitored* by b (equivalently b *monitors* J) if the bug b is tuned to listen to J.

We denote by $M(b)$ the set of safe houses monitored by the bug b,

$$M(b) = \{J \in \mathcal{I} \mid J \text{ is monitored by } b\}.$$

Given a set of listening devices B on $\mathcal{I}$, together with the associate monitored set of houses ($M(b)$ with $b \in B$), a *bug signature* (or code) of a safe house J is

$$S_B(J) = \{b \in B \mid J \in M(b)\}^2.$$

▶ **Definition 2** (Surveillance System (SS)). *A set of listening devices B, located in $\mathcal{I}$ is a* Surveillance System *for $\mathcal{I}$ if one can determine the monitored sets $M(b)$ for $b \in B$, so that there exists signatures $S_B(I)$, for $I \in \mathcal{I}$, such that*

$$S_B(I) \neq \emptyset \quad and \quad S_B(I) \neq S_B(J) \qquad for\ all\ distinct\ I, J \in \mathcal{I}. \tag{3}$$

We can now formally state agent X's problem.

Surveillance System (SS)
Input: A set of intervals (safe houses) $\mathcal{I} = (I_1, I_2, \ldots, I_m)$.
Question: Find a minimum size Surveillance System B for $\mathcal{I}$.

2.3 From Field Intelligence to Graph Theory

Before proceeding to the algorithmic dossier, we highlight the exact correspondence between our spycraft terminology and its graph-theoretic counterpart.

The Underlying Watching-System Framework

Watching systems have been introduced in [1] as a more flexible framework with respect to identifying codes. An *Identifying Code* (IC) of a graph $G = (V, E)$ is a subset $\mathcal{C} \subseteq V$ such that sets $\{N[v] \mid v \in \mathcal{C}\}$ are nonempty and distinct, where $N[v]$ denotes the closed neighborhood of the vertex v (the set containing v and all its neighbours)[15]. The goal is to minimize the size of $\mathcal{C}$. For a comprehensive treatment of combinatorial and algorithmic aspects of identifying codes, see also [9].

In a watching system, if a watcher is placed at a vertex v, a subset of neighbor vertices (watching zone) can be selected instead of the whole closed neighborhood of v. It is also possible to place several watchers at the same vertex, with distinct watching zones.

Formally, a *watching set* is a collection $W = \{(z_i, Z_i)\}_{i=1}^{k}$, where each $z_i \in V$ is a location and $Z_i \subseteq N[z_i]$ is the *watching zone* of watcher i. A watcher $w = (z, Z)$ *covers* a vertex v if $v \in Z$. For each $v \in V$, its *code* is the set

$$C_W(v) = \{\, w \in W \mid v \text{ is covered by } w \,\}.$$

The set W is a *watching system* if all codes are nonempty and pairwise distinct. The goal is to find a watching system W of minimum size.

Various properties of watching systems, including structural characteristics and bounds on the watching number across different graph classes, are derived in [3, 1, 2, 13, 14, 17]. For a comprehensive bibliography on identifying, locating-dominating and watching systems, see [16]. We stress that the watching-system framework is strictly more general than identifying codes, and often yields much smaller solutions. Related identification frameworks include locating-dominating sets and their variants [8, 10, 18].

A constrained variant of Watching Systems, called the β-Watching System, in which a single vertex can host up to a bounded number β of watchers, has been recently introduced. For general graphs, the β-Watching System problem remains *hard* (even when $\beta = 1$), but it admits a $(2 \log n + 1)$-approximation algorithm. Moreover, when the underlying graph is a tree, an optimal β-Watching System can be computed in polynomial time [7].

Summarizing: The Surveillance System problem defined in the spycraft narrative is a direct instantiation of Watching Systems on an interval graph: Safe houses $I_1, \ldots, I_m$ are the vertices of an interval graph. Each Listening Device (bug) located in a safe house I plays exactly the role of a *watcher* located in vertex I in the corresponding interval graph. A safe house J is *monitored* by a bug located in I precisely when J belongs to the chosen subset of neighbors on which the watcher located in I is probing. Thus, the set of bugs that can hear J – the bug signature $S(J)$ – coincides with the watcher code $C_W(J)$ of the corresponding vertex in a watching system.

In our work, we instantiate this framework on interval graphs induced by time-intervals of safe houses: this specialization both justifies the surveillance (spy) interpretation and allows us to exploit interval-graph structure to design a polynomial-time factor-2 approximation algorithm for surveillance assignments.General bounds for identifying codes in terms of structural parameters such as maximum degree are studied in [12]. We stress that the identifying code problem remains NP-Hard on interval graphs [11] and a factor-6 approximation algorithm for interval graphs was presented in [6]; to the best of our knowledge, no specific results were previously known for the Watching System problem.

Our Results

In section 3, we present the SURVEILLANCE strategy. More precisely, in section 3.1 we introduce an algorithm that serves as a building block for the proposed SURVEILLANCE algorithm, which is then fully described in section 3.2. In section 4, we prove that the Surveillance System problem is NP-hard on chordal and bipartite graphs.

3 The Timeline Deployment Strategy

To secure the spy network with the fewest possible listening devices, Headquarters follows a rigorously codified deployment protocol – part greedy improvisation, part Cold-War bureaucracy carved in stone.

The operation begins by ordering all safe houses according to the *Timeline Ordering* introduced earlier (increasing right endpoint, with ties broken by decreasing left endpoint). This establishes the sequence in which the intervals $I_1 \prec I_2 \prec \cdots \prec I_m$ are processed.

For each safe house I_i in this order, the following procedure is executed:

1. **Attempt Signature Assignment.** Look for a bug signature that I_i can receive by using the bugs already deployed; that is, all existing bugs located in intervals overlapping I_i and can, therefore, be tuned to monitor I_i.
   If a signature, which is nonempty and distinct from all previously assigned ones, exists, then nothing further is required: the safe house is uniquely identifiable.
2. **Deploy a New Bug.** Otherwise, a new bug must be introduced to guarantee a unique signature for I_i. By operational decree, this new bug must be placed at the *rightmost safe house* that still intersects I_i. Once positioned, the new bug is tuned to monitor I_i (and potentially other overlapping houses, as needed).

The introduction of a new bug may require a local adjustment of signatures for preceding safe houses: Some safe houses may now be monitored by this bug, others may cease to be monitored by existing ones in order to maintain consistency and injectivity. Whenever such changes occur, the algorithm recomputes the affected signatures – always following the established timeline order.

An example of the Timeline Deployment Strategy is given in figure 2.

This procedure shows that the safe houses can be secured with a *number of bugs* which is at most twice the optimal.

3.1 A building block: Compute Signatures

In this section, we present an algorithm that serves as the basis for the proposed SURVEILLANCE algorithm, used later in section 3.2, namely the algorithm COMPUTESIGNATURES.

To describe the algorithm, we introduce an order on signatures, which is essentially the standard lexicographic ordering of subsets of a totally ordered set.

▶ **Definition 3** (Lexicographic Order on Bug Signatures). *Let* $X = \{x_1, x_2, \cdots, x_s\}$ *and* $Y = \{y_1, y_2, \cdots, y_t\}$ *be two ordered subsets of* $[k]$*, that is,*

$$x_1 < x_2 < \cdots < x_s \qquad \text{and} \qquad y_1 < y_2 < \cdots < y_t.$$

We say that $X \prec_{\text{lex}} Y$ *if and only if one of the following holds:*

1. $x_1 = y_1, \ldots, x_{j-1} = y_{j-1}$*, and* $x_j < y_j$*, for some* $j \leq \min(s, t)$*;*
2. $x_j = y_j$ *for all* $j = 1, \ldots, s$ *and* $s < t$*.*

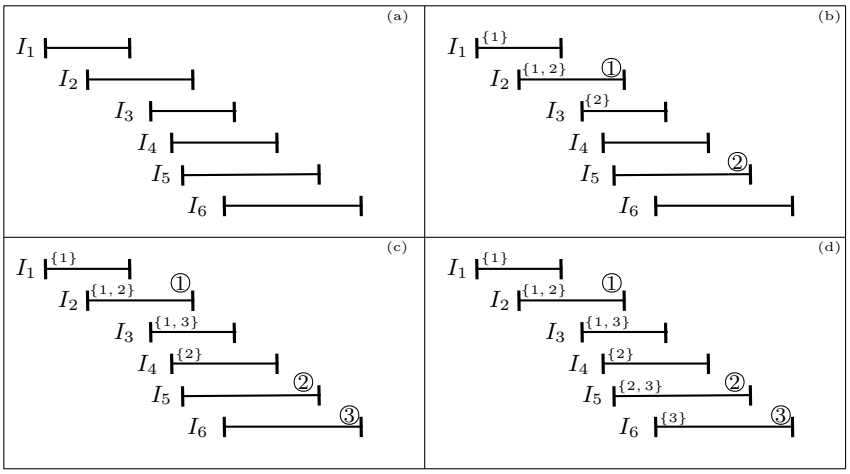

Figure 2 (a) Six safe houses are represented by the intervals $I_1, I_2, \ldots, I_6$ ordered according to the *Timeline Ordering* in (1). (b) Two bugs ①,② have been located in I_2 and I_5, allowing us to identify distinct signatures for the first three intervals (signatures are shown inside the intervals while bugs appear as circled numbers). (c) Another bug ③ is located in I_6, and accordingly, some signatures are recomputed. (d) A Surveillance System composed of three bugs.

▶ **Definition 4.** *Let $\mathcal{I} = \{I_1, \ldots, I_m\}$ be a family of intervals (safe houses), and let $B = \{1, 2, \ldots, k\}$ be a set of bugs. For each $j \in [m]$, we define Σ_j as the set of bugs able to listen to I_j*

$$\Sigma_j = \{b \in [k] \mid I_j \text{ is in the range of } b\}.$$

▶ **Definition 5** (Partial Surveillance System (PSS))**.** *A set of listening devices B, located in $\mathcal{I} = \{I_1, \ldots, I_m\}$, is a* Partial Surveillance System *for $\mathcal{I}_i = \{I_1, \ldots, I_i\}$ if one can determine the sets $M(b) = \{I_j \in \mathcal{I}_i \mid b \in \Sigma_j$ and I_j is monitored by $b\}$, for $b \in B$, such that the signatures $S_j = \{b \in B \mid I_j \in M(b)\}$, for $I_j \in \mathcal{I}_i$ satisfy*

$$S_j \neq \emptyset \quad \text{and} \quad S_j \neq S_{j'} \text{ for all } 1 \leq j, j' \leq i \text{ with } j \neq j'. \tag{4}$$

Clearly, for $i = m$ the above becomes Definition 2.
The following observation will be used to characterize (Partial) Surveillance System by specifying the signatures associated with each safe house.

▶ **Observation 6.** *Each Surveillance System B for $\mathcal{I} = \{I_1, \ldots, I_m\}$ is uniquely identified by the signatures S_j associated with each safe house I_j, for every $j \in [m]$*[3].

Indeed, if $b \in B$ is located in interval I, then b must be tuned to listen to the interval in the set $M(b)$, where $M(b) = \{ I_j \in \mathcal{I} \mid b \in S_j\}$.

The following fact will be useful in the following proofs.

▶ **Fact 7.** *Consider the ordered intervals $I_1 \prec \ldots \prec I_m$. For any a, b, c such that $1 \leq a \leq c \leq b \leq m$, if $I_a \cap I_b \neq \emptyset$ then $I_c \cap I_b \neq \emptyset$.*

Proof. Since $I_a \prec I_b$ and $I_a \cap I_b \neq \emptyset$, by (1) we have $\ell(I_b) \leq r(I_a)$. Furthermore, for each I_c with $a \leq c \leq b$, it holds $r(I_a) \leq r(I_c) \leq r(I_b)$. Hence, $\ell(I_b) \leq r(I_c) \leq r(I_b)$, implying $I_c \cap I_b \neq \emptyset$. ◀

[3] For a positive integer a, we use $[a]$ to denote the set of integers $[a] = \{1, 2, \ldots, a\}$.

The following lemma states that if a given set of bugs forms a Partial Surveillance System, then one can identify the signatures that certify this property by exploiting the lexicographic order on bug signatures.

▶ **Lemma 8.** *Let $\mathcal{I} = \{I_1, \ldots, I_m\}$ be a family of intervals (safe houses), and let B be a Partial Surveillance System for $\mathcal{I}_i$. Let $\mathcal{S} = (S_1, S_2, \ldots, S_i)$ be the list of signatures where, for each $j \in [i]$, S_j is the lexicographically smallest signature on Σ_j excluding $\{S_1, \ldots, S_{j-1}\}$. (e.g. $S_j \neq S_1, \ldots, S_j \neq S_{j-1}$). Then $\mathcal{S}$ satisfies condition* (4).

Proof. By hypothesis, B is a PSS for $\mathcal{I}_i$. Hence, at least a list of signature $\mathcal{S}' = (S'_1, \ldots, S'_i)$ satisfying (4) exists. In the following, we transform $\mathcal{S}'$ to match $\mathcal{S}$. We proceed by induction on i.

Base case: $i = 1$. If $S'_1 = S_1$, then there is nothing to prove. If the signature S_1 does not appear in $\mathcal{S}'$, then it is sufficient to set $S'_1 = S_1$.

Otherwise, S_1 appears in $\mathcal{S}'$ as the signature of some interval I_p with $p > 1$, i.e., $S'_p = S_1$. We show that the signatures of I_1 and I_p can be swapped in $\mathcal{S}'$ that is $S'_p \subseteq \Sigma_1$ and $S'_1 \subseteq \Sigma_p$, so that $\mathcal{S}'$ and $\mathcal{S}$ agree on I_1.

Since $S'_p = S_1 \subseteq \Sigma_1$, it remains to prove that $S'_1 \subseteq \Sigma_p$. Suppose, for contradiction, that the signature $S'_1 \not\subseteq \Sigma_p$. This implies that there exists a bug $z \in S'_1$ with $z \notin \Sigma_p$. Let z be located in I_h, we have $I_h \cap I_p = \emptyset$.

Because S_1 is the first signature in the construction of $\mathcal{S}$, we know $S_1 = \{b\}$, where bug b is the first bug in Σ_1 (according to (2)). Let bug b be located in the interval I_q with $q \geq 1$.

We know that:

- $S'_p = S_1 = \{b\}$ implies that bug $b \in \Sigma_p$ and, consequently,

  $$I_q \cap I_p \neq \emptyset$$

- $z \in S'_1$, and consequently $z \in \Sigma_1$. Moreover, $S_1 = \{b\}$, and S_1 is the first signature in lexicographic orderon Σ_1. It follows that $b < z$ and by (2) consequently

  $$q \leq h.$$

We distinguish two cases:

Case I: $p \leq h$. Recalling that $z \in \Sigma_1$ and z is located in I_h, we have $I_1 \cap I_h \neq \emptyset$. Using $p \leq h$ and $I_1 \cap I_h \neq \emptyset$, we know by Fact 7 that I_p must also intersect I_h, which contradicts $I_h \cap I_p = \emptyset$.

Case II: $h < p$. Using $q \leq h < p$ and $I_q \cap I_p \neq \emptyset$, we know by Fact 7 that I_h must also intersect I_p, which contradicts $I_h \cap I_p = \emptyset$.

Thus, the contradiction shows that the signature S'_1 *can* be assigned to I_p and the swap is feasible.

Inductive step. Assume that lists $\mathcal{S}'$ and $\mathcal{S}$ agree on their first $i-1$ entries. Since Σ_i depends only on the bug placement and both the lists $\mathcal{S}'$ and $\mathcal{S}$ agree on their first $i-1$ entries, both S'_i and S_i belong to the same set that is the set of all the subsets of Σ_i excluding $\{S_1, \ldots, S_{i-1}\}$. This set is nonempty because $\mathcal{S}'$ satisfies (4).

If $S'_i = S_i$, or if S_i does not appear in $\mathcal{S}'$, we are done. Otherwise, S_i appears in $\mathcal{S}'$ as the signature of some interval I_p with $p > i$, i.e., $S'_p = S_i$.

We show that the signatures of I_i and I_p can be swapped in $\mathcal{S}'$ that is $S'_p \subseteq \Sigma_i$ and $S'_i \subseteq \Sigma_p$, obtaining that $\mathcal{S}'$ and $\mathcal{S}$ agree on their first i entries.

Algorithm 1 ComputeSignatures$(\mathcal{I}, \mathbf{k}, i)$.

Input: A set of intervals $\mathcal{I} = \{I_1, \ldots, I_m\}$, a vector of integers $\mathbf{k} = (k_1, \ldots, k_m)$ where each k_j is the number of bugs located in I_j, and an integer i.

Output: Whether there exists a PSS for $\mathcal{I}_i$ compatible with $\mathbf{k}$. In case of a positive answer, a signature assignment is provided.

```
1  id ← 1
2  for j = 1, 2, ..., m do
3  |  Σ_j ← ∅,   B_j ← ∅
4  |  for t = 1, ..., k_j do  B_j ← B_j ∪ {id}, id ← id + 1
5  for j = 1, 2, ..., m do
6  |  N_j ← { h | I_h ∩ I_j ≠ ∅ }
7  |  foreach h ∈ N_j do  Σ_h ← Σ_h ∪ B_j
8  for j = 1, 2, ..., i do
9  |  S_j ← MIN_lex(Σ_j, {S_1, ..., S_{j-1}})      // S_j is the lexicographically smallest signature
   |                                                  on Σ_j with S_j ≠ S_1, ..., S_j ≠ S_{j-1}
10 |  if (S_j=null) then return (false, ∅)   // S_j is null if no admissible signature remains
11 return (true,  B = (S_1, ..., S_i))
```

Since $S'_p = S_i \subseteq \Sigma_i$, it remains to prove that $S'_i \subseteq \Sigma_p$. Suppose, for contradiction, that the signature $S'_i \not\subseteq \Sigma_p$. This implies that there exists a bug $z \in S'_i$ with $z \notin \Sigma_p$. Let z be located in I_h, we have $I_h \cap I_p = \emptyset$.

Let b be the first bug (under (2)) in the signature S_i and let it be located in interval I_q. We know that:

- Since $b \in S_i = S'_p$, we have that bug $b \in \Sigma_p$ and consequently

$$I_q \cap I_p \neq \emptyset$$

- $z \in S'_i$, and consequently $z \in \Sigma_i$. Moreover, $b \in S_i$, and S_i is the first signature in lexicographic order taken from the set of all the subsets of Σ_i excluding $\{S_1, \ldots, S_{i-1}\}$. It follows that $b < z$ and by (2) consequently

$$q \leq h.$$

We distinguish two cases:

Case I: $p \leq h$. Recalling that $z \in \Sigma_i$ and z is located in I_h, we have $I_i \cap I_h \neq \emptyset$. Using $i < p \leq h$ and $I_i \cap I_h \neq \emptyset$, we know by Fact 7 that I_p must also intersect I_h, which contradicts $I_h \cap I_p = \emptyset$.

Case II: $h < p$. Using $q \leq h < p$ and $I_q \cap I_p \neq \emptyset$, we know by Fact 7 that I_h must also intersect I_p, which contradicts $I_h \cap I_p = \emptyset$.

The contradiction shows that the signature S'_i *can* be assigned to I_p, and the swap is feasible. ◀

Exploiting the result of Lemma 8, the algorithm ComputeSignatures checks whether a prescribed set B of bugs, located in the intervals of $\mathcal{I}$, yields a Partial Surveillance System (PSS) for $\mathcal{I}_i = \{I_1, \ldots, I_i\}$. If so, the algorithm returns the bug-signature list $\mathcal{B}$, which coincides with the signature list $\mathcal{S}$ defined in the statement of Lemma 8.

Algorithm 2 SURVEILLANCE($\mathcal{I}$).

Input: A set of intervals $\mathcal{I} = \{I_1, \ldots, I_m\}$.
Output: A Surveillance System B, encoded by a vector $\mathbf{k} = (k_1, \ldots, k_m)$ specifying the number of bugs in each I_i, along with the associated signatures $\mathcal{B}$.

```
1 for i = 1, 2, ..., m do  k_i ← 0,  // k_i is the number of bugs located in the interval I_i
2 for i = 1, 2, ..., m do
3     (Res, B) ← ComputeSignatures(I, k, i)
4     if (Res = false) then
5         h ← max{j | I_j ∈ I, I_j ∩ I_i ≠ ∅}
6         k_h ← k_h + 1                          // The new bug is located in I_h
7 (Res, B) ← ComputeSignatures(I, k, m)
8 return (k, B)
```

The input to the procedure COMPUTESIGNATURES consists of: a set of intervals (safe houses) $\mathcal{I} = \{I_1, \ldots, I_m\}$; a vector $\mathbf{k} = (k_1, \ldots, k_m)$ of integers; and an index $i \in [m]$. The vector $\mathbf{k}$ specifies a candidate set of bugs B with the associated locations: specifically, it indicates how many bugs are placed in each safe house.

The algorithm begins by assigning identifiers to the bugs in increasing order (lines 1–4), consistently with the ordering (2). The sets B_j record the bugs located in each interval I_j. Next, for each $j \in [m]$, the procedure constructs the set Σ_j of all bugs able to listen to I_j (lines 5–7). Finally, the intervals are processed in timeline order. For each I_j, the algorithm selects the *lexicographically smallest* signature (according to Definition 3) on Σ_j with $S_j \neq S_1, \ldots, S_j \neq S_{j-1}$, provided such a signature exists (line 9). If no admissible signature remains ($S_j = null$), then no Surveillance System on $\mathcal{I}$ can be obtained using the multiplicities prescribed by $\mathbf{k}$, and the algorithm returns *false*. Otherwise, it returns *true* together with the list $\mathcal{B}$ of the distinct nonempty bug signatures assigned to the intervals $I_1, \ldots, I_i$.

It is important to note that the algorithm only needs to look for the first signature in lexicographic order that does not belong to $\{S_1, \ldots, S_{j-1}\}$. Since at most $j-1$ signatures are excluded, it suffices to generate up to the first j signatures on Σ_j in lexicographic order.

The signatures constructed by the algorithm exactly match the choice described in Lemma 8, namely, each signature S_j is the *lexicographically smallest* one (according to Definition 3) on Σ_j avoiding the previously assigned signatures $\{S_1, \ldots, S_{j-1}\}$. As a consequence, Lemma 8 immediately implies the correctness of COMPUTESIGNATURES, formally stated in Lemma 9.

▶ **Lemma 9.** *Given $\mathcal{I} = \{I_1, \ldots, I_m\}$, let $\boldsymbol{k} = (k_1, \ldots, k_m)$ be a vector of non-negative integers, which characterizes a set of bugs B. The algorithm* COMPUTESIGNATURES$(\mathcal{I}, \mathbf{k}, i)$ *returns* true*, if and only if B yields a Partial Surveillance System for $\mathcal{I}_i = \{I_1, \ldots, I_i\}$.*

3.2 The Timeline Deployment Strategy Algorithm

The algorithm SURVEILLANCE($\mathcal{I}$) computes a Surveillance System B for a set of safe houses $\mathcal{I} = \{I_1, \ldots, I_m\}$.

The algorithm initializes the bug vector $\mathbf{k} = (0, \ldots, 0)$. It then proceeds iteratively. At step i, assuming $\mathbf{k}$ provides a Partial Surveillance System (PSS) for $\mathcal{I}_{i-1} = \{I_1, \ldots, I_{i-1}\}$, the procedure COMPUTESIGNATURES checks if $\mathbf{k}$ also covers the new interval I_i (line 3). If

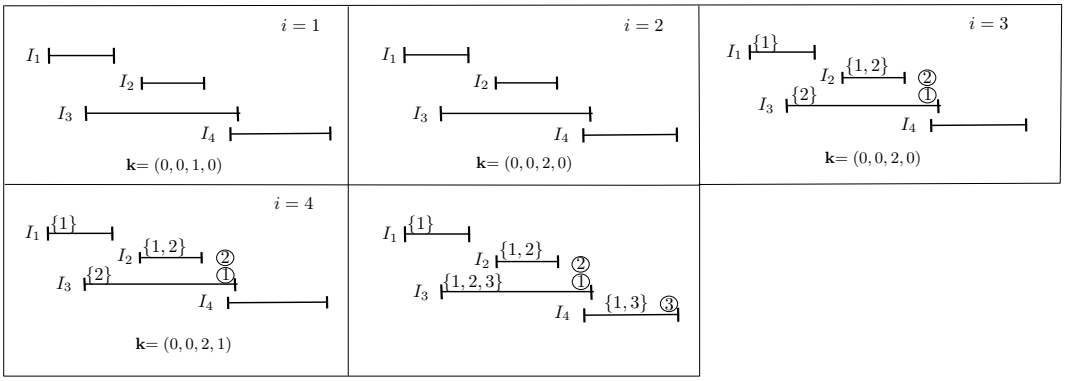

Figure 3 An example illustrating the execution of the algorithm SURVEILLANCE($\mathcal{I}$).

it does not, $\mathbf{k}$ is extended by adding a single bug within the rightmost interval intersecting I_i (lines 5–6). This update ensures $\mathbf{k}$ becomes a PSS for $\mathcal{I}_i = \mathcal{I}_{i-1} \cup \{I_i\}$ (see Lemma 12). Consequently, after processing all inputs, the final vector $\mathbf{k}$ yields a Surveillance System for the entire set $\mathcal{I}$. An example of how Algorithm SURVEILLANCE works is given in Example 10, while Example 11 shows that Algorithm SURVEILLANCE may not be optimal.

▶ **Example 10.** The set of intervals $\mathcal{I} = \{I_1, I_2, I_3, I_4\}$ depicted in figure 3 is the input of Algorithm SURVEILLANCE. At the beginning, vector $\mathbf{k} = (0, 0, 0, 0)$.
At step $i = 1$, COMPUTESIGNATURES($\mathcal{I}, \mathbf{k}, 1$) returns false and the vector $\mathbf{k}$ is updated so that $k_3 = 1$, (i.e., I_3, the rightmost interval intersecting I_1, must locate the first bug).
At step $i = 2$, COMPUTESIGNATURES($\mathcal{I}, \mathbf{k}, 2$) again returns false (one bug is not enough to give distinct signatures to I_1 and I_2) and the vector $\mathbf{k}$ is updated so that $k_3 = 2$.
At step $i = 3$, COMPUTESIGNATURES($\mathcal{I}, \mathbf{k}, 3$) locates bug ① and ② in I_3, and assigns signatures to I_1, I_2, I_3 returning $(\text{true}, (S_i = \{1\}, S_2 = \{1, 2\}, S_3 = \{2\}))$. The vector $\mathbf{k}$ remains unchanged.
At step $i = 4$, COMPUTESIGNATURES($\mathcal{I}, \mathbf{k}, 4$) returns false and the vector $\mathbf{k}$ is updated so that $k_4 = 1$, (i.e., I_4, the rightmost interval intersecting I_3 must locate a new bug).
Finally, COMPUTESIGNATURES($\mathcal{I}, \mathbf{k}, 4$) is called again to assign the final signatures $S_1 = \{1\}, S_2 = \{1, 2\}, S_3 = \{1, 2, 3\}, S_4 = \{1, 3\}$ to the intervals in $\mathcal{I}$.

▶ **Example 11.** The suboptimality of algorithm SURVEILLANCE is depicted in figure 4: for the same family of intervals $\mathcal{I} = \{I_1, \cdots, I_7\}$, the algorithm SURVEILLANCE produces a solution with four bugs (left), whereas there exists an optimal solution that uses only three bugs (right).

For each $i \in [m]$, define $q(i) = \max\{j \in [m] \mid I_i \cap I_j \neq \emptyset\}$ as the index of the rightmost interval that intersects I_i. Let $Q = \{j \in [m] \mid \exists\, i \in [m] \text{ such that } q(i) = j\}$, we notice that, Algorithm SURVEILLANCE places bugs only in safe houses indexed by elements of Q.

The following lemma shows that starting from a PSS for $\mathcal{I}_{i-1}$ and adding a single bug located in $q(i)$ is sufficient to obtain a PSS for $\mathcal{I}_i$.

▶ **Lemma 12.** *Given $\mathcal{I} = \{I_1, \ldots, I_m\}$ and a vector $\boldsymbol{k} = (k_1, \ldots, k_m)$ that yields a PSS for $\mathcal{I}_{i-1} = \{I_1, \ldots, I_{i-1}\}$, for some $i < m$. If $\boldsymbol{k}$ does not yield a PSS for $\mathcal{I}_i = \{I_1, \ldots, I_i\}$, then $\boldsymbol{k}' = (k'_1, \ldots, k'_m)$ yields a PSS for $\mathcal{I}_i$, with*

$$k'_j = \begin{cases} k_j + 1 & \text{if } j = q(i), \\ k_j & \text{otherwise,} \end{cases}$$

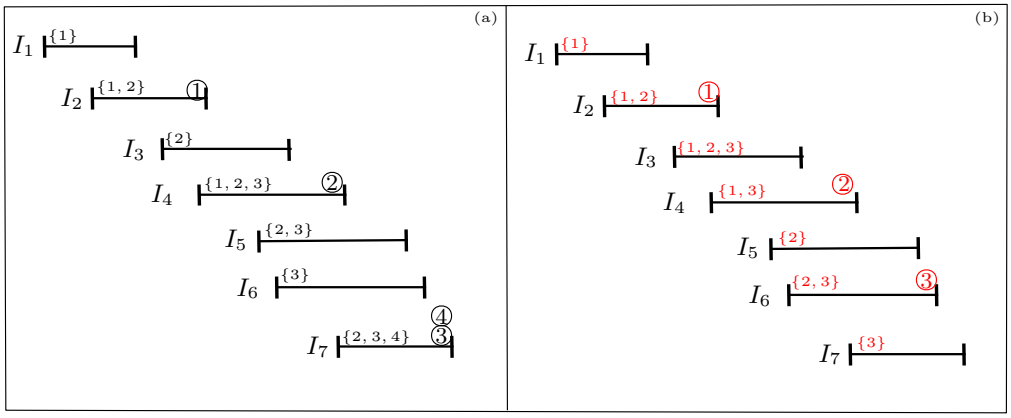

Figure 4 An example illustrating the suboptimality of Algorithm SURVEILLANCE.

Proof. We assume that $\mathbf{k}$ yields a PSS for $\mathcal{I}_{i-1}$. Let $\mathcal{S} = (S_1, \ldots, S_{i-1})$ be a list of nonempty, pairwise distinct signatures assigned to the intervals of $\mathcal{I}_{i-1}$ and associated with $\mathbf{k}$. By adding one additional bug b to $I_{q(i)}$, we obtain the new signature $\{b\}$, which is available for I_i and distinct from S_j for every $j \in [i-1]$. Therefore, the updated vector $\mathbf{k}'$ yields a PSS for $\mathcal{I}_i$. ◀

Hence, at the end of the **for**-loop (lines 2–6), we have that the vector $\mathbf{k}$ yields a SS for $\mathcal{I}$. The procedure COMPUTESIGNATURES is then invoked once more (line 7) to compute the signature $\mathcal{B}$ associated with each interval in $\mathcal{I}$.

The following theorem establishes that the SURVEILLANCE algorithm is a 2-approximate algorithm.

▶ **Theorem 13.** *Given a family of intervals (safe houses) $\mathcal{I} = \{I_1, \ldots, I_m\}$, Algorithm* SURVEILLANCE *returns, in polynomial time, a 2-approximate solution Surveillance System for $\mathcal{I}$ identified by the pair $(\boldsymbol{k}, \mathcal{B})$, where $\boldsymbol{k} = (k_1, k_2, \ldots, k_m)$ specifies the number of bugs located in each safe house and $\mathcal{B} = (S_1, \ldots, S_m)$ assigns a unique signature to each safe house in $\mathcal{I}$.*

Proof. Let $\mathbf{k} = (k_1, k_2, \ldots, k_m)$ be the vector returned by the algorithm, and let $\mathcal{B} = (S_1, \ldots, S_m)$ denote the corresponding list of signatures (chosen according to lexicographic order). Let $\mathbf{o} = (o_1, o_2, \ldots, o_m)$ be an optimal solution, and let $\mathcal{C} = (C_1, C_2, \ldots, C_m)$ be the list of associated signatures (chosen according to lexicographic order).

We show that $\mathbf{o}$ and $\mathcal{C}$ can be transformed into $\mathbf{k}$ and $\mathcal{B}$, respectively, by adding some bugs in $\mathbf{o}$. We refer to the bugs present in the optimal solution $\mathbf{o}$ as *original bugs* and to the additional ones as *cloned bugs*. We show that at the end of the process, there is at most one cloned bug for each original one. This implies that $\mathbf{k}$ yields a 2-approximate solution.

We transform $\mathbf{o}$ and $\mathcal{C}$ step by step, starting from $\mathbf{o}^{(0)} = \mathbf{o}$, to $\mathbf{o}^{(1)}, \mathbf{o}^{(2)}, \ldots, \mathbf{o}^{(m)} = \mathbf{k}$. Similarly, we will have $\mathcal{C}^{(0)} = \mathcal{C}$, to $\mathcal{C}^{(1)}, \mathcal{C}^{(2)}, \ldots, \mathcal{C}^{(m)} = \mathcal{B}$. We denote by Ω_j the set of bugs that can listen to I_j at the current iteration, according to $\mathbf{o}^{(i)}$; we stress that the bugs are identified with integers from 1 to $\sum_{\ell=1}^{m} o_\ell^{(i)}$ and are ordered according to (2). The following invariants are maintained by each $\mathbf{o}^{(i)}$ and the list of associated signatures $\mathcal{C}^{(i)}$.

▷ Claim 14. For each $i \in [m]$, the vector $\mathbf{o}^{(i)} = (o_1^{(i)}, o_2^{(i)}, \ldots, o_m^{(i)})$ and the corresponding list of associated signatures $\mathcal{C}^{(i)} = (C_1^{(i)}, C_2^{(i)}, \ldots, C_m^{(i)})$ yields a Surveillance System for $\mathcal{I}$ such that

(a) $k_j = o_j^{(i)}$ for each $j < q(i)$,

(b) $S_j = C_j^{(i)}$ for each $j = 1, \ldots, i$.

Base case: $i = 1$. We recall that by construction, Algorithm SURVEILLANCE places bugs only in safe houses indexed by elements of Q. Moreover, for each $j < q(1)$ we have $q(j) \geq q(1)$, and thus $k_j = 0$, while $k_{q(1)} > 0$ (because the first bug is placed in $q(1)$).

Starting from $\mathbf{o}^{(0)} = \mathbf{o}$, we shift all bugs located in $I_1, I_2, \ldots, I_{q(1)-1}$ to $I_{q(1)}$. Formally, we define $\mathbf{o}^{(1)} = (o_1^{(1)}, \ldots, o_m^{(1)})$ by

$$o_j^{(1)} = \begin{cases} 0, & \text{if } j < q(1), \\ \sum_{h=1}^{q(1)} o_h^{(0)}, & \text{if } j = q(1), \\ o_j^{(0)}, & \text{otherwise.} \end{cases} \tag{5}$$

The total number of bugs is unchanged.

Feasibility. We show that $\mathbf{o}^{(1)}$ still yields a Surveillance System for $\mathcal{I}$. Let a be bug located in I_p with $p < q(1)$. For any index j such that $a \in \Omega_j$ (i.e., $I_p \cap I_j \neq \emptyset$), we show that $I_{q(1)} \cap I_j \neq \emptyset$, which implies that a can be shifted to $I_{q(1)}$.

Case I: $j \leq q(1)$. Since $I_1 \cap I_{q(1)} \neq \emptyset$, by Fact 7, we obtain $I_{q(1)} \cap I_j \neq \emptyset$.

Case II: $j > q(1)$. Since $p < q(1) < j$ and $I_p \cap I_j \neq \emptyset$, Fact 7, we obtain $I_{q(1)} \cap I_j \neq \emptyset$.

Thus, a belongs to Ω_j also after being shifted to $I_{q(1)}$. Repeating this argument for all such bugs yields the solution $\mathbf{o}^{(1)}$.

Claim 14-(a). Since $k_j = o_j^{(1)} = 0$ for all $j < q(1)$, Claim 14-(a) holds.

Claim 14-(b). If $S_1 = C_1^{(0)}$, there is nothing to prove and $\mathcal{C}^{(1)} = \mathcal{C}^{(0)} = \mathcal{C}$.
If S_1 does not appear in $\mathcal{C}^{(0)}$, we simply set $C_1^{(1)} = S_1$ and $C_j^{(1)} = C_j^{(0)}$ for each $j > 1$.
Otherwise, S_1 appears in $\mathcal{C}^{(0)}$ as $C_t^{(0)}$ for some $t > 1$. We show that the signatures assigned to I_1 and I_t can be swapped in $\mathcal{C}^{(0)}$.

By signatures construction and (5), $C_t^{(0)} = S_1 \subseteq \Omega_1$; it then suffices to show that $C_1^{(0)} \subseteq \Omega_t$.

We first show that $I_{q(1)} \cap I_t \neq \emptyset$. Since $C_t^{(0)} = S_1$, each bug $b \in C_t^{(0)}$ can listen to both I_1 and I_t, and therefore is located in some interval I_p with $p \leq q(1)$ such that $I_p \cap I_t \neq \emptyset$.

If $t \leq q(1)$, then $1 \leq t \leq q(1)$ and, since $I_1 \cap I_{q(1)} \neq \emptyset$, Fact 7, we obtain $I_{q(1)} \cap I_t \neq \emptyset$. Otherwise, if $t > q(1)$, then using $p < q(1) < t$ and $I_p \cap I_t \neq \emptyset$, by Fact 7, we obtain $I_{q(1)} \cap I_t \neq \emptyset$.

Since $C_1^{(0)} \subseteq \Omega_1$, all bugs in $C_1^{(0)}$ are now located in $I_{q(1)}$. Because $I_{q(1)} \cap I_t \neq \emptyset$, we conclude that $C_1^{(0)} \subseteq \Omega_t$. Therefore, the swap is feasible, we set $C_1^{(1)} = C_t^{(0)} = S_1$, $C_t^{(1)} = C_1^{(0)}$ and $C_j^{(1)} = C_j^{(0)}$ for each $j \notin \{1, t\}$ and Claim 14-(b) holds.

Inductive step $1 < i \leq m$. Consider a Surveillance System satisfying Claim 14 for $i - 1$; let it be identified by the vector $\mathbf{o}^{(i-1)} = (o_1^{(i-1)}, \ldots, o_m^{(i-1)})$ with associated list of signatures $\mathcal{C}^{(i-1)}$.

If $q(i) = q(i-1)$, we simply set $\mathbf{o}^{(i)} = \mathbf{o}^{(i-1)}$, moving no bug.

Otherwise, $q(i) > q(i-1)$. We first establish that

$$o_{q(i-1)}^{(i-1)} \geq k_{q(i-1)}. \tag{6}$$

To see why, note that Algorithm SURVEILLANCE introduces a bug only when necessary: specifically, a bug placed at interval $I_{q(i-1)}$ must appear in one of the signatures $S_1, \ldots, S_{i-1}$. By the inductive hypothesis, $\mathcal{B}$ and $\mathcal{C}^{(i-1)}$ share the first $i-1$ signatures; consequently, every bug counted in $k_1 + \cdots + k_{q(i-1)}$ must also be counted in $o_1^{(i-1)} + \cdots + o_{q(i-1)}^{(i-1)}$. Moreover, by Claim 14-(a), $k_j = o_j^{(i-1)}$ for all $j < q(i-1)$. Inequality (6) follows directly.

We now shift all the bugs located in intervals $I_{q(i-1)+1}, \ldots, I_{q(i)-1}$, together with $o_{q(i-1)}^{(i-1)} - k_{q(i-1)}$ bugs at $I_{q(i-1)}$, to $I_{q(i)}$. Formally, we define $\mathbf{o}^{(i)} = (o_1^{(i)}, \ldots, o_m^{(i)})$ as follows:

$$o_j^{(i)} = \begin{cases} o_j^{(i-1)} & \text{if } j < q(i-1) \text{ or } j > q(i), \\ k_j, & \text{if } j = q(i-1), \\ 0 & \text{if } q(i-1) < j < q(i), \\ o_{q(i-1)}^{(i-1)} - k_{q(i-1)} + \displaystyle\sum_{h=q(i-1)+1}^{q(i)} o_h^{(i-1)} & \text{if } j = q(i). \end{cases} \tag{7}$$

Clearly, the total number of bugs is preserved. We stress that after the shifts in (7), any bug located in an interval $I_j \prec I_{q(i)}$ will not be shifted anymore. Recalling that the bugs are numbered from 1 to $\sum_{\ell=1}^{m} o_\ell^{(i)}$ and ordered according to (2), this also implies that from now

$$\Sigma_j \subseteq \Omega_j, \text{ for each } j \leq i. \tag{8}$$

We show now that Claim 14 holds for $\mathbf{o}^{(i)}$.

Feasibility: $\mathbf{o}^{(i)}$ yields a Surveillance System for $\mathcal{I}$. Let a be any bug originally located in an interval I_p, with $q(i-1) \leq p < q(i)$ according to $\mathbf{o}^{(i-1)}$. Suppose a is one of the bugs that are shifted to $I_{q(i)}$. We show that shifting a preserves the feasibility of the Surveillance System.

To this aim, consider any index j such that $a \in \Omega_j$ (i.e., $I_p \cap I_j \neq \emptyset$).

Case $j < i$. By construction, the signatures $S_1, \ldots, S_{i-1}$:

- do not contain any bug located in intervals I_h with $h > q(i-1)$. Indeed, any interval in $\{I_1, \ldots, I_{i-1}\}$ is disjoint from every interval in $\{I_{q(i-1)+1}, \ldots, I_m\}$; therefore, bugs located in $I_{q(i-1)+1}, \ldots, I_m$ cannot listen to any interval among $I_1, \ldots, I_{i-1}$;
- use at most $k_{q(i-1)}$ bugs located in $I_{q(i-1)}$.

Since $\mathcal{B}$ and $\mathcal{C}^{(i-1)}$ coincide on the first $i-1$ signatures, we retain exactly $k_{q(i-1)}$ bugs in $I_{q(i-1)}$ and shift all the bugs that are not contained in $S_1, \ldots, S_{i-1}$. Summarizing, any shifted bug does not appear in any signature $C_j^{(i-1)}$ with $j < i$.

Case $i \leq j \leq q(i)$. Since $I_i \cap I_{q(i)} \neq \emptyset$ and $i \leq j \leq q(i)$, by Fact 7, we obtain $I_{q(i)} \cap I_j \neq \emptyset$. Hence, a remains in Ω_j after the shift.

Case $j > q(i)$. Since $p < q(i) < j$ and $I_p \cap I_j \neq \emptyset$, by Fact 7, we obtain $I_{q(i)} \cap I_j \neq \emptyset$. Again, a remains in Ω_j after the shift.

Repeating this argument for all shifted bugs yields $\mathbf{o}^{(i)}$.

Claim 14-(a). If $q(i) = q(i-1)$, the claim follows immediately from the inductive hypothesis. Otherwise, Claim 14-(a) holds for all $j < q(i-1)$ by induction. Moreover, by (7) we have $o_{q(i-1)}^{(i)} = k_{q(i-1)}$ and $o_j^{(i)} = k_j = 0$ for $q(i-1) < j < q(i)$. Hence, Claim 14-(a) holds for i.

Claim 14-(b). By the inductive hypothesis, $S_1 = C_1^{(i-1)}, \ldots, S_{i-1} = C_{i-1}^{(i-1)}$.

1) If $S_i = C_i^{(i-1)}$, there is nothing to prove and $\mathcal{C}^{(i)} = \mathcal{C}^{(i-1)}$.

2) If S_i does not appear in $\mathcal{C}^{(i-1)}$, we simply set $C_i^{(i)} = S_i$ and $C_j^{(i)} = C_j^{(i-1)}$ for all $j \neq i$.

3) Otherwise, $S_i = C_t^{(i-1)}$ for some $t > i$. by (8), we know that $C_t^{(i-1)} = S_i \subseteq \Sigma_i \subseteq \Omega_i$. Hence, we can again set $C_i^{(i)} = C_t^{(i-1)} = S_i$ and $C_j^{(i)} = C_j^{(i-1)}$ for all $j \notin \{i, t\}$. However, we need $C_t^{(i)}$.

To this aim, denote by b the smallest bug in $C_t^{(i-1)}$.

3.1) If there exists a signature S in which the smallest bug is b such that $S \subseteq \Omega_t$ and $S \notin \mathcal{C}^{(i-1)}$, we assign $C_t^{(i)} = S$.

3.2) Otherwise, a new bug $\tilde{b}$ (hereafter, the clone of b) is located in the interval $I_{q(i)}$. Let b be located in the interval I_ℓ, since $b \in C_t^{(i-1)} = S_i$ we know that $I_i \cap I_\ell \neq \emptyset$. Recalling that $I_{q(i)}$ is rightmost interval intersecting I_i, we know that $\ell \leq q(i)$. Moreover, since $C_t^{(i-1)} = S_i$ we have $I_t \cap I_\ell \neq \emptyset$. Hence we have $I_t \cap I_{q(i)} \neq \emptyset$ and we set $C_t^{(i)} = C_t^{(i-1)} \cup \{\tilde{b}\}$.

We observe that $\tilde{b}$ is located in $I_{q(i)}$ and b is located in I_ℓ with $\ell \leq q(i)$. Hence, we may assume that $b \prec \tilde{b}$ and b is still the smallest bug in $C_t^{(i)}$.

We also notice that for any $t' > i$ such that $b \in C_{t'}$, the interval $I_{t'}$ intersects the interval I_ℓ containing b. Because $q(i) \geq \ell$, the interval $I_{q(i)}$ also intersects $I_{t'}$. This implies that at any subsequent step i', we can still use the clone $\tilde{b}$ located in $I_{q(i)}$ to eventually construct a signature $C_{t'}^{(i')}$; that is, Case 3.2) cannot occur again, and no bug cloning is required.

If case 3.2) applied, we added a new bug $\tilde{b}$ located in $q(i)$, we have $o_{q(i)}^{(i)} = o_{q(i)}^{(i)} + 1$. To maintain the identifiers of the bugs in increasing order according to the ordering (2), we update the bugs identifier and the signatures in $\mathcal{C}^{(i)}$ consistently. We notice that since $\tilde{b}$ is located in $q(i)$, this update affects only the identifiers of the bugs located in intervals after $q(i)$ and consequently does not affect any signature before $C_j^{(i)}$ with $j < i$.

Let k^* be the size of the optimal solution. Initially, all signatures contain only the k^* original bugs of the optimal solution. Moreover, we have already observed that, after cloning a bug b, any interval containing b will not require a new cloned bug, e.g., Case 3.2) occurs at most once for each original bug and consequently, the total number of clones is at most k^*. In conclusion, when reaching $\mathbf{o}^{(m)}$, we have $k_j = o_j^{(m)}$ for all $j < q(m) = m$, and $\mathcal{C}^{(m)}$ coincide with $\mathcal{B}$. Since $\mathcal{B}$ uses all and only the bugs described by $\mathbf{k}$ and $\mathbf{o}^{(m)}$ contains **at most k^* clones**, we conclude that $\mathbf{k}$ yields a 2-approximate solution.

Time complexity. The algorithm ComputeSignatures needs $O(m^3)$ time. The initialization of the variables (lines 1–4) takes $O(m)$ time. Computing the neighborhoods requires $O(m^2)$ time. The computation of all sets Ω_j, for $j \in [m]$, also takes $O(m^2)$ time in total. As observed above, to compute a signature S_j, since at most $j-1$ signatures are excluded, it suffices to generate the first j signatures over Σ_j in lexicographic order and return the first one that is not excluded. The signatures over Σ_j can be generated by a straightforward algorithm that performs an iterative depth-first traversal with a preorder visit of the implicit lexicographic tree of subsets of Σ_j. Since the size of any signature is at most $k^* < m$ (a trivial solution, placing one bug in each safe house, each tuned only to its own house, is always feasible), generating and checking a single signature requires $O(m)$ time. Therefore, generating the first j signatures over Σ_j takes $O(jm)$ time. Summing over all $j \in [m]$, the total time spent generating signatures is $O(m^3)$. Hence, the total running time of ComputeSignatures is $O(m^3)$.

The time complexity of Algorithm SURVEILLANCE is $O(m^4)$. Indeed, the algorithm invokes COMPUTESIGNATURES a total of $m+1$ times, and the computation of h, performed at most m times, requires $O(m^2)$ time.

By using appropriate data structures for the generation and checking of signatures or by means of an amortized analysis, this time complexity can be reduced to $O(m^3)$. ◀

Moral of the story: in Cold War Berlin, as in algorithms, a small amount of redundancy along the timeline buys you certainty at only twice the cost.

4 Hardness

In section 2.3, we highlighted the strong connection between the SURVEILLANCE SYSTEM problem and the WATCHING SYSTEM problem when the input graph is an interval graph.

In this section, we show that if we slightly enlarge the class of input graphs by considering chordal graphs, the problem becomes NP-hard.

It is known that the DOMINATING SET (DS) problem is NP-hard even when restricted to chordal and bipartite graphs [4, 5]. Here, we prove that the Watching System (WS) problem remains NP-hard even when restricted to chordal and bipartite graphs, by means of a reduction from DS.

To this end, we consider the decision version of the WS problem, which asks whether there exists a watching system of the input graph of size at most k. Recall that, given a graph $G=(V,E)$ and an integer k, the DS problem asks whether there exists a set $D \subseteq V$ such that $|D| \leq k$ and $D \cap N(v) \neq \emptyset$ for every $v \in V \setminus D$.

▶ **Theorem 15.** *The Watching System problem remains NP-hard even on chordal and bipartite graphs.*

Proof. Let $\langle G=(V,E),k\rangle$ be an instance of DS. We construct a graph G' from G as follows. For each vertex $v \in V$, we introduce three new vertices v_1, v_2, v_3 and add edges connecting v_1 to v, v_2, and v_3.

By construction, if G is chordal (respectively, bipartite), then G' is also chordal (respectively, bipartite). Indeed, for each vertex $v \in V$, we attach a tree to v. If G is bipartite, attaching a tree to a single vertex preserves bipartiteness; applying this argument to every vertex of G implies that G' is bipartite. Moreover, since the added gadgets are trees, no induced cycle of length greater than three is created. Therefore, if G is chordal, then G' is chordal as well.

In order to complete the proof, we establish the following claim.

▷ Claim 16. *For any integer k,*

$\langle G=(V,E),k\rangle$ *is a YES-instance of* DS

if and only if

$\langle G', k'=k+2|V|\rangle$ *is a YES-instance of WS.*

Assume first that $D \subseteq V$ is a dominating set of G with $|D| \leq k$. We show that

$$W = \{(v, N_{G'}[v]) \mid v \in D\} \cup \{(v_1, \{v, v_1, v_3\}),\ (v_2, \{v_1, v_2\}) \mid v \in V\}.$$

is a watching system for G'.

By construction, $|W| = |D| + 2|V|$. Moreover, at most one watcher is located in each vertex; hence, we identify a watcher with the vertex where it is placed.

We now show that the codes associated with the vertices of G' are all nonempty and pairwise distinct. The codes are as follows:

$$C_W(v_1) = \begin{cases} \{v, v_1, v_2\} & \text{if } v \in D \\ \{v_1, v_2\} & \text{otherwise} \end{cases}, \qquad C_W(v_2) = \{v_2\}, \qquad C_W(v_3) = \{v_1\},$$

and, for each $v \in V$,

$$C_W(v) = \{v_1\} \cup (N_G[v] \cap D).$$

Hence, every vertex of G' has a nonempty code. Furthermore, all codes are pairwise distinct. In particular, for each $v \in V$:

- $C_W(v) \neq C_W(v_1)$, since $v_2 \in C_W(v_1)$ but $v_2 \notin C_W(v)$;
- $C_W(v) \neq C_W(v_3)$, since D is a dominating set and therefore $N_G[v] \cap D \neq \emptyset$.

Finally, for any two distinct vertices $u, v \in V$, the codes $C_W(u)$ and $C_W(v)$ are distinguished by u_1 and v_1, respectively.

Thus, W is a valid watching system for G' of size $|D| + 2|V| \leq k + 2|V|$.

Conversely, let W be a watching system for G' with $|W| \leq k + 2|V|$. For each vertex $v \in V$, consider the gadget formed by v_1, v_2, and v_3. At least two watchers must be placed in $\{v_1, v_2, v_3\}$ in order to distinguish v_2 and v_3.

Moreover, since two watchers located in v_1 are sufficient to distinguish v_1, v_2, and v_3, if three or more watchers are placed in $\{v_1, v_2, v_3\}$, we can obtain an equivalent watching system by moving two watchers to v_1 and relocating any remaining watchers to v. Hence, in the remainder of the proof, we may assume that each gadget hosts exactly two watchers, and consequently that at most k watchers are located in vertices of V.

Let $D \subseteq V$ be the set of vertices of V hosting a watcher. By the above argument, $|D| \leq k$. We claim that D is a dominating set of G.

Suppose, for contradiction, that there exists a vertex $v \in V \setminus D$ such that $N_G(v) \cap D = \emptyset$. Since v is adjacent to v_1 in G', and $C_W(v) \neq \emptyset$, the watchers placed in the gadget of v must distinguish the vertices v, v_1, v_2, and v_3. This is impossible unless either $v \in D$ or $N_G(v) \cap D \neq \emptyset$, contradicting the assumption. Therefore, D is a dominating set of G. ◀

5 Conclusions

In this paper, we sent Agent X on a mission through the timelines of Cold War Berlin and, along the way, studied the *Watching System* problem on interval graphs. By translating safe houses into intervals and listening devices into watchers, we showed that the temporal structure of interval graphs can be exploited in a surprisingly effective way.

Our main result is a *simple polynomial-time 2-approximation algorithm*, built around a timeline deployment strategy: whenever the existing bugs cannot uniquely identify a newly activated safe house, one carefully placed extra device suffices. The algorithm is greedy, local, and easy to implement, yet its analysis reveals a subtle global invariant: each optimal bug may need to be duplicated at most once. In short, certainty along the timeline can be bought at twice the optimal cost.

Not all spy networks are so cooperative. We also proved that as soon as the setting is slightly generalized, to chordal or bipartite graphs, the problem becomes NP-hard. This sharply marks interval graphs as a rare safe zone where surveillance remains algorithmically manageable.

Several mysteries remain unsolved. Is the factor 2 approximation the best Agent X can hope for, or is an even more efficient deployment possible? Can similar timeline-based ideas be adapted to other geometric graph classes? We leave these questions for future investigations and future missions.

For now, the lesson is clear: when safe houses line up neatly on a timeline, a small amount of well-placed redundancy is enough to keep every secret uniquely identified.

References

1 David Auger, Irène Charon, Olivier Hudry, and Antoine Lobstein. Watching systems in graphs: An extension of identifying codes. *Discret. Appl. Math.*, 161(12):1674–1685, 2013. doi:10.1016/j.dam.2011.04.025.

2 David Auger, Irène Charon, Olivier Hudry, and Antoine Lobstein. Maximum size of a minimum watching system and the graphs achieving the bound. *Discret. Appl. Math.*, 164:20–33, 2014. doi:10.1016/j.dam.2012.08.028.

3 David Auger and Iiro S. Honkala. Watching systems in the king grid. *Graphs Comb.*, 29(3):333–347, 2013. doi:10.1007/s00373-S011-S1124-S0.

4 Alan A. Bertossi. Dominating sets for split and bipartite graphs. *Inf. Process. Lett.*, 19(1):37–40, 1984. doi:10.1016/0020-S0190(84)90126-S1.

5 Kellogg S. Booth and J. Howard Johnson. Dominating sets in chordal graphs. *SIAM J. Comput.*, 11(1):191–199, 1982. doi:10.1137/0211015.

6 Nicolas Bousquet, Aurélie Lagoutte, Zhentao Li, Aline Parreau, and Stéphan Thomassé. Identifying codes in hereditary classes of graphs and vc-dimension. *SIAM J. Discret. Math.*, 29(4):2047–2064, 2015. doi:10.1137/14097879X.

7 Gennaro Cordasco, Luisa Gargano, and Adele A. Rescigno. Watching systems with bounded probes. In Luca Moscardelli and Francesca Scozzari, editors, *Proceedings of the 26th Italian Conference on Theoretical Computer Science, Pescara, Italy, September 10-12, 2025*, volume 4039 of *CEUR Workshop Proceedings*, pages 3–15. CEUR-WS.org, 2025. URL: https://ceur-Sws.org/Vol-S4039/paper20.pdf.

8 Gennaro Cordasco, Luisa Gargano, and Adele Anna Rescigno. Red-blue unshared dominators. In *Proceedings of the 25th International Symposium on Fundamentals of Computation Theory (FCT 2025)*, pages 94–108, 2025.

9 Florent Foucaud. *Combinatorial and algorithmic aspects of identifying codes in graphs*. PhD thesis, Université Sciences et Technologies - Bordeaux, 2012.

10 Florent Foucaud, Michael A. Henning, Christian Löwenstein, and Thomas Sasse. Locating-dominating sets in twin-free graphs. *Discret. Appl. Math.*, 200:52–58, 2016. doi:10.1016/j.dam.2015.06.038.

11 Florent Foucaud, George B. Mertzios, Reza Naserasr, Aline Parreau, and Petru Valicov. Identification, location-domination and metric dimension on interval and permutation graphs. II. algorithms and complexity. *Algorithmica*, 78(3):914–944, 2017. doi:10.1007/s00453-S016-S0184-S1.

12 Florent Foucaud and Guillem Perarnau. Bounds for identifying codes in terms of degree parameters. *Discrete Applied Mathematics*, 232:99–114, 2017.

13 Mehdi Ghorbani, Matthias Dehmer, Hossein Maimani, Saeid Maddah, Mohammad Roozbayani, and Frank Emmert-Streib. The watching system as a generalization of identifying code. *Applied Mathematics and Computation*, 380, 2020.

14 Modjtaba Ghorbani and Sheyda Maddah. On the watching number of graphs using discharging procedure. *J. Appl. Math. Comput.*, 67(1-2):507–518, 2021. doi:10.1007/s12190-S020-S01482-Sw.

15 Mark G. Karpovsky, Krishnendu Chakrabarty, and Lev B. Levitin. On a new class of codes for identifying vertices in graphs. *IEEE Trans. Inf. Theory*, 44(2):599–611, 1998. doi:10.1109/18.661507.

16 Antoine Lobstein. Watching systems, identifying, locating-dominating and discriminating codes in graphs: A bibliography. arXiv preprint arXiv:1004.2192, 2010.

17 Maryam Roozbayani and Hamidreza Maimani. Identifying codes and watching systems in kneser graphs. *Discret. Math. Algorithms Appl.*, 9(1):1750007:1–1750007:9, 2017. doi:10.1142/S1793830917500070.

18 Suk Jai Seo and Peter J. Slater. Open neighborhood locating dominating sets. *Australas. J Comb.*, 46:109–120, 2010. URL: http://ajc.maths.uq.edu.au/pdf/46/ajc_v46_p109.pdf.