Computational Topology - group project

The Evasion Problem

Andrej Jočić
Matija Ojo

December 30, 2023

1 Introduction

The evasion problem asks if there is a way to move in an area, covered with moving sensors, and not be detected. We assume the sensors follow a predefined path and their movement is periodic. This means that, after some period, the area covered by sensors is identical. Furthermore, we assume the area in question is a subspace of the plane $\mathbb{R}^2$. These assumptions do not lose generality, since movement of real sesnors is often periodic and movement through an area can be modelled in the plane.

This solution could be applied to the problem of determining the optimal amount of sensors and their paths to cover the area. The goal in this case is not to cover the entire area at all times, but to cover parts of it with fewer sensors, such that a potential intruder cannot avoid detection throughout the whole period.

The following are additional assumptions taken into account, which do lose on generality, however greatly simplify the implementation:

- the intruder's movement speed is unbounded,
- the area in question is a rectangular grid,
- the paths along which the sensors move are straight line segments,
- all sensors move at the same speed of one unit of distance per one unit of time,
- each sensor covers a square area two units of distance across oriented parallel to the edges of the room they protect.

2 Methods

By considering the area not covered by the senors (two-dimensional object) at each point in time $t \in \mathbb{R}$, we obtain a three-dimensional (free) complex F embedded in space-time. We are only interested in the times up to the (global) sensor network period p, since the object simply repeats itself after p. Therefore, the complex we obtain is similar to a torus, where the first point in time is "glued" to the time at $t = p$. One way to obtain the evasion paths would be to compute $H_1(F, \mathbb{Z})$ and check which generators represent vaild cycles/paths (i.e. cycles going strictly forward in time). However, we decided to take a different approach and collapse the free complex to a directed graph and compute cycles from the starting nodes of the graph.

2.1 Sensors

Since all the sensors have the same properties (area covered, speed of movement), they are only defined by the `Path` which they follow. A `Path` is a sequence of `Positions`, where a `Position` is point in the plane (x, y). Since we want the sensors to move along straight line segments, two consecutive `Positions` need to either have the same x or y coordiantes. The length of a `Path` is the sum of all the consecutive line segments that form it. Sensors move towards the next defined `Position` in the `Path`, and move towards the first `Position` when reaching the last one.

The `SensorNetwork` class is a collection of `Sensors` along with the width and height of the rectangular area. `SensorNetwork` represents the entire configuration of the input, since the behaviour of the system is determined by the initial situation. The global period of the system (p) is the least common multiple of all the sensor `Path`'s lengths.

2.2 Free complex

For the data structure of the free complex (F), we used the `CubicalComplex` from the gudhi library. This approach is the simplest, because the area covered by each sensor is a square (ie. a 2-dimensional simplex, defined by 4 points), so we do not have to do any transformation.

We can even further simplify the complex construction, by not taking any wedge shapes into account. A wedge shape is a situation where a cell in the grid was covered at time t, but is not covered at time $t + 1$ or vise versa - not covered at t, covered at $t + 1$. In a 3-dimensional cubical complex that would result in a wedge. We can omit those shapes, because the object trying to avoid detection can either be in a particular cell or not - we do not allow partially taking halves of 2 cells. Therefore, the valid positions for the object, trying to avoid detection, are only those, which are not covered at two consecutive times. If we consider the case where the cell was covered at time t and was not covered at time $t + 1$ and take a look at the complex from the top-down perspective, we would obtain something like this:

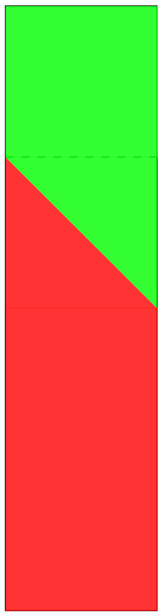

Figure 1: Top-down representation of a wedge shape in a cubical complex.

In Figure 1 the covered area is shown in red and the free area is showin in green. We only allow the object to enter the cell once it is fully free and can therefore omit the wedge shapes.

The covered planar slices are obtained by performing a union on all the covered areas by all sensors at each point in time. Covered area is simply a `Position` representing the top-left corner of a covered cell.

The cubical complex of dimensions $width \times height \times period$ is constructed by assigning the filtration value of 1 to free cells and 0 to covered cells. According to reasoning described above, a cell in the complex is marked free *iff* the corresponding space is free at two consecutive time-steps t and $t + 1$ (corresponding to 2 faces of the cube in F).

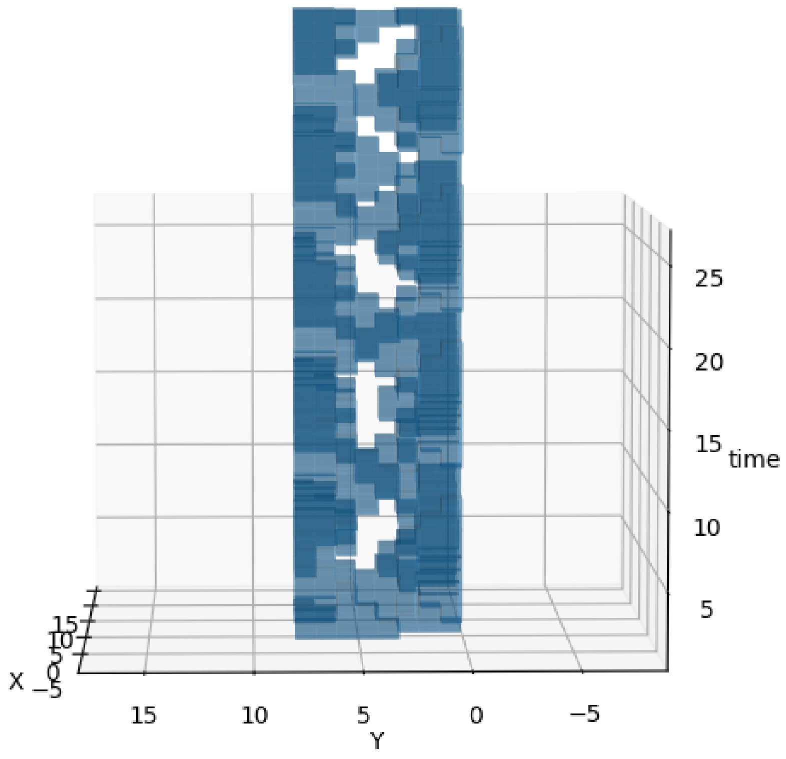

Figure 2: Free cubical complex of the example configuration (shown in Figure 5).

2.3 Computing evasion paths

Once the cubical complex F is obtained, we collapse it into a graph (1-dimensional complex) G in a way that preserves $H_1(F)$. A proper elementary collapse would have to preserve higher-dimensional homology as well, however we are only interested in $H_1(F)$. Incidentally, F shouldn't have any H_2 generators (enclosed caves) in the case of continuous sensor movement; this would imply there is something homeomorphic to a ball in the covered complex, meaning at some time t_0 one or more sensors appear out of thin air, and then disappear into a point at time $t_1 > t_0$.

The vertices of G correspond to connected components (by 4-connectivity) of F at each time interval $[t, t+1)$, $0 \leq t < p$. The idea is that an intruder with unbounded speed can move between any pair of points in a connected component of free space within a finite time interval without being detected. The vertices are labeled with (t, c) where c is the index of the connected component in this time slice, and they also hold information about the area they cover (so that we can obtain concrete `Positions` after computing a cycle).

Edges in G allow the intruder to move forward in time, so an edge from (t, c) to $(t+1, c')$ is added *iff* the connected areas c and c' intersect (and thus the intruder can move from c to c' in one time step). If we wanted to strictly follow the homology approach to this problem, these edges would need to be undirected so we end up with a simplicial complex on which to compute H_1. But in order to avoid having to filter out cycles that don't go forward in time, we kept track of the direction of edges in G.

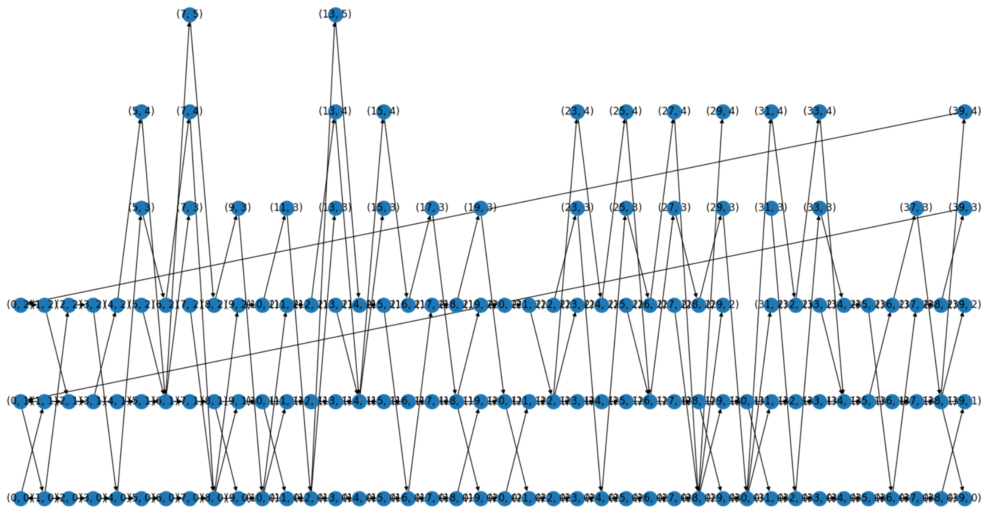

Figure 3: Evasion graph G of example sensor configuration (shown in Figure 5).

Cycles of G are then obtained by performing a depth-first search (with depth limited to p) on the graph and checking whether the current node is equal to the starting node. The above construction guarantees that each cycle in G corresponds to a valid evasion path of length p.

3 Results

We tested the algorithm with several configurations. The first configuration is trivial: a single sensor moving in the counter-clockwise direction (Figure 4).

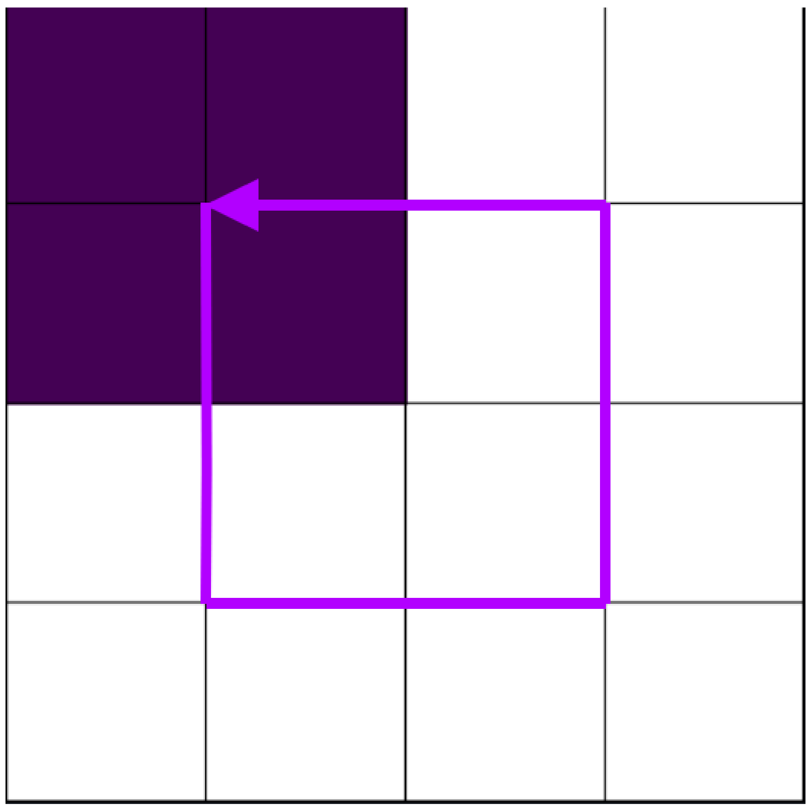

Figure 4: Counter-clockwise circular sensor in 4x4 room, period: 8

The algorithm correctly found 1 evasion path, an animation of which can be found here.

The next configuration is the example from the instructions (Figure 5).

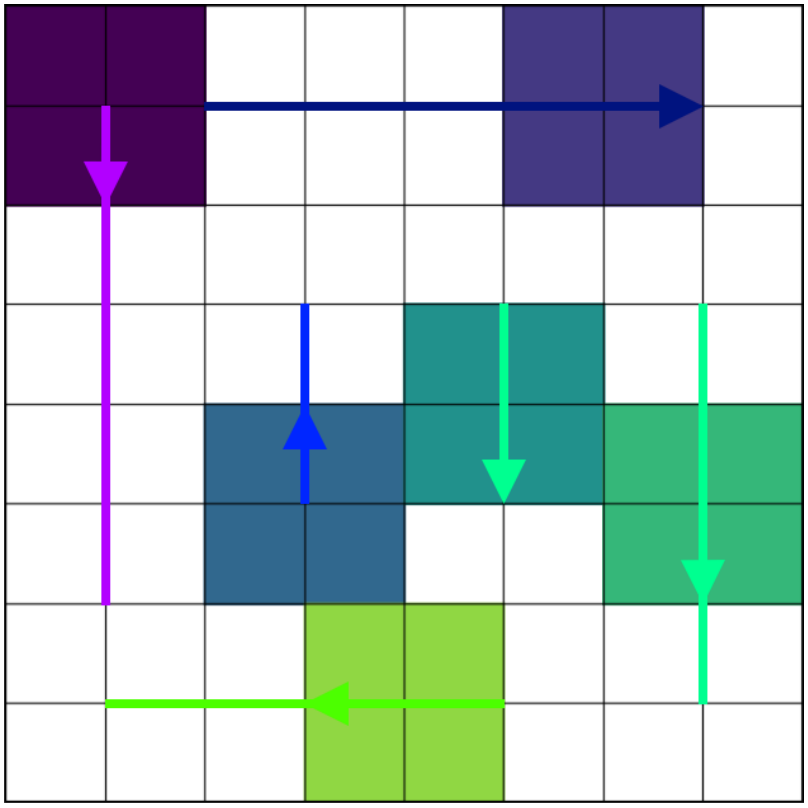

Figure 5: 6 sensors in 8x8 room, period: 40

The algorithm found 288 different evasion paths through components of free space; an animation of one path can be found here.

Lastly, we tested the algorithm on random data.

First example was on a 6×6 room with 5 sensors, going along paths with up to 5 points. The algorithm found 2 evasion paths through components of free space; an animation of one path can be found here.

Second example was on a 7×5 room with 14 sensors, going along paths with up to 5 points. The algorithm found 1 evasion path through components of free space; an animation of one path can be found here.

We also generated larger rooms with a greater amount of sensors, however that in turn made the underlying structures (`CubicalComplex`, graph) huge and the algorithm turned out to be too computationally expensive, so it did not compute the evasion paths in reasonable time. Furhtermore, if we generate a large room with a small amount of sensors, the area covered will not be substantial, thus the evasion paths will be somewhat trivial, so we will not present those results.

4 Discussion

We implemented evasion path detection by using the cubical complex structure, reducing it to a graph and finding the cycles there. This is equivalent to computing the $H_1(F, \mathbb{Z})$ on the free complex, however this way we did not need to handle edge cases as described.

We also implemented paths formed of arbitrary line segments, for the sensors to follow and an algorithm that returns areas of random sizes and sensor configurations. Lastly, we implemented animations of evasion paths as well, which help to visually confirm whether the algorithm works or not.

There is however, another limitation, considering all the assumptions made, left out to be discussed here. Since we only allow axis-aligned cuboids in the `CubicalComplex` (ie. no diagonal edges), the algorithm is not able to detect *tight* evasions. A tight evasion is an evasion, where the object trying to escape should always be positioned on a free cell, which will be covered on the next time step. This is a limitation of the `CubicalComplex` in the gudhi library, so if we were to do this, we would have to use another library for the data strucre. Furthermore, the algorithm for collapsing the complex to a graph should be changed, such that the nodes of the graph are in time slices, instead of time intervals, and edges are spanned through intervals. Since this is a limitation of the gudhi library, we decided not to implement it, as it would require too many changes.

Alternative approaches

An alternative approach would subdivide the cubical complex into a simplicial one (dividing squares into triangles and cubes into tetrahedra) and computing H_1 generators with standard TDA software (e.g. dionysus).

A completely different approach would be to represent the time-varying (2-dimensional) free space with a zig-zag filtration and derive evasion paths from zig-zag persistence information [1].

5 Bibliography

References

[1] Henry Adams and Gunnar Carlsson. "Evasion paths in mobile sensor networks". In: *The International Journal of Robotics Research* 34.1 (Nov. 2014), pp. 90–104. ISSN: 1741-3176. DOI: 10.1177/0278364914548051. URL: http://dx.doi.org/10.1177/0278364914548051.