Computational Topology - group project

The Evasion Problem

Andrej Jočić
Matija Ojo

December 28, 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 sesnors follow a predefined path and their movement is periodic. This means that, after some period, the area covered by sesnors 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 of such problem is not to cover the entire area at all times, but to cover parts of it with less 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 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 ($\mathbb{R}$), we obtain a three-dimensional (free) complex (F). We are only interested in the times up to the period, since the object simply repeats after the period. Therefore, the complex we obtain is a torus, where the first point in time is "glued" to the time at the period. One way to obtain the evasion paths would be to compute $H_1(F, \mathbb{Z})$ and check which generators represent vaild cycles/paths (ie. cycles going along the period of sensors, cycles not going in the negative direction of time). However, we decided to take a different approach and collapse the free complex to a directional 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 `Position`s, where a `Position` is point in the plane (x, y). Since we want the sensors to move along straight line segments, two consecutive `Position`s 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 `Sensor`s 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. To determine the period of the `Sensor`s, the least common multiple of lengths of all the `Path`s is computed.

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:

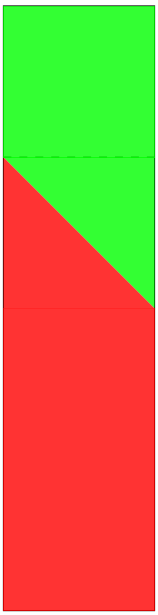

Figure 1: Top-down representation of a wedge shape in a cubical complex

In the figure 1, the red color denotes the cell is covered by a sensor, the green color denotes the cell is free (not covered). 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 free cell is only free, if it is not covered at 2 consecutive times t and $t+1$.

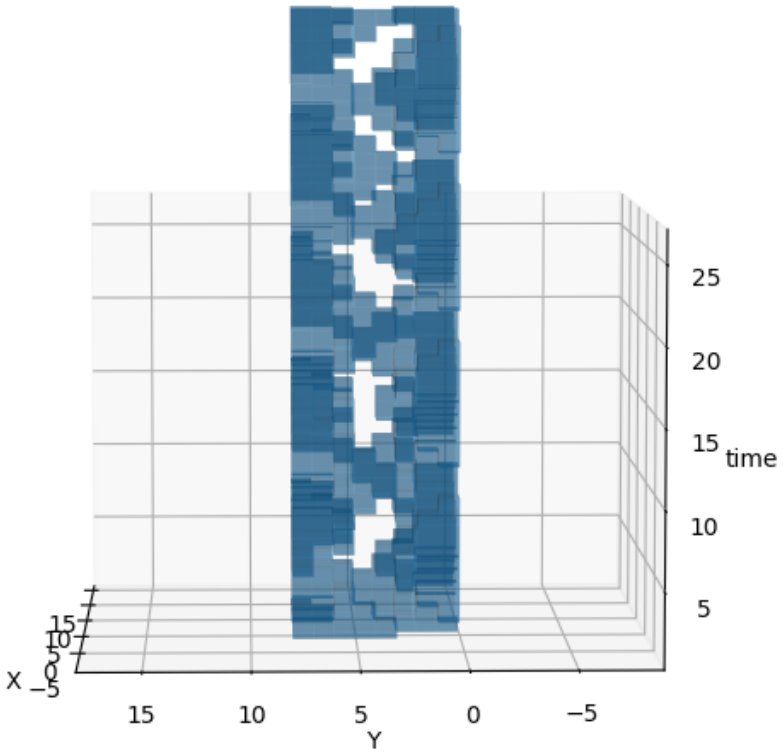

Figure 2: Free cubical complex of the example configuration

2.3 Computing evasion paths

Once the cubical complex is obtained, it is collapsed to a directed graph, preserving the homotopy. The graph is obtained by computing the connected components of the cubical complex at each time (up to period). Nodes of the graph are of form $(t, label)$, where t is the time and $label$ is the label/index of the connected component. Each node also holds the data about its `Position` in the plane, so that we can obtain actual `Positions`, after computing the cycle. The edge is added if there exists a non-empty intersection between components at time t and at time $t + 1$. Note that edges always point in the forward direction of time, therefore any cycle computed on such graph, will always point in the forward direction as well.

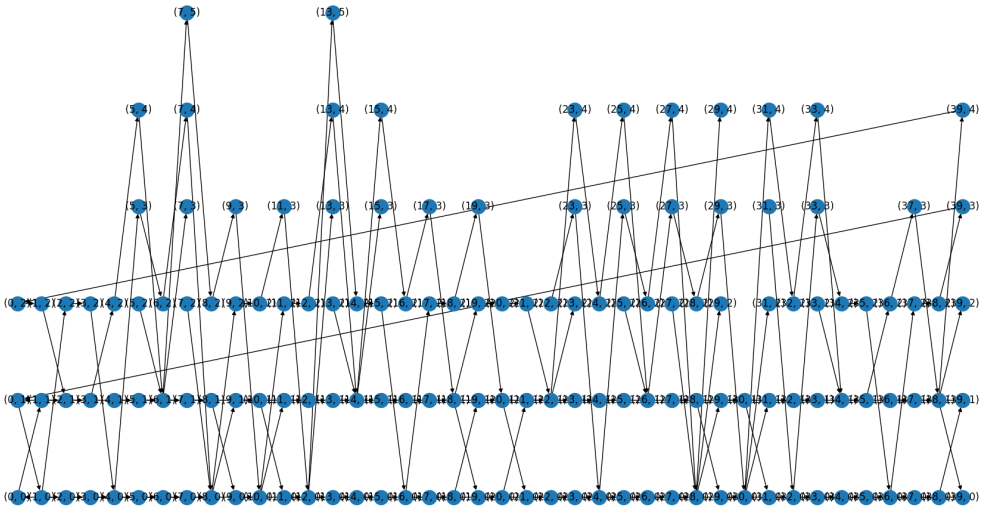

Figure 3: Graph of example configuration

Cycles are then obtained by performing a depth first search on the graph and checking whether the current node is equal to the starting node. Note that, because graph is constructed by only adding the edges in the forward direction, each such cycle will have to loop around the entire complex and will therefore have length equal the period of sensors.

3 Results

We tested the algorithm with several configurations.

The first configuration is trivial - a single sensor moving the counter clockwise direction.

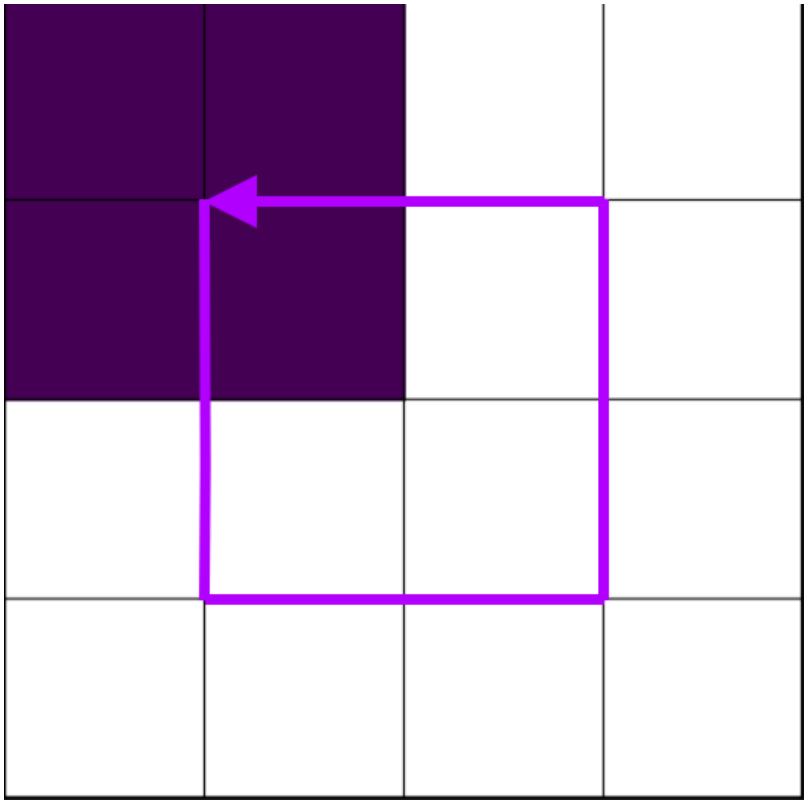

Figure 4: Counter clockwise circular sensor in 4x4 room, period: 8

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

The next configuration is the example from the instructions.

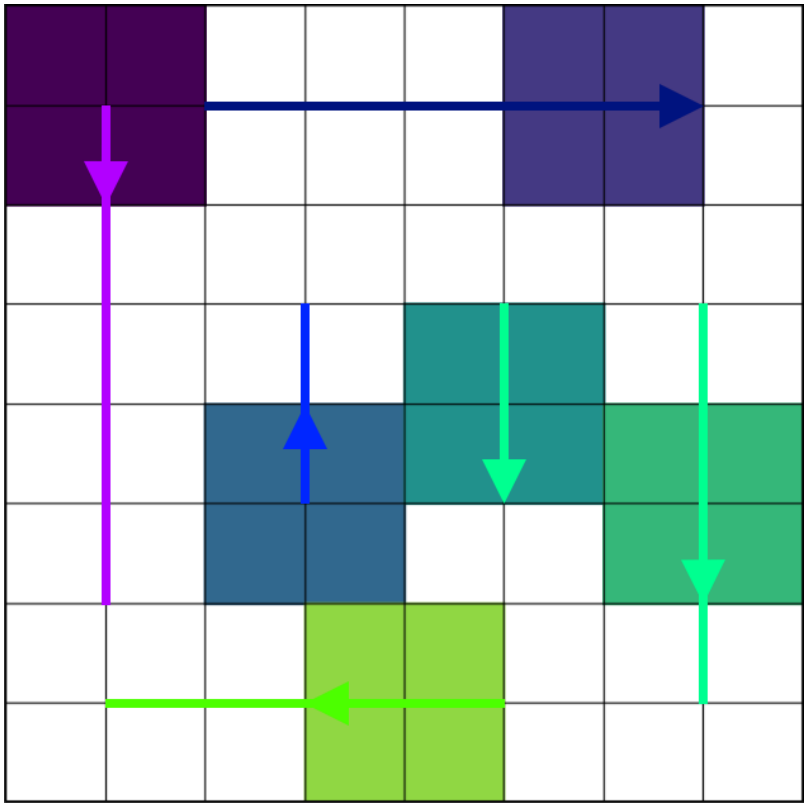

Figure 5: 6 sensors in 8x8 room, period: 40

Algorithm found 288 evasion paths, animation can be found here.

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 complex, however this way we did not need to handle edge cases as described.

We also implemented animations of evasion paths, which help to visually confirm whether the algorithm works or not.

Another approach would be to look at a square as 2 adjacent triangles (2-dimensional simplex defined by 3 points), however that would turn out to be more complicated as it would include representing grid-like data with 2 partial segments of it.

A different approach would be to use the zig-zag persistance, which would require us to use the simplicial complexes and also to specify a list of times when each simplex enters and leaves the filtration.