UCLouvain

epl

École polytechnique de Louvain

Mining Regions of Interest from trajectory data

Author: **Alexandre Dubray**
Supervisors: **Pierre Schaus, Guillaume Derval**
Reader: **Siegfried Nijssen**
Academic year 2018–2019
Master [120] in Computer Science

Abstract

The number of location-tracking devices is constantly increasing and so does the amount of captured trajectory data. There is hence an emerging need for scalable trajectory mining tools to extract knowledge from mobility data. In this work we focus on the problem of partitioning a space into meaningful regions of interest (ROIs) that are most frequently crossed by trajectories. This problem is of great importance as it constitutes the first step of the possible subsequent analysis of trajectories. Our contribution is that we formulate the task of discovering ROIs as a discrete optimization problem aiming to compress the dense regions with a given number simple parameterized shapes, such as rectangles. We give a linear program for solving this optimization problem assuming a given fixed number of ROIs. We then extend the approach to discover the best number of ROIs automatically by relying on the Minimum Description Length Principle. Our experiments on real and synthetic data show that the approach is scalable and able to retrieve ROIs of higher quality than those extracted with the well-known PopularRegion algorithm.

Acknowledgements

I would like to thank my supervisors, Pr. Pierre Schaus and Guillaume Derval, for the long discussions and exchanges of ideas that made this work what it is. This work has come a long way and many times we changed its direction, but we were able to find a good approach that offers a lot of perspective for the future, thanks to the time you dedicated advising me.

I would also like to thank Pr. Siegfried Nijssen for his support during the writing of our paper by reviewing it, restructuring it and giving us the most unexpected (but accurate) grammar rules.

Contents

Chapter 1

Introduction

Nowadays tracking devices are ubiquitous. Smartphones and wearables equipped with GPS are the most important generator of location data allowing various applications to track the position of the users. As shown by Liang et al., even without GPS information, the user location can be inferred precisely with the sensors data, for example the accelerometers[13]. Some low-consumption devices can also be located based on *Time Difference of Arrival* techniques [1]. Finally, connected objects such as bike-sharing systems offer another way to generate trajectories [5].

The analysis of this type of data raises privacy protection issues, but open the path to a lot of new applications, in particular in urban management[11]. Most of the time the methods used to analyse trajectories will not directly use the GPS data point sequences since it is not practical. Instead they will try to discover interpretable regions of interest (ROIs). This task consists of identifying regions frequently crossed by the trajectories. These regions are interesting because they are popular and represent places where users go more often. The discovery of ROIs is of practical importance as it can be instrumental for other tasks related to trajectory analysis.

In trajectory pattern mining, the goal is to find frequent sub-trajectories that appears at least a given amount of time in the database, and Giannotti et al. decomposed it as a two-step process [7]. First each trajectory is expressed as a sequence of ROIs. Then the modified database can be analysed by applying a frequent subsequence mining algorithm [3, 2] allowing extracting sequential patterns with a minimum support. Another useful application is location prediction. This task consists in, given a database of trajectories and the start of a new trajectory, predicting what will be the next location in the trajectory [16, 22, 6, 15]. In the area of urban management, systems were proposed to help manage taxis in large cities [23, 24]. Finally, the trajectory search problem can also benefit from the discovery of ROIs. This problem consists in, given a set of ROIs and a set of trajectories, discovering the trajectories with the highest spatial-density correlation to the query regions[19].

The most used algorithm for identifying ROIs is the *PopularRegion* algorithm [7] that is both easy to implement and scalable but suffer from pathological behaviour for some configurations. Intuitively, it first partitions the map by imposing a grid on the map, and then for each cell of the grid, counts the number of crossing trajectories. A cell is considered dense if its crossing counter is above a given threshold. The cells are greedily expanded to form rectangles as long as the average density of the rectangle is above the given threshold.

The contributions of this thesis are the following

- We state a formal definition for the problem of finding ROIs, which consists in finding the most parsimonious representation of all the dense cells.
- For a fixed number of K regions, we show how to discover the optimal regions using an extended

linear programming model.

- With this extended formulation, we show that we can easily relax the constraint on the shape of the regions.
- We show that the number of ROIs can be automatically selected using the minimum description length principle.
- We give a slightly modified linear program to find the set of ROIs that minimize the MDL criterion and thus automatically select the best set of ROIs.
- We evaluate qualitatively our new approach and compare it to *PopularRegion* on both real-life and synthetic data set.

This thesis is organized as follows. First we will give the basics of *Integer Programming* and describe two approaches to extract ROIs. One based on the semantic meaning of the ROIs and *PopularRegion*. We will also show how *PopularRegion* can give some unwanted results.

Chapter 3 will give a mathematical formulation of our problem and introduce a first *Integer Pogramming* model. We will also explain the limitation of this formulation.

In Chapter 4, we will extend the first formulation to address its limitations. We will decompose the ROI discovering task as a two-step process, which allows a lot more flexibility in the type of regions we can find.

We will finish in Chapter 5 with a comparison of the extended formulation and *PopularRegion* which will show that our method is scalable and indeed solve the problems of *PopularRegion*.

Chapter 2

Preliminaries and State of The Art

2.1 (Integer) Linear programming

Linear Programming (LP) is a well-known paradigm to solve optimization problems. In this framework, one can optimize linear functions subject to linear constraints. Every linear program can be represented under its canonical form.

Definition 1. *The canonical form of a linear program is expressed as follow*

$$\begin{array}{lrcl} \min\limits_{x} & c^T x & & \\ s.t. & Ax & \leq & b \\ & x & \geq & 0 \end{array}$$

Where $x \in \mathbb{R}^n$ is the vector of variables. $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ enforce the constraints of the model while c represent the coefficients of the variables.

Note that under this form, the set of feasible solutions forms a convex polytope in $\mathbb{R}^n$. The best-known methods to solve this type of optimization are

- The *Simplex* algorithm that traverses the polytope on its edge, from vertex to vertex, improving at each iteration the value of the objective. While it has been proved not to be polynomial in the worst case, it offers good performances in practice [10].
- The *Interior Points* methods which go through the interior of the polytope. Some of these methods have a worst-cast time complexity that is polynomial. For a more in-depth analysis, see [17, 20].

Integer Linear Programming (ILP) refers to a classical linear program where the variables are restricted to be integers. This means that the constraint $x \in \mathbb{Z}^n$ is added to the model shown in Definition 1. A final variant of this is *0–1 Integer Programming*, where each variable is either 0 or 1. ILP and its more restrictive variant, *0–1 ILP*, are known to be NP-complete [9]. Popular techniques to solve these kinds of problems such as *Cutting-plane method*, *Branch and Cut*, relies on the linear relaxation of the problem in order to find the (integer) optimal solution. Thus, it is important to take that into account when modelling a problem in the ILP form. Problems with weak linear relaxation will be much harder to solve.

2.2 State of the Art

We will now describe the main approaches to find interesting regions from a data set of trajectories.We first introduce algorithms based on *Stay Points* which slightly differ from our work. Then we will do a more in-depth analysis of the *PopularRegion* algorithm, which is at the basis of our work. But first, let us introduce the notion of trajectories, common to all algorithms.

Definition 2 (Trajectory)**.** *A trajectory s is an ordered sequence of points in $\mathbb{R}^2$, each one associated with an increasing timestamp.*

That is we have $s = \langle (s_1, t_1), \ldots, (s_n, t_n) \rangle$, where $\forall i \in [1, \ldots, n] : s_i \in \mathbb{R}^2, t_i \in \mathbb{R}^+$ and $\forall i, j \mid 1 \leq i < j \leq n : t_i \leq t_j$. The size of the trajectory, denoted $|s|$, is the length of the sequence.

This definition is based on GPS data since our real word data set contains trajectories from taxis. However, a different definition, for other use cases, might be applied and do not impact the contribution of this work. Moreover, we do not impose that the timestamps increase strictly. In some use cases, it might be the case that the data are formatted in such way that multiple consecutive data point have the same timestamp (e.g. a sensor that only record every hour the position), but it can never decrease.

2.2.1 Stay Points

Stay Points (SP) are geographic regions where one or more moving objects stay for a significant amount of time. The distance span by the region and the amount of time needed to be considered as a SP are both parameters of the detection algorithm. SP detection algorithm was first proposed in [12] to discover regions that carry a *semantic* meaning. Algorithm 1 depicts their approach.

Algorithm 1: Stay Point detection (adapted from [12])

Input: A trajectory $\boldsymbol{T}$, a minimum distance threshold $\boldsymbol{\delta}$ and a time threshold $\boldsymbol{\tau}$
Output: A set of Stay Point $\boldsymbol{S}$

```
1  S ← {}
2  i ← 0
3  while i < |s| do
4      for j = i + 1 until |s| do
5          dist ← distance(s_i, s_j)
6          if dist > δ then
7              ΔT ← t_j − t_i
8              if ΔT > τ then
9                  SP ← Mean({s_k | i ≤ k ≤ j})
10                 S ← S ∪ {SP}
11             end
12             i ← j
13             break
14         end
15     end
16 end
17 return S
```

There are two conditions needed to create a SP. First, the current considered data point must be far enough from the previous one (line 5). Moreover, the time needed to go from the previous

data point to the current should be high (line 7). If this is the case, a SP is created. It will be the characterization (i.e. the mean) of multiple data point that are close to each other and for which, the next point in the trajectory is far away as well geographically than temporally. In other words, the algorithm will try to cluster data points spatially and temporally. Note that this algorithm is run on only one trajectory. By running it on each trajectory of the database, we obtain a set of stay points for each input in the database. This means that some SP found might be close enough to be considered the same point of interest. Although it is not a problem for Li et al. since their goal is to find similarities between users, when the final objective is to extract regions of interest, it might be a problem.

In [23, 24] Yuan et al. extended this algorithm by adding filtering and clustering of the detected stay point. The argument for the filtering is that each detected SP might not be relevant. In their work, they study the case of trajectories from taxis. An example of wrongly detected SP is if a taxis is stuck in a traffic jam. It will indeed be detected as a SP by the algorithm, but is not relevant (i.e. has no particular semantic). They address the problem of detecting several times the same SP in different trajectories by running a spatial cluster algorithm on all the SP found by the algorithm. Since for each trajectory, the found SPs represent a fraction of the total number of real SPs, by clustering them they can increase the precision of their algorithm, especially when the SPs have non-regular shape.

2.2.2 PopularRegion algorithm

While the previous approach relies on the time spent in a small area to find semantically interesting region, *PopularRegion* is a density-based algorithm [7]. First, the algorithm applies a grid division on the map. Let us denote by $\mathcal{G}$ this grid, and c_{ij} the cell in the i-th row and j-th column. The key concept in this algorithm is the notion of density of a cell.

Definition 3 (Density of a cell)**.** *Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a set of trajectories and $\mathcal{G}$ be a grid of size $R \times C$. The density of the cell c_{ij} ($1 \leq i \leq R$, $1 \leq j \leq C$), denoted density(c_{ij}), is defined as*

$$\textit{density}(c_{ij}) = \mid \{T \in \mathcal{T} \mid T \cap c_{ij} \neq \emptyset\} \mid$$

Where $T \cap c_{ij}$ is the set of points in T that are included in the cell c_{ij}.

A cell c_{ij} is dense with respect to a minimum density threshold δ if density$(c_{ij}) \geq \delta$. We will denote $\mathcal{G}^* = \{c \in \mathcal{G} \mid \text{density}(c) \geq \delta\}$ the set of dense cells. This division of the map in a grid with dense cells already gives a first idea of *Region of Interest* (ROI). Indeed, for a given minimum density threshold, the dense cells will represent a subset of the original space that is most visited. Figure 2.1 shows an example of set of dense cells on the city of Porto. We can clearly identify some potential

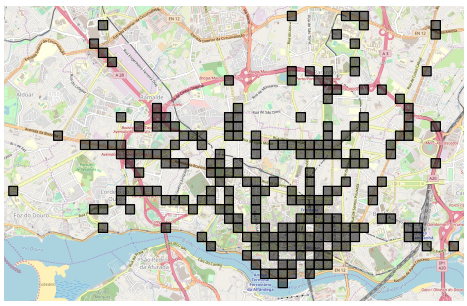


Figure 2.1: All dense cells for a 100×100 grid on Porto

ROI just by looking at the distribution of the dense cells. For example, we can expect to find a large region at the bottom of the map, which is the centre of the city.

The *PopularRegions* algorithm is depicted in Algorithm 2. The algorithm is designed to find a set of *non-overlapping* rectangular regions over the grid. Intuitively, this is an iterative algorithm that considers each dense cell one by one, in descending order of density. Then, each cell is extended as much as possible, while conserving the rectangular shape.

Algorithm 2: PopularRegions (adapted from [7])

Input: A grid $\mathcal{G}$, a minimum density threshold δ
Output: A set R of rectangular regions over $\mathcal{G}$

$R \leftarrow \emptyset$
foreach $c_{ij} \in \mathcal{G}$ **do** $used(c_{ij}) = \text{false}$

foreach $c_{ij} \in \mathcal{G}^*$ *in descending order of density* **do**
  if $\neg used(c_{ij})$ **then**
    $r \leftarrow \{c_{ij}\}$
    repeat
      foreach $dir \in \{left, right, up, down\}$ **do**
        $r_{dir} \leftarrow$ r extended on direction dir
      end
      $ext \leftarrow \{dir | r_{dir} \subseteq \mathcal{G} \wedge avg\_density(r_{dir}) \geq \delta \wedge \exists\, c \in (r_{dir} \setminus r) \mid \text{density}(c) \geq \delta \wedge \forall c \in r_{dir} : \neg used(c)\}$
      if $ext \neq \emptyset$ **then**
        $dir \leftarrow \arg\max_{d \in ext} avg\_density(r_d)$
        $r \leftarrow r_{dir}$
      end
    until $ext = \emptyset$
    foreach $c \in r$ **do** $used(c) = true$

    $R \leftarrow R \cup \{r\}$
  end
end
return R

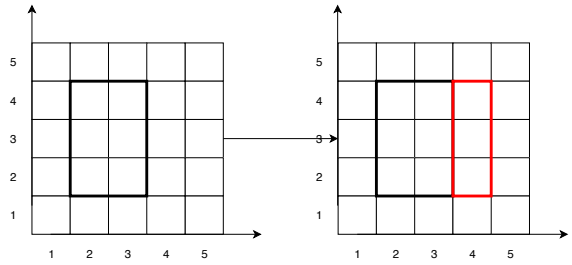


Figure 2.2: Example of extension on the right side of a region.

We will now describe more precisely the extension process. Extending a rectangle in one direction

is done by conserving the rectangular shaped and not modifying any other direction. This process is illustrated on a small example in Figure 2.2. On the left we have the initial rectangle in black. An extension to the right is considered in red, on the right figure. Every extension is not valid. In line 11 of Algorithm 2, Giannotti, et al., enforce some constraints on the type of regions they will produce. For an extension to be valid, it needs to respect those conditions :

- The region resulting from the extension must be in the grid.
- The region resulting from the extension need to be dense *in average*.
- The extension must contain a dense cell.
- The extension must not contain a cell that is already part of another ROI.

They key point here is that the algorithm will only maintain an average density for the whole region. Such mechanism is needed to have a better generalization, and find better ROIs. For example, let us consider the grid shown in Figure 2.3. As for every example from now on, the dense cells are shown in grey. On this simple example, it is clear that the whole centre (i.e. the eight dense cells and the cell $(3, 3)$) can be considered as a ROI. However, if we do not allow non-dense cell to be taken in the extension process, the algorithm would find four regions instead of one.

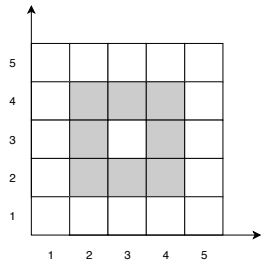


Figure 2.3: Example of a ROI with a non-dense cell inside of it

PopularRegion as shown to give good results and have been widely used. However, as it is shown by Gorawski and Jureczek in [8], it is easy to create examples where the extensions process end up with very large regions. This might be a problem for the underlying application. In [7], the goal is to use the regions in a trajectory pattern setting. Too large regions do not convey enough information and pattern such as $Airport \rightarrow Downtown$ are not really helpful. An example of output is shown in Figure 2.4. Although it is true that the central part of the city is highly visited, a more fine-grained division might be preferred for subsequent analysis.

Figure 2.5 shows the output of the method proposed in this work for the same configuration. It can be seen that we find a more precise decomposition of the centre of the city, but we avoid isolated cells of the outside. This is a design choice that will be discussed in the following parts of this thesis.

A second critic that can be made is that the output of the algorithm is ill-defined; there is no clear characterization of an objective function that is minimized. Although we are considering an unsupervised task, it might be helpful to be able to compare multiple solutions and chose one based on a criterion, which may be user-defined.

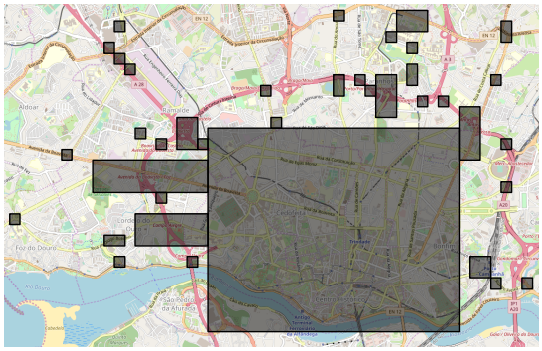


Figure 2.4: Output of PopularRegion for the grid shown in Figure 2.1 and a density threshold of $0.05|\mathcal{T}|$

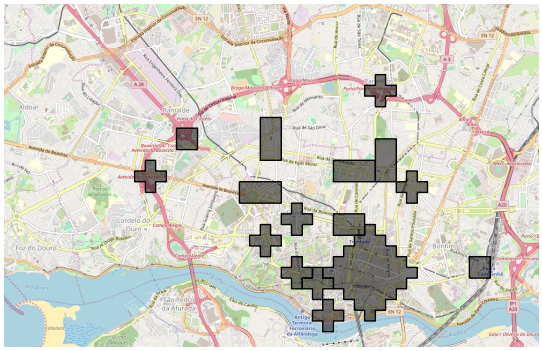


Figure 2.5: Output of our method for the grid shown in Figure 2.1 and a density threshold of $0.05|\mathcal{T}|$

Chapter 3

A Mathematical Formulation of the Problem

To address the issues of *PopularRegion* explained in the previous section, we will change the way the ROIs are found. First we would like the problem to be well defined, in a mathematical way. Moreover we want to avoid having big regions that covers large areas with non-dense cells. To do so our objective function will penalize this type of region as well as dense cells not covered since we want to summarize them.

3.1 An exact model for encoding the ROI on a grid

As done by Giannotti et al., we will be considering only rectangular, isooriented (i.e. with sides parallel to the axis of the grid) and non-overlapping ROIs. In Chapter 4 we will be able to consider other type of shape, for example circular regions as shown in Figure 2.5.

Let us assume that the algorithm returns a set $\mathcal{R}$ of non-overlapping regions. By abuse of notation, we will also denote $\mathcal{R}$ the set of cell cover by these regions. Since these regions has been selected by the algorithm as the interesting ones to return, it is fair to assume that they represent the regions that are popular with respect to our minimum density threshold. Thus our model will assume that every cell in $\mathcal{R}$ is dense, while every other is not. Of course, such a model will make some errors : the non-dense cells covered by a rectangle and the dense cells not covered by any rectangles.

An example of model is shown in Figure 3.1. This model has two rectangles and makes four errors : the cells $(3, 4)$ and $(7, 6)$ are non-dense but covered by a rectangle while the cells $(2, 6)$ and $(8, 7)$ are dense but not covered by any rectangle. Definition 4 formalize this principle.

Definition 4 (Model)**.** *Let $\mathcal{R} = \{r_1, \ldots, r_K\}$ be a model composed of K isooriented, non-overlapping, rectangles on a grid $\mathcal{G}$. The errors made by the model can be expressed as follow*

$$error^+_{\mathcal{R}} = \{c \in \mathcal{G} \mid \textit{density}(c) \geq \delta \wedge c \notin \mathcal{R}\}$$
$$error^-_{\mathcal{R}} = \{c \in \mathcal{G} \mid \textit{density}(c) < \delta \wedge c \in \mathcal{R}\}$$

In our optimization problem, we thus wish to find the set of rectangles $\mathcal{R}_{opt}$ that minimize the errors.

$$\mathcal{R}_{opt} = \arg\min_{\mathcal{R}} |error^+_{\mathcal{R}}| + |error^-_{\mathcal{R}}|$$

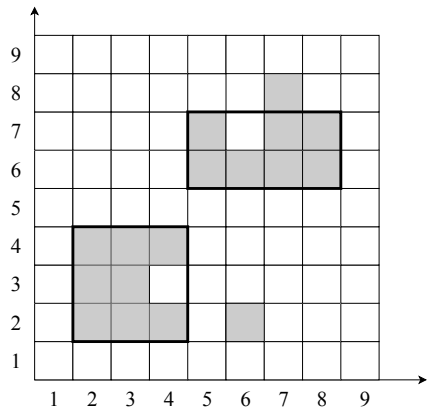


Figure 3.1: An example of model. The dense cells are shown in grey, and the ROIs are shown by the bolder lines.

3.2 A naive MIP formulation

3.2.1 The model

We will work on a $R \times C$ grid (i.e. R rows and C columns). Since it is clear from the context, we will not write the domain of each variable for the clarity. Thus we will express, for example, $\forall c_{ij} \in \mathcal{G}$ simply as $\forall c_{ij}$.

We will associate to each cell a constant indicating if the cell is dense or not.

$$\forall c_{ij} : D_{ij} = \begin{cases} 1 & \text{iif density}(c_{ij}) \geq \delta \\ 0 & \text{otherwise} \end{cases}$$

Each rectangle will be represented by four variables, that is, $R_k = (r_k^1, c_k^1, r_k^2, c_k^2)$. We also need a variable that tells us if a particular rectangle R_k covers a cell c_{ij} (by abuse of notation, we also denote R_k the set of cells covered by R_k)

$$\forall c_{ij}, R_k : cov_{ij}^k = \begin{cases} 1 & \text{iif } c_{ij} \in R_k \\ 0 & \text{otherwise} \end{cases}$$

We can now define the variables that will represent the errors made by the model.

$$\forall c_{ij} : err_{ij}^- = \begin{cases} 1 & \text{iif } \sum_{k=1}^K cov_{ij}^k = 1 \wedge D_{ij} = 0 \\ 0 & \text{otherwise} \end{cases} \qquad \forall c_{ij} : err_{ij}^+ = \begin{cases} 1 & \text{iif } \sum_{k=1}^K cov_{ij}^k = 0 \wedge D_{ij} = 1 \\ 0 & \text{otherwise} \end{cases}$$

Note that for a given cell, we have either err_{ij}^- or err_{ij}^+ set, but never both at the same time. With the non-overlapping constraint, we have that at most one rectangle will cover a given cell, thus the sum of the cov_{ij}^k cannot exceed 1. We can now write our IP model

$$\text{minimize} \quad \sum_{c_{ij} \in \mathcal{G}} err_{ij}^{+} + err_{ij}^{-} \tag{3.1}$$

$$\text{subject to} \quad \forall R_k : 1 \leq r_k^1 \leq r_k^2 \leq R \tag{3.2}$$

$$\forall R_k : 1 \leq c_k^1 \leq c_k^2 \leq C \tag{3.3}$$

$$\forall c_{ij}, R_k : -R(1 - cov_{ij}^k) + r_k^1 \leq i \leq r_k^2 + R(1 - cov_{ij}^k) \tag{3.4}$$

$$\forall c_{ij}, R_k : -C(1 - cov_{ij}^k) + c_k^1 \leq j \leq c_k^2 + C(1 - cov_{ij}^k) \tag{3.5}$$

$$\forall c_{ij}, R_k : -R\ rlim_{ijk}^l < r_k^l - i \leq R(1 - rlim_{ijk}^l) \text{ for } l \in \{1, 2\} \tag{3.6}$$

$$\forall c_{ij}, R_k : -C\ clim_{ijk}^l < c_k^l - j \leq C(1 - clim_{ijk}^l) \text{ for } l \in \{1, 2\} \tag{3.7}$$

$$\forall c_{ij}, R_k : 4 - 4(4 - \sum_{l=1}^{2} rlim_{ijk}^l + clim_{ijk}^l) \leq 4cov_{ij}^k \leq \sum_{l=1}^{2} rlim_{ijk}^l + clim_{ijk}^l \tag{3.8}$$

$$\forall c_{ij} : \sum_{k=1}^{K} cov_{ij}^k \leq 1 \tag{3.9}$$

$$\forall c_{ij} : 2err_{ij}^{+} \leq D_{ij} + (1 - \sum_{k1}^{K} cov_{ij}^k) \leq err_{ij}^{+} + 1 \tag{3.10}$$

$$\forall c_{ij} : 2err_{ij}^{-} \leq (1 - D_{ij}) + \sum_{k=1}^{K} cov_{ij}^k \leq err_{ij}^{-} + 1 \tag{3.11}$$

The constraints (3.4) to (3.8) model the relation of the cov_{ij}^k variables. Constraints (3.4) and (3.5) ensure that if $cov_{ij}^k = 1$, then $c_{ij} \in R_k$. On the other hand, constraints (3.6) to (3.8) ensure that if $c_{ij} \in R_k$, then $cov_{ij}^k = 1$. To do so, we needed to add additional variables. Indeed, to determine if a cell is inside a rectangle, we need to look at the projection of the cell and the rectangle on each of the axis of the grid. If on both axes, the projection of the cell is included in the projection of the rectangle, then the cell is included in the rectangle. This is what the $rlim_{ijk}^l$ and $clim_{ijk}^l$ check. For example

$$\forall c_{ij}, R_k : rlim_{ijk}^1 = \begin{cases} 1 & \text{iif } r_k^1 \leq i \\ 0 & \text{otherwise} \end{cases} \qquad \forall c_{ij}, R_k : rlim_{ijk}^2 = \begin{cases} 1 & \text{iif } i \leq r_k^2 \\ 0 & \text{otherwise} \end{cases}$$

These variables indicate if, when projecting on the row axis, the cell is included or not in the rectangle. Then, constraint (3.8) make the link between the cov_{ij}^k and these four variables. That is, $cov_{ij}^k = 1$ if and only if every other variable is 1.

3.2.2 Issues of the model

While this model perfectly reflects the problem we want to solve, it faces a major drawback : it is not tractable. *Integer Programming* is NP-complete and if without a model carefully designed, we end up with a model that cannot be solved in a reasonable amount of time.

The first problem is this formulation is that it is highly symmetrical and such type of problems are known to be hard to solve [14]. Indeed if we invert two rectangles in a feasible solution, we have another feasible solution. Note that we already broke a source of symmetry by imposing an order on the coordinate of the rectangles.

Moreover this formulation has a weak linear relaxation due to the Big-M constraints, which are known to weaken the linear bounds [21]. It means that when the solver relaxes the integrity constraint

on the variables, the set of feasible solutions will be much larger than the initial set of feasible solution. Thus it is more likely that the relaxed solution will be far away from the actual optimum, which will make the solving process slower.

To address this problem, we will change the formulation of the problem using an *extended formulation* [4] since they avoid symmetries in the model and remove the need for Big-M constraints. In this formulation, the idea is to generate all possible ROI candidates and associate each of them with a binary decision variable. Then we formulate an objective function and constraints over these candidates. Although we will need a large number of variables, in our case it is still polynomial since there are at most R^2C^2 variables in a $R \times C$ grid. If this is not the case, then we can generate the candidates in a column generation fashion.

Chapter 4

Extended IP formulation

Under our new formulation, the process to discover ROIs from trajectory data will be split in two. First we will generate a set of candidates, and then solve an optimization problem that will select the best candidates. We will see that for the candidate generation, enumerating them all would be too costly and thus we need to optimize it. Since we need the objective function to deduce properties on the candidates, we will first give the new formulation before explaining the candidate generation process.

4.1 New IP model

Let us denote $\mathcal{S}$ the set of all candidates. We first need to rewrite our error criterion in terms of these candidates.

Theorem 1. *Let $\mathcal{S}$ be the set of all ROI candidate. Then we can rewrite the objective of the optimization model as follows.*

$$\min_{\substack{\mathcal{C}\subseteq\mathcal{S}\\|\mathcal{C}|=K}} |err_{\mathcal{C}}^{+}| + |err_{\mathcal{C}}^{-}| \Leftrightarrow \min_{\substack{\mathcal{C}\subseteq\mathcal{S}\\|\mathcal{C}|=K}} \sum_{s_i\in\mathcal{C}} u_i - d_i$$

where d_i (resp. u_i) represent the number of dense (resp. non-dense) cells covered by the candidate s_i

Proof. Let $\mathcal{D}$ be the set of the dense cells in the grid $\mathcal{G}$. For a given $\mathcal{C} \subseteq \mathcal{S}$, we have

$$\begin{aligned}
|err_{\mathcal{C}}^{+}| + |err_{\mathcal{C}}^{-}| &= \sum_{c_{ij}\in\mathcal{G}} err_{ij}^{+} + err_{ij}^{-} \\
&= \sum_{c_{ij}\in\mathcal{G}} err_{ij}^{+} + \sum_{c_{ij}\in\mathcal{G}} err_{ij}^{-} \\
&= |\mathcal{D}| - \sum_{s_i\in\mathcal{C}} d_i + \sum_{s_i\in\mathcal{C}} u_i \\
&= |\mathcal{D}| + \sum_{s_i\in\mathcal{C}} u_i - d_i
\end{aligned}$$

□

If we assign a decision variable $x_i \in \{0, 1\}$ to each candidate s_i, we can rewrite our optimization model as follows.

$$\text{minimize} \quad \sum_{s_i \in \mathcal{S}} x_i \cdot (u_i - d_i) \tag{4.1}$$

$$\text{subject to} \quad \sum_{s_i \in \mathcal{S}} x_i \leq K \tag{4.2}$$

$$\forall c_{ij} : \sum_{s_i \in \mathcal{S} | c_{ij} \in s_i} x_i \leq 1 \tag{4.3}$$

The constraints (4.2) limit the number of selected candidates and the constraint (4.3) ensure that there is no overlap between the selected ROIs. It can be seen that this model neither suffer from symmetry nor weak relaxation due to Big-M constraints.

We can also see from this formulation that adding other shape for the ROIs is straightforward. Indeed each candidate only needs a weight in order to be considered in the objective function. For example, circular regions are natural candidates for the ROIs. A circular region $c = (row, col, radius)$ is uniquely defined by its centre and its radius. It covers the cells

$$c = \{c_{ij} \in \mathcal{G} \mid |i - row| + |j - col| \leq radius\}$$

Assuming the Manhattan distance.

4.1.1 A parameter-free model

Finally, we would like our model to automatically select the optimal K because in some case fixing this value can be an arbitrary decision. We can use the *Minimum Description Length* (MDL) [18] to determine the size of the model in a principle manner. The MDL principle trade-off the length of the model (the K) and the error made by the model. More precisely, let us assume that we have a set of models (also called *hypothesis*) $\mathcal{H}$ to select over a data set D. Then this principle tells us to choose the one that minimizes

$$L(D, H) = L(H) + L(D \mid H)$$

Where $L(H)$ is the number of bits to encode the model and $L(D \mid H)$ is the number of bits needed to encode the deviation from the data D when they are encoded with the model H. Using bits with our model is not practical since all values are integers. However, assuming that every integer is encoded using the same number of bits, we can directly use the MDL principle on integers. The optimization model presented in the previous section is composed of

- A grid where each cell needs two integers to be encoded : its row and its column.
- A set of selected ROIs. In this work each ROI is either a circle or a rectangle. Each rectangle needs two cells to be encoded (its lower left and upper right corner), that is, four integers. Each circle needs three integers : its centre and its radius.
- A set of wrongly classified cells, the error of the model

Thus for a set of selected candidates $\mathcal{S}$ and a grid $\mathcal{G}$, we have

$$L(\mathcal{S}) = \sum_{s_i \in \mathcal{S}} length(s_i)$$

$$L(\mathcal{G} \mid \mathcal{S}) = 2 \cdot (D + \sum_{s_i \in \mathcal{S}} (u_i - d_i))$$

where $length(s_i)$ is the number of integers needed to encode the region s_i (e.g. 4 for a rectangle and 3 for a circle) and the value

$$D + \sum_{s_i \in \mathcal{S}} (u_i - d_i)$$

Counts the number of errors made by the model.

Now we can easily find the set of ROIs that yields the minimum description length using the following optimization model

$$\text{minimize} \quad \sum_{s_i \in \mathcal{S}} x_i \cdot (2(u_i - d_i) + length(s_i)) \tag{4.4}$$

$$\text{subject to} \quad \sum_{s_i \in \mathcal{S} | s_{ij} \in s_i} x_i \leq 1 \tag{4.5}$$

with (4.4) minimizing the MDL criterion and (4.5) enforcing the non-overlap between the ROIs. Note that we do not directly minimize $L(\mathcal{G} \mid \mathcal{S})$ but we removed the $2 \cdot D$ since it does not impact the optimization.

4.2 Generation of rectangular candidates

Now that we have expressed the objective function in terms of the candidates, we can reason about it to avoid enumerating candidates that will never be selected. In this section, we will first explain the general framework to generate rectangles. Then we will see some properties on the rectangles of the optimal solution that allows us to reduce the number of explored candidates. Finally, we will compare the brute force approach to two different optimization. In this work, we have not optimized the generation process for the circular regions.

4.2.1 Generating rectangles by their centre

First, we need to note that every rectangle has a *centre*.

Definition 5 (Centre of a rectangle)**.** *The centre of a rectangle $r = (r^1, c^1, r^2, c^2)$ is a set of contiguous cells $C \subseteq r$ such that the following two inequalities hold*

$$\min_{c \in C} d(c, r^1) = \min_{c \in C} d(c, r^2)$$
$$\min_{c \in C} d(c, c^1) = \min_{c \in C} d(c, c^2)$$

where d is the function computing the Manhattan distance

For example, if a rectangle has an odd height h, then its centre will be located on the row $\frac{h+1}{2}$. On the contrary, if its height is even, then it will be located on rows $\frac{h}{2}$ and $\frac{h}{2} + 1$. There are four types of rectangle if we class them by their centre : i) The centre is a single cell ii) The centre is two cells on the same row iii) The centre is two cells, on the same column iv) The centre is four cells. These sets are distinct, this means that we can reason about them separately and combine the result for each centre to get all the candidates. We will now explain the methods used to generate the rectangles.

Rectangle centred in one cell

The simplest case is when the rectangle is centred in one cell. The idea behind the algorithm is to start in the centre of the rectangles (here one cell), and to extend it in width and height to form all rectangles centred in that cell. Algorithm 3 shows the algorithm to generate all one-cell centred rectangle on a grid $\mathcal{G}$.

Algorithm 3: Generating all rectangles centred in one cell

Input: A grid $\mathcal{G}$ of size $R \times C$
Output: A set of ROI candidates $\boldsymbol{S}$

```
1  selected ← {}
2  for row = 1 to R do
3      for col=1 to C do
4          maxHeight ← min(row − 1, R − row)
5          maxWidth ← min(col − 1, C − col)
6          for height = 0 to maxHeight do
7              for width = 0 to maxWidth do
8                  r ← (row − height, col − width, row + height, col + width)
9                  if isValid(r) then
10                     selected ← select ∪ {r}
11                 end
12             end
13         end
14     end
15 end
16 return selected
```

We can note at line 9 that we do not include every considered rectangle in the set of candidates. The `isValid` function will filter out candidates that we know will never be selected by the optimal solution. For now, we only discard candidates that cover more non-dense than dense cells. Then, for a given cell, we cannot extend it beyond the limit of the grid. If we want to extend the cell at row r, we cannot go below 1 and above R. Thus, the maximum length before reaching the first row is $r - 1$. On the other side, if we extend more than $R - r$, we would have a row superior to R be included in the rectangle. The maximum height is thus the minimum of these two values. The same reasoning applies on the columns, and if we consider all possible heights and widths, we can generate all rectangles centred in a given cell.

Rectangle Centred in Multiple Cells

For generating rectangles centred in two cells, the algorithm is almost the same. The only thing that changes is the computation of the bounds, and the rectangle itself. These differences are shown in Algorithm 4 for a centre of two cells on the same row. First, at line 3, we replaced `to` by `until`. Indeed since we consider the cell at the column col to be the leftmost one, it cannot be on the column C otherwise the other cell of the centre would be outside the grid. At line 5 we changed the computation of the upper bound on the width. Definition 5 gives us the following condition

$$maxWidth = \min(col - 1, C - col, (col + 1) - 1, C - (col + 1))$$

Algorithm 4: Generating all rectangles centred in two cells on the same row

Input: A grid $\mathcal{G}$ of size $R \times C$
Output: A set of ROI candidates $\boldsymbol{S}$

```
selected ← {}
for row = 1 to R do
    for col=1  until C do
        maxHeight ← min(row − 1, R − row)
        maxWidth ← min(col − 1, C − (col + 1))
        for height = 0 to maxHeight do
            for width = 0 to maxWidth do
                r ← (row − height, col − width, row + height, (col + 1) + width)
                if isValid(r) then
                    selected ← select ∪ {r}
                end
            end
        end
    end
end
return selected
```

However, we always have $col - 1 < (col + 1) - 1 = col$ and $C - (col + 1) = C - col - 1 < C - col$. Thus we can reduce it to the computation shown in the algorithm. Finally, the upper right corner of the rectangle needs to be computed using the appropriate column, that is, $col + 1$.

Since the algorithm for computing the rectangles centred in two cells on the same column is exactly the same except that the changes noted above are applied on the rows, we will not include it.

Finally, rectangles that are centred in four cells are a combination of the ones centred in two cells. The Algorithm 5 shows the process. We simply applied both changes from the two cells on the same row and column.

By running these fours algorithms on the grid $\mathcal{G}$, and combining the results, we generate all rectangles that cover more dense cells than non-dense. Note that with this approach, it is easy to incorporate constraints on the rectangles. Indeed all we have to do is modify the `isValid` function. For example, in order to have more homogeneous regions, we have imposed the following ratio constraint on the rectangles

$$height \leq 2 * width \wedge width \leq 2 * height$$

In the worst case, the whole grid is dense. Thus, it is theoretically impossible to have a better complexity than $\mathcal{O}(R^2C^2)$ for the generation process. In practice, most of the time the grid contains a low number of dense cells. This means that most of the rectangles that are generated will be filtered out.

4.2.2 Optimizing the generation process

We will now explore how we can optimize this generation process. First, we will show how to find better boundaries on the height and width of the centres. We will first explain for centre composed of a single cell, and then use these results to extend it to centre composed of multiple cells. Finally, we will show that during the extension, it is possible to stop considering some width.

Algorithm 5: Generating all rectangles centred in four cells

Input: A grid $\mathcal{G}$ of size $R \times C$
Output: A set of ROI candidates $\boldsymbol{S}$

```
selected ← {}
for row = 1 until R do
    for col=1 until C do
        maxHeight ← min(row − 1, R − (row + 1))
        maxWidth ← min(col − 1, C − (col + 1))
        for height = 0 to maxHeight do
            for width = 0 to maxWidth do
                r ← (row − height, col − width, (row + 1) + height, (col + 1) + width)
                if isValid(r) then
                    selected ← select ∪ {r}
                end
            end
        end
    end
end
return selected
```

Lowering the Maximum Width and Height

In the algorithms presented in the previous section, one way to reduce the number of explored candidates is to reduce the maximum height and width for a given centre. For now, we limit the rectangles to the boundary of the grid. However, for some centre we can reduce these boundaries. Let us consider the grid shown in Figure 4.1.

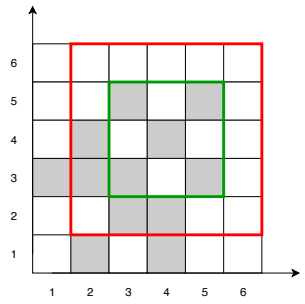


Figure 4.1: Example of upper bound that can be reduced

We have shown in red the boundaries on the height and width of the rectangles centred in the cell $(4, 4)$. We can see that all the cells on the sixth row (included in the boundaries) are not dense. This means that, for a height of 2, every generated rectangle centred in $(4, 4)$ will have a side full of non-dense cells, and we have the following property

Theorem 2. *Let $\mathcal{R} = \{R_1, \ldots, R_K\}$ be a model. If there exists $R_k = (r^1, c^1, r^2, c^2) \in \mathcal{R}$ such that one of its sides does not cover any dense cell, then $\mathcal{R}$ is not optimal with respect to the description length.*

Proof. Let us assume that, without loss of generality, the left side of r do not cover any dense cell. Let us consider $r' = (r^1, c^1 + 1, r^2, c^2)$, the sub-rectangle formed by removing the left side or r, and a new model $\mathcal{R}'$ with

$$\mathcal{R}' = (R \setminus \{R_k\}) \cup \{r'\}$$

If we denote d_k the number of dense cells covered by R_k, and u_k the number of non-dense cells covered by R_k, we have

$$\begin{aligned}\mathcal{L}(\mathcal{R}') &= \mathcal{L}(\mathcal{R}) - (4 - 2d_k + 2u_k) + (4 - 2d_k + 2(u_k - (r^2 - r^1 + 1))) \\ &= \mathcal{L}(\mathcal{R}) - 2(r^2 - r^1 + 1) \\ &< \mathcal{L}(\mathcal{R})\end{aligned}$$

Since we have found a model that is valid and has a length inferior to $\mathcal{M}$, it was not an optimal model. □

If we get back to our example, we know that every rectangle having a height of 2, centred in $(4, 4)$, will never be selected in the optimal solution. It is thus useless to explore them, and we can reduce by 1 the maximum height. Algorithm 6 shows the process to compute the upper bound. As always, d_r represent the number of dense cells covered by rectangle r.

Algorithm 6: Compute upper bound on height and width for a given centre

Input: A centre (r, c), a $R \times C$ grid $\mathcal{G}$
Output: The upper bound on the height and width for rectangles centred in (r, c)

$maxHeight \leftarrow \min(r - 1, R - r)$
$reduced \leftarrow true$
while $reduced \wedge maxHeight > 0$ **do**
  $rowSup \leftarrow (r + maxHeight, c - maxWidth, r + maxHeight, c + maxWidth)$
  $rowInf \leftarrow (r - maxHeight, c - maxWidth, r - maxHeight, c + maxWidth)$
  $reduced \leftarrow d_{rowSup} == 0 \vee d_{rowInf} == 0$
  if $reduced$ **then**
    $maxHeight \leftarrow maxHeight - 1$
  end
end
$maxWidth \leftarrow \min(c - 1, C - c)$
$reduced \leftarrow true$
while $reduced \wedge maxWidth > 0$ **do**
  $colLeft \leftarrow (r - maxHeight, c - maxWidth, r + maxHeight, c - maxWidth)$
  $colRight \leftarrow (r - maxHeight, c + maxWidth, r + maxHeight, c + maxWidth)$
  $reduced \leftarrow d_{colLeft} == 0 \vee d_{colRight} == 0$
  if $reduced$ **then**
    $maxWidth \leftarrow maxWidth - 1$
  end
end
return $(maxHeight, maxWidth)$

Note that in this algorithm, it does not matter if we start by reducing the maximum height or maximum width. Indeed the algorithm finds the rectangles with the largest area, centred in (r, c) such that there is at least one dense cell in each of its side. Since this rectangle is unique, starting from the width or the height does not matter.

Increasing the Minimum Width and Height

As we have done in the previous section, it is possible to find a lower bound on the height and width of the rectangles. The reasoning is the same, if we know that for a height of 0, there is no width such that a viable rectangle can be created, we can skip this height. Let us look at the small example of Figure 4.2 and imagine that we are currently extending the centre $(3, 3)$ (in red).

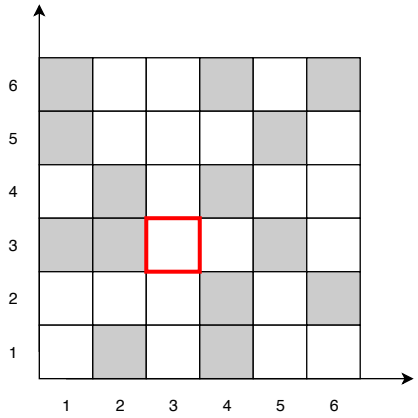


Figure 4.2: Example of lower bound computation

We know that the third column is free of dense cells. Thus, every rectangle on this column will not contain any dense cell, and will never be selected by the optimal solution (following Theorem 2). We can thus increase the minimum width to 1.

Algorithm 7 shows this process for a centre of one cell. Note that we also take into account the maximum boundaries of this particular centre. Indeed, if a cell is dense on the same row, but will never be considered in any rectangle for this centre, this it is useless to take it into account in the computation.

Extension to centre of multiple cells

The bound found for the single cell centred rectangles can be used to compute the bounds for multiple cell centred rectangles. Let us explain the process with a centre consisting of two cells on the same rows. Let $\{(r, c), (r, c+1)\}$ be a centre, $(maxHeight1, maxWidth1), (maxHeight2, maxWidth2)$ the upper bounds of the cells and $(minHeight1, minWidth1), (minHeight2, minWidth2)$ the lower bounds of the cells. The upper bounds for the centre are given by

$$maxHeight = \max(maxHeight1, maxHeight2)$$
$$maxWidth = \min(\max(maxWidth1, maxWidth2),\ \min(c-1, C-(c+1)))$$

It is clear that the cells included in each of the separate boundaries are part of a rectangle centre in both cells. Thus, it is natural to take the maximum of these boundaries; otherwise we might miss some viable candidates. For the height, there is no problem since both cells are at the same distance from the upper most and lowest row of the grid. However, for the width, since each cell is closer to one of the borders than the other, we must take care that when taking the maximum would result in going beyond the left or right side of the grid. We have $\min(c-1, C-(c+1))$ that compute the minimum distance from the centre to the border. If the maximum of the two bounds is greater than this value, then this value must be taken as the maximum width.

Algorithm 7: Compute lower bound on height and width for a given centre

Input: A centre (r, c), a grid $\mathcal{G}$, the upper bound for this centre $(maxHeight, maxWidth)$
Output: The lower bound on the height and width for rectangles centred in (r, c)

$minHeight \leftarrow 0$
for *deltaRow = 0* **to** *maxHeight* **do**
  $upperRow \leftarrow (r + deltaRow, c - maxWidth, r + deltaRow, c + maxWidth)$
  $lowerRow \leftarrow (r - deltaRow, c - maxWidth, r - deltaRow, c + maxWidth)$
  if $d_{upperRow} == 0 \wedge d_{lowerRow} == 0$ **then**
    $minHeight \leftarrow minHeight + 1$
  else
    break
  end
end
$minWidth \leftarrow 0$
for *deltaCol = 0* **to** *maxWidth* **do**
  $leftCol \leftarrow (r - maxHeight, c - deltaCol, r + maxHeight, c - deltaCol)$
  $rightCol \leftarrow (r - maxHeight, c + deltaCol, r + maxHeight, c + deltaCol)$
  if $d_{leftCol} == 0 \wedge d_{rightCol} == 0$ **then**
    $minWidth \leftarrow minWidth + 1$
  else
    break
  end
end
return $(minHeight, minWidth)$

Let us consider the minimum height and width. In this case this is simpler since we do not need to take into account the borders of the grid

$$minHeight = \min(minHeight1, minHeight2)$$
$$minWidth = \min(minWidth1, minWidth2)$$

In this case since we are considering a minimum width and height, we must take the minimum of the two values, following the same argument of the upper bounds.

Extending this to the case of a centre composed of two cells on the column in straightforward. The only difference is that, in order to compute the upper bound, we will need to be careful with the upper most and lowest row of the grid while the maximum width can be computed directly. For centre consisting of four cells, we just need to make a combination of both cases.

Skipping Some Widths

In the algorithms presented above, the idea is to check every width for a given centre and height and then increase the height. However, it is possible not to consider all the width, for a given height. To do so we need the following result.

Theorem 3. *Let $\mathcal{R} = \{R_1, \ldots, R_K\}$ be a model. If there exists $R_k = (r_k^1, c_k^1, r_k^2, c_k^2) \in \mathcal{R}$ for which we can find $c \in [c^1, \ldots, c^2]$ such that*

$$4 + 2c_d < 2c_u$$

Where c_d (resp. c_u) is the number of dense (resp. non-dense) cells covered in the sub-rectangle (r_k^1, c, r_k^2, c), then $\mathcal{R}$ is not optimal with respect to the description length.

Proof. Let us construct another model $\mathcal{R}'$ derived from $\mathcal{R}$. There are two cases to consider. First, let us consider the case where $c \neq c^1, c^2$. Let

$$\mathcal{R}' = (\mathcal{R} \setminus \{R_k\}) \cup \{(r_k^1, c_k^1, r_k^2, c-1), (r_k^1, c+1, r_k^2, c^2)\}$$

That is, we replaced the k-th rectangle, with the two rectangles formed by removing the column c from it. Let us denote r_{k1}, r_{k2} these two new rectangles. The length of $\mathcal{R}'$ is given by

$$\begin{aligned}
\mathcal{L}(\mathcal{R}') &= \mathcal{L}(\mathcal{R}) - (4 + 2u_k - 2d_k) + (8 + 2(u_{k1} + u_{k2}) - 2(d_{k1} + d_{k2})) \\
&= \mathcal{L}(\mathcal{R}) + 4 - 2(u_k - u_{k1} - u_{k2}) + 2(d_k - d_{k1} - d_{k2}) \\
&= \mathcal{L}(\mathcal{R}) + 4 - 2c_u + 2c_d \\
&< \mathcal{L}(\mathcal{R})
\end{aligned}$$

Now let us consider the case when $c = c^1$ (the exact same reasoning can be done for $c = c^2$). Let

$$\mathcal{R}' = (\mathcal{R} \setminus \{R_k\}) \cup \{(r_k^1, c+1, r_k^2, c^2)\}$$

This is the same as previously, except only one rectangle (which we will denote r'_k) is created by removing the col c. The length of the new model is given by

$$\begin{aligned}
\mathcal{L}(\mathcal{R}') &= \mathcal{L}(\mathcal{R}) - (4 + 2u_k - 2d_k) + (4 + 2u'_k - 2d'_k) \\
&= \mathcal{L}(\mathcal{R}) - 2(u_k - u'_k) + 2(d_k - d'_k) \\
&= \mathcal{L}(\mathcal{R}) - 2c_u + 2c_d \\
&< \mathcal{L}(\mathcal{R})
\end{aligned}$$

In both cases we have created another model with a smaller length. Thus $\mathcal{R}$ is not optimal.

□

We have stated the theorem on the column, but it is clear that by symmetry it can be applied on the row. In that case, to use the optimization we should invert the order of the loops in the generating algorithms, to first iterate on the width. From Theorem 3, we can derive the following corollary

Corollary 1. *If a rectangle r has a sub-rectangle, with the same height, respecting the condition in Theorem 3, then r is not part of the optimal solution.*

If during the extension process, we find a rectangle such that its first or last column respect the condition of Theorem 3, then we can skip to the next height. Indeed, for this particular rectangle, we know from Theorem 3 that it will not be part of the optimal solution. Moreover, every following rectangle formed with this height will be a super-rectangle of it. Thus from Corollary 1 we know that they can also be skipped safely. The results of these three modifications are shown in Algorithm 8. For conciseness we only show the algorithm for the one cell centred rectangles.

4.2.3 Generating candidates starting from dense cells

While the optimization explained in the previous section will give a real boost in the generation process, it does not use a structural property of the problem. Indeed, it makes no assumption on the way dense cells are distributed in the grid. Dense cells will tend to be grouped, potentially separated

Algorithm 8: Generating all rectangles centred in one cell

Input: A grid $\mathcal{G}$ of size $R \times C$
Output: A set of ROI candidates $\boldsymbol{S}$

```
selected ← {}
for row = 1 to R do
    for col=1 to C do
        (maxHeight, maxWidth) ← computeUB((r, c), G)
        (minHeight, minWidth) ← computeLB((r, c), G, (maxHeight, maxWidth))
        for height = 0 to maxHeight do
            for width = 0 to maxWidth do
                firstCol ← (row − maxHeight, col − width, row + maxHeight, col − width)
                lastCol ← (row − maxHeight, col + width, row + maxHeight, col + width)
                if 4 + 2firstCol_d < 2firstCol_u ∨ 4 + 2lastCol_d < 2lastCol_u then
                    break
                end
                r ← (row − height, col − width, row + height, col + width)
                if isValid(r) then
                    selected ← select ∪ {r}
                end
            end
        end
    end
end
return selected
```

by a large area of non-dense cells. This is due to the fact that, in our use case, the density of a cell is directly dependent on the density of the cells in its neighbourhood. The two ideas behind this optimization are the following.

First, we will change the way the lower bound is computed. Let us consider the case in Figure 4.3 when we will compute the lower bound for the cell $(3, 3)$.

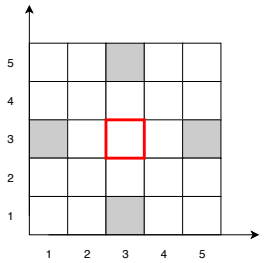


Figure 4.3: Case where the lower bound do not work well

Algorithm 7 will find

$$(minHeight, minWidth) = (0, 0)$$

. Clearly, this will not be of any help, even though it ensures not to skip any viable candidates. A more intuitive approach would be to select these bounds based on the closest dense cell because in

order to be selected, a rectangle must at least cover one dense cell. Algorithm 9 shows the new computation for lower bounds. Notice that we need to not only take the closest cell, with respect to the Manhattan distance, but if two cells are at the same distance, we must take the minimum of each distance on the rows and columns. This way, we end up with the rectangle that has the smallest area and contains the least dense cells.

Algorithm 9: Computation of lower bound for *Dense optimization*

Input: A cell (r, c), a set of dense cells D
Output: The minimum bounds on width and height

$(minHeight, minWidth) \leftarrow (\infty, \infty)$
foreach $(r', c') \in D$ **do**
  $(dHeight, dWidth) \leftarrow (|r - r'|, |c - c'|)$
  if $dHeight + dWidth < minHeight + minWidth$ **then**
    $(minHeight, minWidth) \leftarrow (dHeight, dWidth)$
  else if $dHeight + dWidth == minHeight + minWidth$ **then**
    $(minHeight, minWidth) \leftarrow$
     $(\min(minHeight, dHeight), \min(minWidth, dWidth))$
  end
end
return $(minHeight, minWidth)$

Moreover, generating rectangles far away from any dense cells does not make any sense, since most of them will not cover a sufficient number of dense cells. The second idea behind this optimization is to look only at rectangles for which the centre is near dense cells. More precisely, for each dense cell we compute its upper bounds. We can then extract all cells of the grid that fall inside the upper bound of at least one dense cells (i.e. they are inside the rectangles formed by one dense cell and its maximum height and width). Note that every dense cell is in this set. Then we can apply the generation algorithms as usual, but *only* for the centres that are composed of cells in the previously computed set. Let us take the example in Figure 4.1, in that case all the cells in the green rectangle would be considered as a potential centre, while the cells on the upper border not.

While this optimization will yield better results, we have not proved that it will only skip rectangles that will never be part of the optimal solution. However, as we will see in the next chapter, on our data set, the pruned candidates are indeed not part of the optimal solution.

4.3 A Monte Carlo method for generating Rectangular ROIs

As it can be seen in the formulation (4.4)–(4.5), the solver will favour regions that cover at most dense cell as possible. Moreover, large regions will be preferred since the cost for encoding a region is fixed, for each type of regions. We also expect most of the dense cells to be contained in the final set of ROIs. So for a given dense cell, if its neighbourhood is filled with a lot of dense cells, it is likely that it will be part of the optimal solution. Based on this, we propose a Monte Carlo method for quickly generate a set of good rectangular candidates ROI. The idea is to randomly select a dense cell and then create a set of rectangles that overlap this cell.

Algorithm 10 shows the generation process. It will generate a fixed number of candidates starting from dense cells. Each dense cell has the same probability to be selected. At each iteration of the extension process, the `extentionIsPossible` function check if the extended rectangle would still be in the boundaries of the grid. To select the next rectangle between the multiple extensions, we

give them a weight that is the number of dense cells in the extended rectangle. Then we randomly choose one, each extended rectangle having a probability to be selected proportional to its weight.

Algorithm 10: Monte Carlo method for generating candidates

Input: A grid $\mathcal{G}$, a set of dense cells $\mathcal{D}$ and a maximum number of candidates max_candi
Output: A set of $\mathcal{S}$ of ROI candidates

$\mathcal{S} \leftarrow \{\}$
while $|\mathcal{S}| < max\_candi$ **do**
  $(row, col) \leftarrow$ a cell randomly selected, with uniform distribution, from $\mathcal{D}$
  $rect \leftarrow (row, col, row, col)$
  while *isValid(rect)* $\wedge$ $|\mathcal{S}| < max\_candi$ **do**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{rect\}$
    foreach $dir \in \{left, right, top, bottom\}$ **do**
      if *extensionIsPossible(rect, dir)* **then**
        $rect_{dir} \leftarrow rect$ extended in dir
        $w_{dir} \leftarrow d_{rect_{dir}}$
      end
    end
    $rect \leftarrow$ select randomly between the $rect_{dir}$, with weight w_{dir}
  end
end
return $\mathcal{S}$

We expect this process to quickly generate a good set of candidates. Indeed the first step is to randomly chose a dense cell to start from. Thus the probability to start from a cell in an area with a high concentration of dense cells is higher than to start in an isolated dense cell. Moreover, at each extension step we favour rectangles that covers more dense cells. Thus this generation process will tend to produce rectangles in highly dense zones of the grid, which are the ROIs we are interested in discovering.

Chapter 5

Results and comparison

In this chapter, we will compare our method to the *PopularRegion* approach, on both synthetic and real-world data. This chapter will address the following questions

Q1) How well does our method perform, with respect to our optimization criterion, compared to *PopularRegion* ?

Q2) How efficient is our approach ?

Q3) What is the impact of the filtering of the candidates and what does it imply on the speedup we have by optimizing the generation of the candidates ?

Q4) Does the Monte Carlo generation process quickly generate good sets of candidates ?

Q5) Is our method robust to noise ?

We will use two versions of our model

- *MIP-rectangles* that will denote a model where the candidates are only composed of rectangles.
- *MIP* that will denote a model where there is no restriction on the shape that the candidates can take. In our case, we added circular regions.

For the rest of this chapter, except when explicitly stated otherwise, all our result will assume a 100×100 grid and the use of the exact process to generate the candidates instead of the Monte Carlo approach. Finally, except for the last question, we applied a ratio constraint on the rectangular shaped candidates in order to have more meaningful regions.

For the first four questions, we will use a real-life dataset coming from the taxi destination prediction challenge that was organized by the 2015 ECML/PKDD conference and proposed as a Kaggle competition. This data set contains more than 1.6 million trajectories from taxis of the city of Porto[1]. This is the same data set on which we have generated the Figure 2.1, Figure 2.4 and Figure 2.5. For the last question, we will use a synthetic data set for which we will explain the creation process in the associated section.

[1]The data set can be downloaded at this link `https://www.kaggle.com/crailtap/taxi-trajectory/home`. We filtered out incomplete trajectories and the few trajectories that went too far away from Porto.

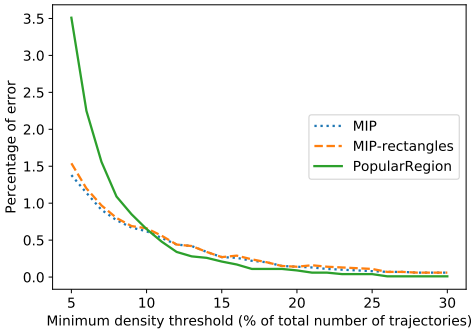


Figure 5.1: Percentage of error in function of the minimum density threshold

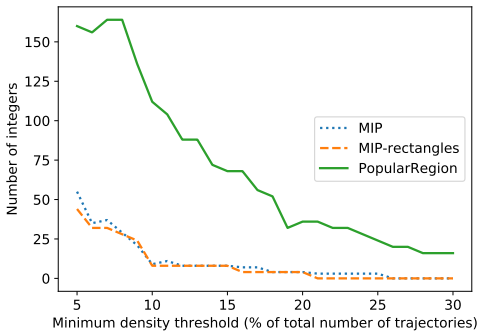


Figure 5.2: Number of integers needed to encode the ROIs, in function of the minimum density threshold.

5.1 Performances with respect to the MDL criterion

First we will compare the two approaches with respect to our optimization criterion, the description length which can be decomposed in two parts

- The number of ROIs and their length
- The number of errors made when we encode the grid with the ROIs. In our case we will analyse the error rate, that is,

$$\frac{|error^+| + |error^-|}{|\mathcal{G}|}$$

Figure 5.1 and 5.2 respectively show the error rate and the number of integers needed to encode the ROIs, as a function of the minimum support threshold.

Let us first talk about the error rate. It can be seen that for low-density threshold, our method performs better. This is due to the fact that there are more dense cells, and *PopularRegion* only need to maintain an average density for its ROIs. As explained above, in that case the algorithm finds big regions that covers a lot of non-dense cells. Note that the error made by *PopularRegion* only come from covered non-dense cells since it will cover every dense cell. On the contrary, our approach will find regions that will better fit the initial distribution of the dense cells, and thus cover fewer non-dense cells. But due to the MDL criterion, we avoid some dense cell. For example, we will never cover a single dense cell. But for low-density threshold, those cases are rarer and thus we obtain a better error rate. When the minimum density threshold increase, the dense cells become more isolated. Thus the effect we just described no longer apply. *PopularRegion* will perform slightly better because it will cover the isolated dense cells, while we do not.

On the other hand, we always give a set of ROIs that need fewer integers to be encoded. Again this is due to the fact that *PopularRegion* will create a single region, which cost four integers, for each isolated dense cell. It can be seen that for high density threshold our method tends to 0 integers because it is more advantageous to store exception than to create regions.

From these two figures, we can conclude that the main benefits of our approach are observed for lower threshold values, where we reduce the classification error using fewer regions. We can also see that the introduction of the circular regions allows to a more descriptive set of ROIs. Indeed, we see that for some density threshold, our *MIP* model obtain a lower error rate than *MIP-rectangles*

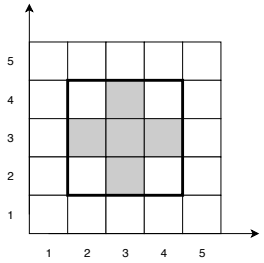


Figure 5.3: Example of ROI that cannot be found by the *MIP-rectangles* model but can with the *MIP* model

but with a higher number of integers for its ROIs. It means that some area with dense cells that are ignored by *MIP-rectangles* are correctly recovered by *MIP*. An example is given in Figure 5.3. If the cells form a circular shape (it can happen for example at a highway interchange or highway exit), then it cannot be recovered with a rectangular ROI using a MDL optimization criterion. Indeed the smallest possible rectangular ROI covering it is shown in bolder line. The cost of this region is $4 + 8 = 12$ (four for the rectangle, and four exceptions). If we would not have covered it, the cost of the exception would be 10. Thus this ROI will never be selected. However, circular ROI can perfectly match this type of region, with a cost of 3. Thus for some configuration we allow to have a larger set of ROIs if it decreases enough the error rate. Note that the full model cannot have a higher description length since we only add candidates.

5.2 Execution times

Next we will see how efficient is our approach. Table 5.1 shows the run time of our method for three different grid size, and two different minimum density threshold. In this table we report for our method the time needed to generate the candidates (rectangles and circles) as well as the time needed to solve the optimization problem defined by Equations (4.4)–(4.5). As can be seen, the resolution of

Minimum density threshold	2%			5%		
Grid side size	100	150	200	100	150	200
Number of dense cells ($\|\mathcal{G}^*\|$)	571	596	534	229	178	135
Number of MIP candidates	71 320	25 892	10 559	7 210	3 348	959
MIP candidate generation time (s)	3.102	7.515	14.242	1.222	3.745	7.818
MIP optimization time (s)	14.608	2.766	0.547	0.516	0.178	0.037
PopularRegion run time (s)	0.018	0.027	0.037	0.012	0.021	0.036

Table 5.1: Run time of the methods for different grid sizes and minimum density thresholds.

the MIP is mostly determined by the number of candidates as those correspond to the number of the variables in the model. It is thus interesting to see how these candidates evolve with the configuration. The worst case is when most of the grid would be filled with dense cells. Fortunately, this does not correspond to practical applications because we would not discover meaningful ROIs. Even for a low-density threshold (e.g. 2%), there is only 571 dense cells in a 100×100 grid. Moreover, as explained in the previous chapter we have optimized the generation of the rectangular candidates in such way that we have reduced the number of candidates, by not exploring useless ones. As expected,

the number of candidates drops when the size of the grid increases. It is due to the fact that when there are more cells in the grid; thus the dense cells will be less packed. Thus for a given region, it is harder to cover more dense than non-dense cells. The number of candidates also decreases when the minimum density threshold increase since there is less dense cells. Finally, we can see that the time needed to generate the candidates increases with the size of the grid. It was expected since we have more candidates to consider. Most of the time needed to generate the candidates comes from the circular regions, since we have not optimized the generation process for them and we do a brute-force approach.

Finally, as expected, our method is much slower than *PopularRegion*. However, it still runs in a reasonable amount of time and there is still room to improvement by, for example, optimizing the process to generate circular regions.

5.3 The candidate generation process

The goal of the optimization process was twofold. First it needed to drastically reduce the number of explored candidates. Given the low percentage of selected candidates, exploring the whole search space is too high a cost. Secondly, we need to reduce the growth of the search space when the size of the grid increase. We know that we cannot, theoretically, have a better complexity than $\mathcal{O}(R^2C^2)$. But since the grid will, on plausible data, be mostly filled with non-dense cells, we can expect a much lower average complexity.

The Table 5.2 and Table 5.3 shows the results for two different density threshold, and three grid sizes. In order to have a fair comparison, the initial grid was reduced such that each border of the grid contains at least one dense cell. Otherwise, the number of candidates explored by the naive process would be too high (e.g. in a 200×200 grid there are more than 400 millions distinct rectangles). We do not want to compare to an approach that explores regions of the grid that do not contain any data. Note that both optimization would naturally reduce the grid this way by reducing the upper bounds, but it would consume a lot of resources. The tables are divided in two parts. The upper part shows the number of candidates considered and selected while the lower part shows how the set of selected candidates has evolved. When a rectangle is selected by the naive process but not by one of the optimizations, we needed to decide if it would potentially have been selected in the optimal solution. To do so, we first check if one of the sides contains more non-dense than dense cells. In this case we know that it will never be part of the optimal solution. If this is not the case, then we try to manually find a better decomposition of this rectangle in one or more sub-rectangles such that the replacement of the candidate by its sub-rectangles yields a better solution.

grid size	100			150			200		
	Explored	Selected	Time(s)	Explored	Selected	Time(s)	Explored	Selected	Time(s)
Naive process	2 904 450	11 727 (0.4038 %)	2.88	14 107 500	5 279 (0.0374 %)	14.33	34 627 230	966 (0.0028 %)	36.17
Bound optimization	81 445	6 818 (8.37 %)	0.25	168 143	3 096 (1.84 %)	0.59	242 602	918 (0.38 %)	0.96
Dense optimization	54 805	6 801 (12.41 %)	0.22	96 460	3 115 (3.23 %)	0.40	148 010	899 (0.61 %)	0.55
	Pruned	Side more non-dense	Better decomp.	Pruned	Side more non-dense	Better decomp.	Pruned	Side more non-dense	Better decomp.
Bound optimization	4 909	4 821	88	2 183	2 160	23	48	47	1
Dense optimization	4 926	4 829	97	2 164	2 141	23	67	57	10

Table 5.2: Comparison for the three process for a minimum support threshold of $0.05|\mathcal{T}|$

The first thing we can observe on these data it that, as expected, the percentage of selected candidates within the explored set if very low for the naive process, less than 1% for most of the configuration. Thus almost all of the computation time is waste by exploring unnecessary candidates. Although there is an improvement when we used one of the optimizations, the best-case reach 30%. But for both optimization, the reduction of the computation time balance this low percentage.

The reduction of the number of explored candidates is substantial for both optimization. As expected, the *Dense Optimization* prunes even more non-necessary candidates. Moreover, it seems

grid size	100			150			200		
	Explored	Selected	Time(s)	Explored	Selected	Time(s)	Explored	Selected	Time(s)
Naive process	5 826 171	265 760 (4.5615 %)	6.16	22 719 480	53 463 (0.2353 %)	23.08	52 366 860	17 734 (0.0339 %)	52.38
Bound optimization	247 142	66 370 (26.86 %)	0.60	418 287	24 494 (5.86 %)	1.07	672 628	9 826 (1.46 %)	1.77
Dense optimization	215 979	66 139 (30.62 %)	0.88	278 660	24 397 (8.76 %)	1.30	397 059	9 875 (2.49 %)	1.67
	Pruned	Side more non-dense	Better decomp.	Pruned	Side more non-dense	Better decomp.	Pruned	Side more non-dense	Better decomp.
Bound optimization	199 390	190 952	8 438	28 969	28 456	513	7 908	7 774	134
Dense optimization	199 621	191 093	8 528	29 066	28 510	556	7 859	7 727	132

Table 5.3: Comparison for the three processes for a minimum support threshold of $0.02|\mathcal{T}|$

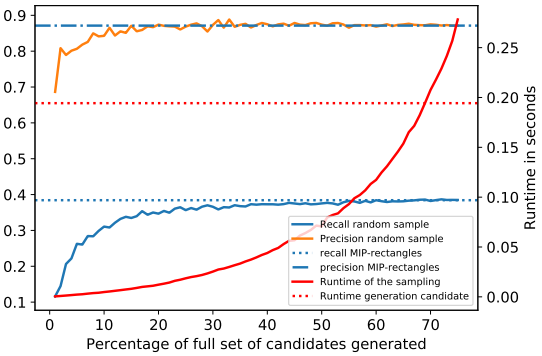


Figure 5.4: Recall, precision and run time of the Monte Carlo approach in function of the percentage of candidates

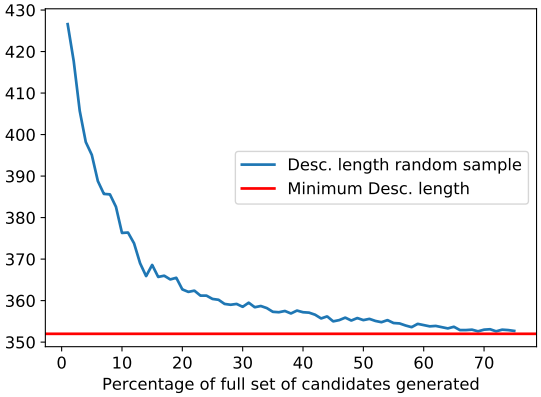


Figure 5.5: Description length of the Monte Carlo approach in function of the percentage of candidates

that the number of explored candidates grows slower in both optimization than in the naive process. This does not necessarily mean that the average complexity has decreased, but determine that experimentaly is hard since we cannot increase too much the size of the grid. Doing so would create so small cells that it would even be hard to have dense cells.

5.4 The Monte Carlo method

Now we will see how well the Monte Carlo process defined in Section 4.3 performs. Since this method only generate rectangular candidates, we will compare it to the *MIP-rectangles* approach. The number of candidates that will be generated by Algorithm 10 will be expressed as a percentage of the total number of candidates that are valid (i.e. for which the function `isValid` return `true`). To do our experiment, we generate a set of candidates using the Monte Carlo approach, for a given percentage, and then use this set to solve the optimization problem defined by Equations (4.4)–(4.5) The experimentation was done on a 100×100 grid with a minimum density threshold of $0.05|\mathcal{T}|$. We will use the recall and precision as metrics to evaluate the performances of this method

$$recall = \frac{|\mathcal{G}^* \cap \mathcal{S}|}{|\mathcal{G}^*|} \tag{5.1}$$

$$precision = \frac{|\mathcal{G}^* \cap \mathcal{S}|}{|\mathcal{S}|} \tag{5.2}$$

with $\mathcal{G}^*$ the set of dense cells and $\mathcal{S}$ the set of cells covered by the solution returned by the solver.

Figure 5.4 shows how the recall, the precision and the run time of the approach evolve when the percentage of candidates increases. For each of them, we also show the value of the full approach,

with the *Dense Optimization* defined in Section 4.2.3, to generate the rectangular candidates. As can be seen, the Monte Carlo based approach quickly reach a high level of recall and precision, with respect to the optimal solution. Since our sampling start from dense cells, we can quickly expect that each dense cell is at least in one candidate. Thus it is not surprising the recall is good for a low percentage of candidates. Moreover until a percentage of 70% the time needed to generate the random sample is inferior to the time needed to generate all the candidates with the *Dense Optimization*. So with respect to these metrics, the Monte Carlo process performs well. However for the MDL criterion we need at least 70% of the candidates in order to reach the optimum, as shown in Figure 5.5. For this number of candidates, it is more efficient to use the generation process describe in Section 4.2.3.

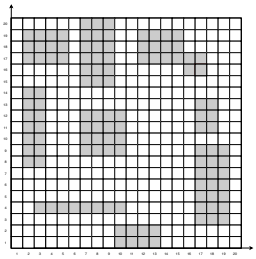
In conclusion, the Monte Carlo approach is a good approach to quickly generate a set of candidates that will have a good recall and precision. However reaching the optimum with respect to the MDL criterion is harder and might require more time. Moreover this approach is much slower for sparser matrix.

5.5 Robustness to noise

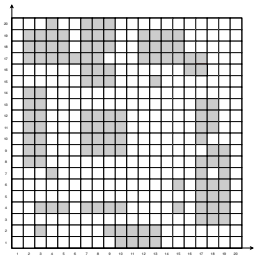
In this section we will describe experimentation, we have done on synthetic data. To create the input matrix of the algorithms, we start from a grid with predefined regions of interest. Then we apply a noise on this matrix, that is, we change the state of each cell with a given probability p. We then try to identify the ROIs on this modified matrix, and we compute the recall and the precision of the solutions, with respect to the initial matrix. In this experiment we start from a 20×20 grid shown in Figure 5.6a. Figure 5.6b shows an example of grid after applying a noise of 0.03 (i.e. each cell has a probability $p = 0.03$ to be inverted). After applying the noise, every cell is given a value representing the number of trajectories that goes by it. Assuming a minimum density threshold of 0.05, we give each non-dense cell a value between 0 and 4. For the dense cells, we give them a value between 5 and 30. We do not go beyond 30 because such values are unlikely in real data set while being strong outliers the *PopularRegion* algorithm. We should note that using a uniform might not be the best choice given the fact that the density of the cells most likely does not follow a uniform distribution. However simulating the real distribution of the density and the interactions between the cells is not trivial, and outside the scope of this thesis. Figure 5.6c and 5.6d shows the solution found by our approach and *PopularRegion* for the grid in Figure 5.6b.

Figure 5.7 shows how these two metrics evolve when the noise increases. Note that we did not go beyond a noise of 0.5 since at this level the matrix becomes random, and after that we would start to lose the information. *PopularRegion* has a recall that is always close to 1.0. This is due to the fact the algorithm will cover every dense cell. For low levels of noise, the initial dense cells are still in area with a lot of dense cells (see for example Figure 5.6b) and will naturally be covered. With a higher level of noise, the matrix will have a more uniform distribution of the dense cells. It will thus be easier for *PopularRegion* to extend and maintain an average density. This will gives big regions, covering almost all the grid, and thus most of the dense cells. This high value for the recall comes at the cost that the precision quickly drops to a low value. Indeed the large ROIs cover a lot of non-dense cells, impacting the precision. It can be seen that the precision stabilize around 0.3, which is approximately the proportion of dense cells in the grid : $\frac{199}{400} = 0.2975$.

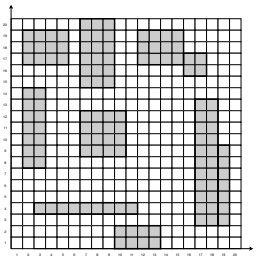
The behaviour of our method is quite different. Both metrics start at a high level due to the fact that the found regions will match as much as possible the dense cells. For low levels of noise, the false dense cells are too isolated to be considered by our algorithm since it costs more to cover them than to treat them as an error. In the same manner, the flipped dense cells inside the initial ROIs will have a too low impact and the whole region will still be considered as a ROI by our method. When the noise increase, the inversion of the cells will cause a deformation of the found ROIs, as it



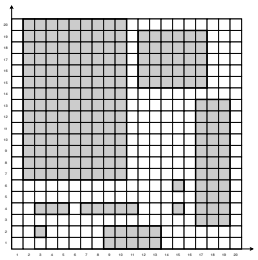
(a) Initial grid. Each coloured cell is dense.



(b) Grid after applying the noise with $p = 0.03$



(c) Regions of interest returned by the MIP-rectangles method without ratio restriction



(d) Regions of interest returned by *PopularRegion*

Figure 5.6: Example of synthetic data

is shown on Figure 5.6c. But since we will try to match at best the distribution of the dense cell, the drop in the metrics is more progressive. Interestingly when the probability of flipping a cell is $p = 0.5$, our algorithm reaches the same precision value as *PopularRegion*. Since the matrix is random for this amount of noise, each cell cover by the regions is dense with the following probability

$$\frac{|G^*|}{400} = 0.2975$$

thus it is not surprising that both approaches reach this value.

The conclusion of this synthetic experiment is that our method is quite robust to noise. Even if the found regions are sometimes different, the main areas are still discovered while avoiding isolated exception.

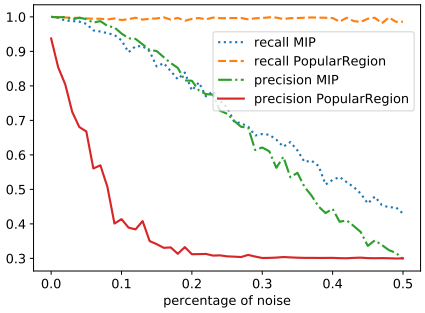


Figure 5.7: Recall and precision for *PopularRegion* and our algorithm in function of the noise

Chapter 6

Conclusion and further work

In this master thesis, we have introduced a new approach to discover a set of *Region of Interest* based on an extended *Integer Programming* formulation. In this formulation, we first generate a set of candidates and then select the ones that minimize a *Minimum Description Length* criterion.

We have shown that our approach solve known pathological aspects of the current main method, *PopularRegion*. In particular we avoid finding too large ROIs for low-density threshold, while providing similar solutions for higher density threshold.

However this comes at the cost of a much higher run time. While *PopularRegion* was greedy, our method is based on an IP formulation which is known to be NP-complete. Thus the optimization of the candidate generation process is essential and has proved to drastically reduced the number of candidates, and thus the optimization time. Overall our method has shown good performances for the tested configuration, that are the most common in our use case.

The decomposition of the problem as a two-step process allows a lot more flexibility in the kind of shape we can have for our ROIs. In the scope of this work, we have only added circular ROIs and it has introduced a small improvement compared to the model with only rectangular ROIs. But there is no limit in what type of candidates we can generate, as long as we can assign them a weight. More work will be done on this aspect. In the near future, we might explore a constraint-based algorithm to generate candidates based on a set of constraints that the user define. It would be useful since every application might not need the same type of regions. The choice of the MDL criterion might also be questioned. The function to be minimized by the optimization solver might be dependent on the application. We chose this one because it is a general principle that we should favour smaller models. But in our scheme this function is easily replaced as long as it operates on the candidates.

Finally, we need to evaluate the impact of this work in the broad range of applications that use trajectory data. We have the intuition that our approach allows having better ROIs, but it needs to be confirmed in subsequent work. For example, we need to analyse the pattern found in a *Trajectory Mining* task, or is the accuracy increase when doing *Next Location Prediction*.

Bibliography

[1] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne. Understanding the limits of lorawan. *IEEE Communications magazine*, 55(9):34–40, 2017.

[2] R. Agrawal, R. Srikant, et al. Mining sequential patterns. In *IEEE International Conference on Data Engineering (ICDE)*, volume 95, pages 3–14, 1995.

[3] J. O. Aoga, T. Guns, and P. Schaus. An efficient algorithm for mining frequent sequence with constraint programming. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 315–330. Springer, 2016.

[4] M. Conforti, G. Cornuéjols, and G. Zambelli. Extended formulations in combinatorial optimization. *4OR*, 8(1):1–48, 2010.

[5] P. DeMaio. Bike-sharing: History, impacts, models of provision, and future. *Journal of public transportation*, 12(4):3, 2009.

[6] S. Gambs, M.-O. Killijian, and M. N. del Prado Cortez. Next place prediction using mobility markov chains. In *Proceedings of the First Workshop on Measurement, Privacy, and Mobility*, page 3. ACM, 2012.

[7] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 330–339. ACM, 2007.

[8] M. Gorawski and P. Jureczek. Regions of interest in trajectory data warehouse. In *Asian Conference on Intelligent Information and Database Systems*, pages 74–81. Springer, 2010.

[9] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[10] V. Klee and G. J. Minty. How good is the simplex algorithm. Technical report, WASHINGTON UNIV SEATTLE DEPT OF MATHEMATICS, 1970.

[11] X. Kong, M. Li, K. Ma, K. Tian, M. Wang, Z. Ning, and F. Xia. Big trajectory data: A survey of applications and services. *IEEE Access*, 6:58295–58306, 2018.

[12] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 34. ACM, 2008.

[13] Y. Liang, Z. Cai, Q. Han, and Y. Li. Location privacy leakage through sensory data. *Security and Communication Networks*, 2017, 2017.

[14] F. Margot. Symmetry in integer linear programming. In *50 Years of Integer Programming 1958-2008*, pages 647–686. Springer, 2010.

[15] W. Mathew, R. Raposo, and B. Martins. Predicting future locations with hidden markov models. In *Proceedings of the 2012 ACM conference on ubiquitous computing*, pages 911–918. ACM, 2012.

[16] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 637–646. ACM, 2009.

[17] Y. Nesterov and A. Nemirovskii. *Interior-point polynomial algorithms in convex programming*, volume 13. Siam, 1994.

[18] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[19] S. Shang, L. Chen, C. S. Jensen, J.-R. Wen, and P. Kalnis. Searching trajectories by regions of interest. *IEEE Transactions on Knowledge and Data Engineering*, 29(7):1549–1562, 2017.

[20] T. Terlaky. *Interior point methods of mathematical programming*, volume 5. Springer Science & Business Media, 2013.

[21] F. Vanderbeck and L. A. Wolsey. Reformulation and decomposition of integer programs. In *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer, 2010.

[22] J. J.-C. Ying, W.-C. Lee, T.-C. Weng, and V. S. Tseng. Semantic trajectory mining for location prediction. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 34–43. ACM, 2011.

[23] J. Yuan, Y. Zheng, L. Zhang, X. Xie, and G. Sun. Where to find my next passenger. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 109–118. ACM, 2011.

[24] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie. T-finder: A recommender system for finding passengers and vacant taxis. *IEEE Transactions on knowledge and data engineering*, 25(10):2390–2403, 2013.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**