POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master Degree Thesis

A parallel algorithm for mining sequences of spatio-temporal co-location patterns

Supervisors
Prof. Paolo GARZA
Dott. Luca COLOMBA

Candidates
Luca Ferraro

Accademic year 2022-2023

Summary

This master thesis focuses on spatio-temporal data mining, a specialized field of data mining and spatial analysis that aims at discovering interesting patterns, relationships, and insights from data that have both spatial and temporal dimensions. These relationships are useful to help in a wide spectrum of applications and contexts, from urban planning to epidemiology. Among different approaches to address spatio-temporal data mining we consider co-location pattern mining. This method tries to uncover correlation between features or attributes of a dataset whose instances are usually found together in the same geographic area and at the same time. For example, if we consider a dataset that represents events that may happen in a urban context, we may find that an episode of type "Traffic-congestion" is often spatially and temporally close to an episode of type "Sport-event". If these two types of events are found close a certain number of times, a co-location pattern ["Traffic-congestion", "Sport-event"] may be discovered.
Most of state of art methods to perform co-location pattern mining consider only spatial datasets without any temporal information. In this context the work "Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation" proposes an innovative approach to perform spatial co-location mining. This method divides the entire set of neighbor relationships among instances into some partitions. The co-location mining is then applied independently for each partition employing some new ideas to reduce the search space and thus the time and resources required by co-location mining. So, this algorithm avoids serial processing and the limitations typical of single machine computing. It can be applied also to process massive spatial dataset.
One step forward in the field of co-location mining has been made in "Temporal co-location pattern discovery in spatiotemporal data through parallel computing". This work shows that the previously mentioned algorithm can be extended to perform mining of spatio-temporal co-location patterns. The contribution of this master thesis is the introduction of some new solutions in

order to make spatio-temporal co-location mining more efficient and scalable. In particular we propose some new ideas to find effectively spatio-temporal neighbors of each event in the dataset. Our solutions are effective and improves the efficiency of the baseline algorithm.
Successively, in order to extend the power of co-location mining, we introduce the concept of "sequence of co-locations" or (co-location sequence). We define a co-location sequence as a collection of co-locations that has an aggregator event in common and respect some spatial and temporal constraints. Co-location sequence mining should be able to discover relationships and correlations, among spatio-temporal data, that simple co-location mining is not able to find. In this thesis we provide a formal definition of co-location sequence, we introduce some suitable metrics to evaluate how interesting are the sequences found and we develop an algorithm to perform co-location sequence mining. The algorithm works in parallel manner and it is available for huge spatio-temporal datasets (more than 30 M events). Our solutions are developed using Apache Spark framework and Python language, the experiments are run on BigData@Polito Cluster. The results we get confirm the effectiveness of our algorithm for co-location sequence mining except for some limitations mainly due to the lack of proper real-word dataset and restricted amount of available computational resources.

Contents

Chapter 1

Introduction

1.1 Big Data and Spatio-temporal Data

In an era defined by rapidly advancing technology, our world is flooded with an unprecedented amount of information. This influx of data has given rise to a phenomenon that has revolutionized industries, transformed decision-making processes, and redefined the boundaries of innovation, the era of Big Data. At the heart of the Big Data revolution is the concept of data mining, the process of uncovering valuable patterns, trends, and relationships within large and complex datasets. With the right tools and techniques, we can manage to extract meaningful knowledge from this huge amount of information. These data encompass a wide spectrum of sources and types, including social media platforms, e-commerce websites, scientific research, healthcare records, and beyond. In this thesis we focus on spatio-temporal data, this type of data captures the interaction between geographical coordinates and chronological instants. Its significance lies not only in the individual attributes of space and time, but in the intricate relationships that emerge when they are combined. Understanding these relationships is crucial for making informed decisions and predicting trends on our dynamic real world. In particular in the next sections we'll focus on two types of spatio-temporal data:

- People trajectories represented by a sequence of time-stamped points, each of which contains also the information of latitude and longitude.

- Traffic and weather events, each of which contains a geographical location along with a starting and ending timestamp.

1.2 Mining Spatio-temporal Data and Co-location Patterns

Spatio-temporal data encapsulates a multidimensional landscape where geographic coordinates and chronological instants converge. One way to extract useful knowledge from this kind of data is **co-location pattern mining**. A co-location pattern (or co-location) represents a subset of spatial features, whose instances are frequently located together in spatial neighborhoods. Originally the concept of co-location was related only to spatial features and data, though the definition of co-location pattern can be extended to spatio-temporal data (as we'll discuss in chapter 3). We can visualize the main objective of co-location mining considering the situation in figure 1.1. In this simple example we can observe 3 types of events located in a city map: car accident, traffic jam and construction work . The goal of spatio-temporal mining is to discover the neighbors of each data event and finally extract some general co-location rules. In this simple example potential co-locations are: $[traffic - jam, accident]$; $[public - event, construction]$. A good implementation for co-location mining should be able to find out only patterns that are meaningful. It's likely that the event "construction" and "public event" are close just for a coincidence; while it's reasonable that a "traffic-jam" event generates some issues to the traffic flow that may result in a "car accident". Good algorithms for co-location mining define suitable metrics and solutions to extract only significant patterns.

In this thesis we first analyze a spatial co-location pattern mining (SCPM) algorithm: Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation [1]. Then we discuss an updated version of this SCPM algorithm developed by Liturri [9] that takes in account also the temporal dimension and it is usable for data that has both geographical position and also temporal information. We aim to work with massive spatio-temporal datasets, this makes the mining process very computational expensive. Thus, our first contribution tries to make the algorithm developed by [9] more efficient and scalable in order to be employed with larger datasets. Beyond this improvement our main contribution tries to take the mining of co-location patterns to the next level. We introduce the concept of **sequence of co-location patterns** as a set of consecutive co-location patterns that happens in the same geographical area. The events that compose the sequence of co-locations should also meet some temporal constraints (discussed in 4.1). We introduce this new concept of sequence of co-locations

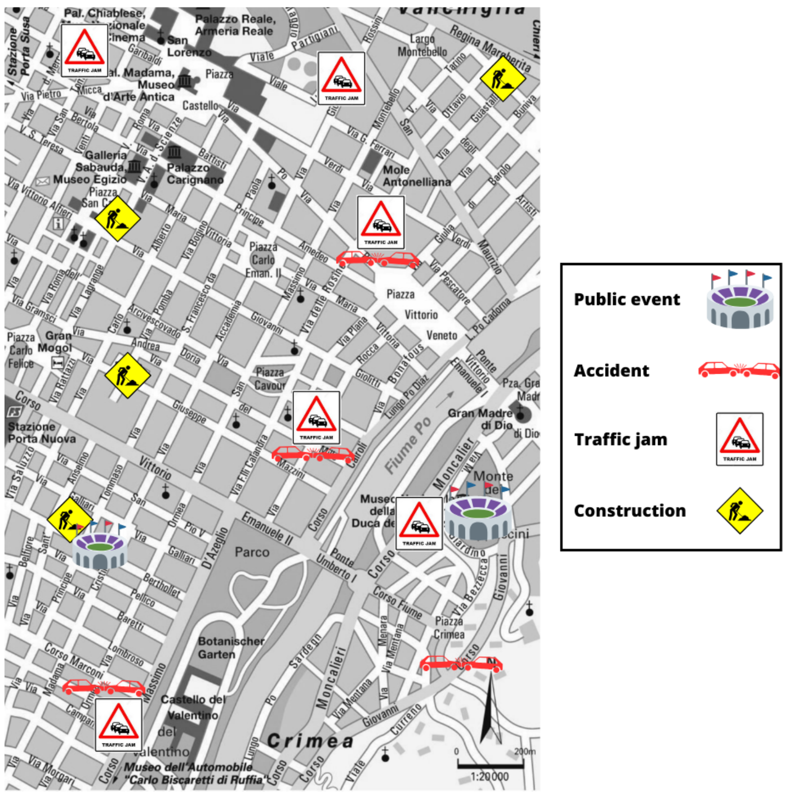

Figure 1.1. Example of different types of traffic events located in a map

to study and analyze the relationship between registered traffic events in a specific area and their dependency to weather events. The rules defined by the spotted sequences of co-locations may help to understand and organize better an urban city environment; for example reducing car accidents and making the traffic flow more functional. We define some new metrics to evaluate the importance of the co-location sequences. We also develop a new algorithm for co-location sequence mining, which works in parallel way and is able to compute the defined metrics in efficient and scalable manner. The algorithm is developed and run using Spark technology, this makes it available to be employed for dataset with very huge amount of samples (up to 30 millions).

1.3 Thesis Structure

The following sections of this document are organized in this way.:

- Chapter 2 provides a formal definition of co-location patterns and overviews the state-of-art techniques available for Spatio Co-location Pattern Mining (SCPM). Then we focus on Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation [1] which is a modern SCPM technique designed to be highly scalable and suitable to operate with massive dataset.

- In chapter 3 we present a temporal extension of the co-location pattern mining. We discuss the algorithm designed by Liturri in [9] to find spatial and temporal correlations between trajectories data. We propose some improvements to the spatio-temporal co-location pattern mining algorithm in order to make it scalable and usable with larger datasets. We show some evaluation results comparing the performance of original and improved version of the algorithm. Finally we report a summary of the results obtained applying spatio-temporal co-location mining on a dataset containing information about traffic and weather events.

- The main contribution of the thesis is presented in chapter 4. Here we introduce the new concept of co-location sequences. We brainstorm and define some metrics suitable to measure the importance of co-location sequences. Then we define an algorithm to make co-location sequence mining in a parallelizable and efficient way exploiting Apache Spark technology and Python. We also report some results obtained analyzing a dataset which contains traffic and weather events.

- Finally we summarize the results obtained in the conclusion chapter 5. Here we also discuss the limitations of our contribution as well as possible future extensions and improvements.

Chapter 2

Spatial Co-location Pattern Mining

Co-location patterns refer to the simultaneous occurrence of multiple geographic features or spatial objects in close proximity to each other within a given geographical area. Co-location patterns represent spatial features whose instances are often located in close geographic proximity. These patterns reveal associations and relationships that may not be apparent through individual analysis of each feature. Co-location pattern mining purpose is to identify and analyze these patterns to gain insights into the underlying spatial processes, interactions, and dependencies. However mining co-location patterns may result particularly challenging when working with huge amount of data because it is a very expensive computation process. In this chapter we first define the main notions for understanding co-location pattern mining, then we overview state of art methods to address the problem focusing on a a innovative approach called Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation [1]

2.1 Fundamental Concepts And Problem Definition

First we need to introduce some fundamental new concepts and metrics in the field of Spatial Co-location Mining. In order to clearly explain these new concepts we refer to a simple example of spatial dataset. The dataset is composed of few samples localized in the space and associated with a category label (represented in 2.1). Each sample instance could be close to one or more instances belonging to a different category type (this relationship is shown with red lines in figure 2.1).

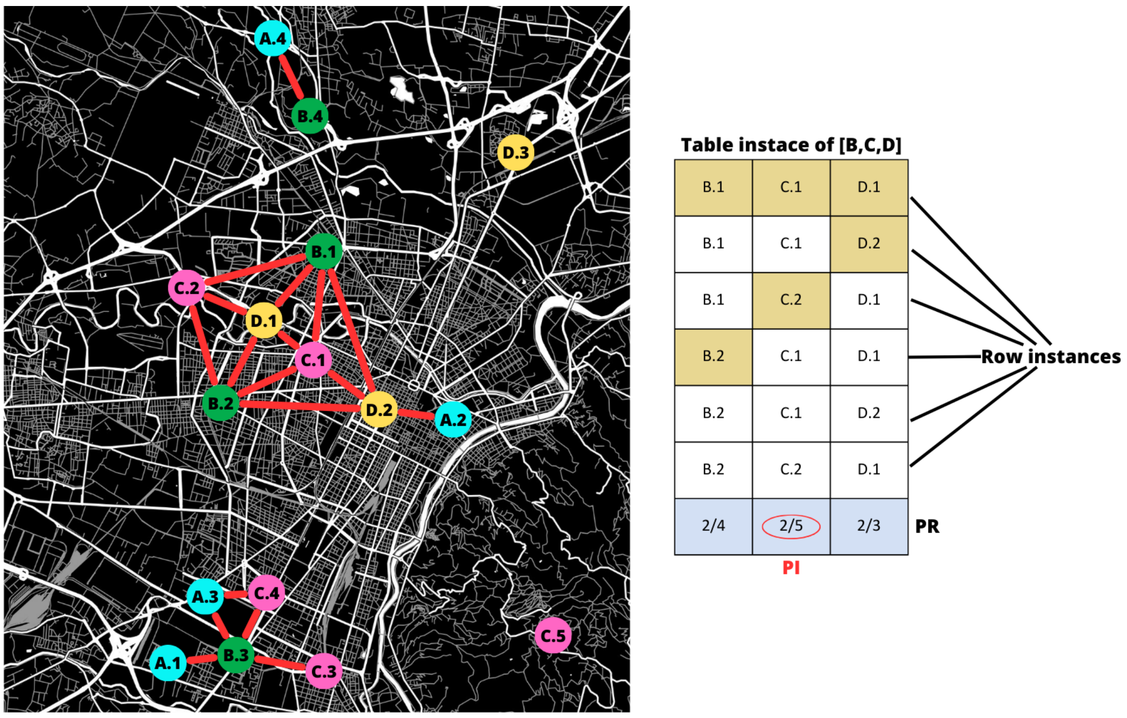

Table instace of [B,C,D]

B.1	C.1	D.1
B.1	C.1	D.2
B.1	C.2	D.1
B.2	C.1	D.1
B.2	C.1	D.2
B.2	C.2	D.1
2/4	2/5	2/3

Figure 2.1. Set of spatial instances and neighbor relationships; table instance for co-location [B,C,D]

Spatial feature: is the name that describes a location or spatial entity. The set of features in the dataset is represented with $F = \{f_1, f_2, f_3, ...f_m\}$. In the simple example (figure 2.1) we consider a set of 4 generic features $F = \{A, B, C, D\}$. In a more realistic scenario spatial features may be categories of shops (e.g. restaurant, supermarket) or categories of events related to traffic (e.g. car-accident, traffic-jam).

Spatial instance: refers to the occurrence of a specific spatial feature with a geographic location. Given a set of spatial features F, the set of instances of F is represented as $O = \{o_1, o_2, o_3, ...o_n\}$. Each instance o_i is composed by at least this information: $< \mathit{feature\ type}, \mathit{location} >$. The location is usually expressed in term of latitude and longitude. For example in figure 2.1 there are 3 spatial instances for feature D (yellow points).

Spatial neighbor relationship (R) : when two spatial instances (o_i, o_j)

have a distance which is less than a specified distance threshold (*min_distance*) they have a neighbor relationship. This is denoted as

$$R(o_i, o_j) \implies distance(o_i, o_j) \leq min\_distance \tag{2.1}$$

(o_i, o_j) should belong to different features. The neighbor relationships are showed with red lines in figure 2.1.

Neighbor set: given an instance o, the neighbor set of o is defined as the collection of instances that has the neighbor relationship with o. It is referred as $\{Neigh(o) = o' | o' \in O, R(o, o'), o' \neq o\}$. For example in figure 2.1, $Neigh(D.1) = \{B.1, B.2, C.1, C.2\}$. The neighbor set of some instance may also be empty.

Neighbor database (D): the neighbor set of all instances is the neighbor database. In table 2.1 is reported the neighbor database for our simple example (figure 2.1).

Instance	Prefix feature	Neighbor set
A.1		{B.3}
A.2		{D.2}
A.3		{B.3, C.4}
A.4		{B.4}
B.1		{C.1,C.2,D.1,D.2}
B.2		{C.1,C.2,D.1,D.2}
B.3	A	{A.1,A.3,C.3,C.4}
B.4	A	{A.4}
C.1	B	{B.1,B.2,D.1,D.2}
C.2	B	{B.1,B.2,D.1}
C.3	B	{B.3}
C.4	A,B	{A.3,B.3}
C.5		{}
D.1	B,C	{B.1,B.2,C.1,C.2}
D.2	A,B,C	{A.2,B.1,B.2,C.1}
D.3		{}

Table 2.1. Neighbor database for example in figure 2.1

Spatial co-location (C): two or more features compose a spatial co-location if the instances of these features frequently form cliques under the constraint of spatial neighbor relationship. To establish "how frequently" these cliques should be formed to compose a co-location, some suitable metric are introduced in the following. Given a set of features $F = \{f_1, f_2, f_3, ...f_m\}$ a spatial co-location $C = [f_1, f_2, f_3, ...f_k]$ is a non-empty subset of F $(C \subseteq F)$ whose instances are usually spatially close. The number (k) of features in the spatial co-location is called ***size***. The features of the co-location C should be sorted in ascending lexicographic order.

Row instance or ***co-location instance*** (***RI***) of a co-location C: is a set of instances that respect 2 constraints:

1. the row instance RI covers all the features of C and no subsets of RI covers all the features of C.

2. The spatial neighbor relationship $R(o_i, o_j)$ holds for any pairwise taken from *RI*. The instances of RI form a ***clique***.

Table instance of a co-location C: is the collection of all row instances of C, it is denoted as *T(C)*. The table instance for co-location pattern [B,C,D] is reported in figure 2.1.

Participation Ratio(PR): considering f_i as a feature of the co-location C $(f_i \in C)$, the participation ratio of f_i in C is the ratio between the number of distinct instances of f_i in the table instance of C *T(C)* and the total number of f_i instances in the original dataset. The participation ratio of f_i in C is denoted as $PR(f_i, C)$.

$$PR(f_i, C) = \frac{number\ of\ distinct\ instances\ of\ f_i in T(C)}{number\ of\ distinct\ instances\ of\ f_i} \tag{2.2}$$

Participation Index : introduced by [2] as a metric to measure the importance of each co-location. The participation index of a co-location C is denoted as *PI(C)* and represents the minimum participation ratio among all the features in C:

$$PI(C) = min_{i=1}^{k} PR(f_i, C) \tag{2.3}$$

Considering the pattern [B,C,D] in 2.1, the feature C has the minimum participation ratio, so $PI([B,C,D]) = PR(C,[B,C,D]) = 2/5$.

A co-location C is considered **prevalent** if its participation index *PI(C)* is greater than a defined threshold *min_Participation_Index*. The objective of the Spatial Co-location Pattern Mining is to extract all the prevalent patterns from a dataset containing spatial instances.

2.2 State of Art Methods

Mining spatial co-location patterns is an arduous task, it requires to compute the participation index. This index depends on the table instance, so it demands to identify all the row instances of the co-location patterns, resulting in a very computational expensive process. Lot of efforts have been made to find efficient algorithms able to perform co-location and reduce time and resources needed to compute participation index. Another challenge of algorithms for co-location pattern mining is that they should be designed to discover all significant patterns without missing anyone. We now overview some of the recent studies in this field.

One of the first work in this domain has been released by Huang et al. in [3]. This is a join-based approach for co-location mining. In order to create a table instance of a size-k candidate, this technique connects two common size-(k1) co-locations. The algorithm developed in [3] is legacy, however in this paper are introduced some concepts which are still fundamental for modern co-location mining applications.

The join-based approach has been overcome by a solution presented in the paper *A Join-less Approach for Co-location Pattern Mining: A Summary of Results* [10]. It introduces a join-less methodology for co-location pattern mining and a way to materialize the neighbor relationships of a geographical dataset without duplicating the neighbor relationships or losing co-location instances. The join-less co-location mining approach contains a coarse pruning step that can select possible co-locations without actually locating co-location instances, which lowers the computational cost of finding instances of co-location patterns using an instance lookup scheme. This algorithm has been evaluated using synthetic datasets and real world datasets; the results show the join-less co-location mining algorithm outperforms the join-based algorithms [3] and that it is scalable in dense spatial datasets. However, even better results are obtained by Wang et al. in the paper *A* *new join-less approach for co-location pattern mining* [8]. In this work a new structure called CPI-tree (Co-location Pattern Instance Tree) is introduced. Spatial neighbor relationships between instances are put into the CPI-tree, then the table instances are generated from there.

The aforementioned algorithms use participation index as metric to discover prevalent patterns. However, Yao et al. [6] observed that the participation index calculation treats the spatial space as homogeneous; neighboring instances are connected but without taking into account their amount of proximity or orientations, which can reduce the potency of the mined patterns. Tobler's First Law indicates that the contributions of instances to their pattern's significance diminish as the distance decreases. In order to enhance the rough handling of pattern acquisition in conventional methods and make the results more applicable and understandable, this study suggests a variation of the commonly used measurement as a solution to this issue. The developed algorithm provides two main innovations: (1) it fully takes into account Tobler's first law of geography and; and (2) it employs a novel calculating method to find common patterns. This approach considers the influence that a neighboring instance's distance and direction have on the relevance of its pattern. According to the experimental findings, the algorithm's common co-locations are more accurate and have higher fineness than co-locations mined using more conventional co-location techniques.

A different approach is presented in *A clique-based approach for co-location pattern mining* [7]. The authors of this paper observed that in current works for prevalent co-location pattern mining, the prevalence threshold is commonly a subjective value which determines whether a co-location pattern is prevalent (interesting) or not. In this case the user should start the algorithm many times to find the suitable prevalence threshold. This approach is not very efficient due to the expensive costs of identifying row-instances needed to find co-location patterns. By avoiding detecting row-instances of co-location patterns, this new method makes it much simpler to determine an appropriate prevalence level. To start, two effective schemes are created to produce comprehensive and accurate cliques. Then, these cliques are converted into a hash structure that is not dependent on the threshold for prevalence. Finally, the hash structure effectively determines the frequency of each co-location pattern.

Maximal dynamic spatial co-location pattern [4] proposes the concept of the dynamic spatial co-location pattern that can reflect the dynamic relationships among spatial features. It shows a technique to mine a small number of prevalent maximal dynamic spatial co-location patterns that can derive all prevalent dynamic spatial co-location patterns, which can improve the efficiency of obtaining all prevalent dynamic spatial co-location patterns. It also propose an algorithm for mining prevalent maximal dynamic spatial co-location patterns and two pruning strategies.

The aforementioned works are interesting and show interesting results to partially solve the problem of co-location pattern mining. However these algorithms employ a serialized approach, which is particularly time-consuming and has scalability issues. The purpose of this thesis is to deal with Big Data, thus we need a technique capable of handling huge amount of information, exploiting some kind of parallel calculation. *Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation* [1] achieves this goal, so we provide an in-depth presentation of this work in the next sections.

2.3 Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation

Yang et al. in [1] propose a new solution for spatial co-location mining that is designed to be efficient and available also for massive spatial datasets. This new approach is called Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation. This is a new SCPM algorithm, it computes participation index exploiting some novel concepts: **neighbor-dependency partition** and **column calculation**. Furthermore it introduces some pruning strategies to optimize the computation of participation index. This solution is very interesting for our purpose of analysing huge spatio-temporal datasets, all the new algorithms and solutions that we'll introduce in next chapters will be built upon Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation. So, in the next sections of this chapter, we explain how this SCPM algorithm is working.

2.3.1 Neighbor-Dependency Partition

One of the main idea of Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation is to divide the entire set of neighbor relationships in groups. Each group is a **partition** and the co-location mining task can be performed on each partition independently and so in parallel. [1] defines a process to divide the neighbor database into suitable partitions. Before diving into this process we need to introduce some concepts.

Given an instance o_i whose feature type is f_i, if there is any instance o_j of type f_j (where $f_i \neq f_j$) that satisfies the neighbor relationship $R(o_i, o_j)$ and f_j is less in alphabetical order than f_i, f_j is the ***prefix feature*** of instance o_i. The lexicographic order is used to compare features. It should be noted that some instances have not any prefix feature, while some instances can have more than one prefix feature. For example, considering figure 2.1, the neighbor set of instance B.3 is $neigh(B.3) = \{A.1, A.3, C.3, C.4\}$, thus A is a prefix feature of instance B.3 (because A is less in alphabetical order than B). While the neighbor set of B.1 is $neigh(B.1) = \{C.1, C.2, D.1, D.2\}$, so B.1 has not any prefix feature.

Given a neighbor database D the ***neighbor-dependency partition*** centered around feature f is denoted as *D(f)* and contains a subset of records of D that respect the following constraints:

1. All instances of feature f and their neighbor set.

2. The instance o_j and its neighbor set $Neigh(o_j)$, if the feature type of o_j is not *f*, but f is a prefix feature of o_j.

In table 2.2 is showed the neighbor-dependency partition centered around feature C. This partition contains all the instances of type C and their neighbors as well as the instances D.1 and D.2 with their neighbors because C is prefix feature of instances D.1 and D.2.

Instance	Prefix feature	Neighbor set
C.1	B	{B.1,B.2,D.1,D.2}
C.2	B	{B.1,B.2,D.1}
C.3	B	{B.3}
C.4	A,B	{A.3,B.3}
C.5		{}
D.1	B,C	{B.1,B.2,C.1,C.2}
D.2	A,B,C	{A.2,B.1,B.2,C.1}

Table 2.2. Neighbor dependency partition centered on feature C for example in figure 2.1

The Neighbor-Dependency partition has been introduced because it has an important property.[1] observed and proofed that given the neighbor dependency partition *D(f)*, for any co-location C starting with *f*, $C = \{f, f_1, f_2, f_3, ...f_k\}$ (considering that the features of f are ordered in ascending lexicographic order) all the row instances of C can be found inside *D(f)*. This intuition is very powerful. Co-locations starting with feature f can be discovered from the neighbor-dependency partition D(f) correctly. Any SCPM algorithm can be used on each partition to find prevalent co-locations with the same prefix. If we divide the neighbor database in to many neighbor dependency partitions we can perform the co-location mining task on each partition. It's important to highlight that the mining task on each partition is independent and so can be executed in parallel.

2.3.2 Column Calculation

Commonly, state of art methods for SCPM compute the participation index using table instances of patterns. The calculation of table instances is a very expensive operation, thus [1] developed a new strategy able to compute participation index of co-location patterns without requiring table instances. Let's make an example to show that the entire table instance is not always needed to compute participation index. Supposing we are mining the co-location pattern:

$$\{B, C, D\}$$

We assume that the table instance of the pattern is composed by these 6 rows:

$$\begin{gathered}\mathbf{\{B.1,C.1,D.1\}}\\ \{B.1, C.2, D.1\}\\ \{B.1, C.1, D.2\}\\ \{B.2, C.1, D.1\}\\ \{B.2, C.1, D.2\}\\ \mathbf{\{B.2,C.2,D.2\}}\end{gathered}$$

According to the definition of participation ratio and participation index, we need to count the distinct number of instances for each feature type that appear in the table instance (e.g number of distinct instances of *B, C* and *D*). In this case only 2 rows of table instance are needed: $\{B.1, C.1, D.1\}$, $\{B.2, C.2, D.2\}$. The column calculation for SCPM aims at computing this count without requiring the entire table instances of the mined co-locations.

To describe the column calculation strategy we need to introduce some new concepts.

Given a co-location C and feature $f \in C$, if the instance o of f is included in any row instance of C, o is called the ***participating instance*** of f in C. The collection of all participating instances of f in C is called participating instances set and it is denoted as *PIns(f, C)*. For example in figure 2.1 the instance B.1 is a participating instance of co-location $C = \{B, C, D\}$. The participating instance set of B in co-location $C = \{B, C, D\}$ is $Pins(B, \{B, C, D\}) = \{B.1, B.2\}$. According to this definition we can reformulate the equation for participation ratio as:

$$PR(f_i, C) = \frac{|PIns(f, C)|}{|N(f)|} \tag{2.4}$$

Where N(f) is the instance set of feature f (i.e. the distinct number of instances of type f). Then the participation index is still obtained with expression (2.3). The collection of all participating instances could be obtained from the table instances, but as we said before this is not the most convenient way. The main goal of column calculation is to reduce as much as possible the search space where participating instances are sought.

To introduce the concept of ***Candidate participating instance set*** we consider a co-location pattern C of size $k+1$ (with $k \geq 2$). If f is a feature of pattern C, we define C_k^f as the set of all size-k sub-patterns of C containing f. The candidate participating instance set $CPIns(f, C)$ for feature f of a size-(k+1) pattern C is defined as:

$$CPIns(f, C) = \cap_{C' \in C_k^f} PIns(f, C') \tag{2.5}$$

It can be easy verified that participating instance set for feature f of co-location C is a subset of candidate participating instance set:

$$PIns(f, C) \subseteq CPIns(f, C) \tag{2.6}$$

Thus to find $PIns(f, C)$ we only need to verify the instances contained in $CPIns(f, C)$. An instance $o \in CPIns(f, C)$ is a true participating instance of f in C if it exists a row instance of C that contains o. Considering co-location pattern $C = \{B, C, D\}$ the candidate participating instance for feature B is obtained as: $CPIns(B, \{B, C, D\}) = PIns(B, \{B, C\}) \cap PIns(B, \{B, D\})$.

A neighbor set $Neigh(o)$ of instance o can be divided in different groups by feature type. The ***grouped neighbor set*** of o on feature f is:

$$groupN(o,f) = \{o'|o' \in Neigh(o), o'.t = f\}$$

where o'.t is the feature type of instance o'. For example considering the example in figure 2.1, the neighbor set for B.1 is $Neigh(B.1) = \{C.1, C.2, D.1, D.2\}$. The grouped neighbor sets for B.1 are:
$groupN(B.1, C) = \{C.1, C.2\}$
$groupN(B.1, D) = \{D.1, D.2\}$

Finally, we can define the concept of ***Row Instance Search Space (RISS)***. Given a co-location C and an instance o, where $o.t \in C$; let $f \in C$ and $f \neq o.t$, the instance set of f whose instance may form a row instance of C with o is called the row instance search space of instance o on feature f, denoted as *RISS(o, f)*:

$$RISS(o,f) = \begin{cases} \text{group}\, N(o,f) \cap PIns(f,C), PIns(f,C) \text{ known} \\ \text{group}\, N(o,f) \cap CPIns(f,C), \text{ otherwise} \end{cases}$$

It's easy to verify that any instance of type f that may be a row instance of C with instance o, is contained in *RISS(o,f)*. So, the search space is reduced from $groupN(o,f)$ to $RISS(o,f)$. Once the search space is defined, to discover a row instances containing o we can perform the Cartesian product between all the *RISS* of o, then we check if these combination of instances are actually a row instance and we update the participating instance count for the spotted co-locations.
[1] proposes also a concrete implementation of this approach and it shows that SCPM based on Parallel Neighbor-Dependency Partition and Column Calculation outperforms other techniques. This new algorithm is scalable and is able to work with massive spatial datasets (300K instances). However this application is developed to work with data that contains only spatial dimension.
In the next chapter of this thesis we'll show that Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation can be extended to deal with spatio-temporal data.

2.4 Apache Spark

All the algorithms and experiments of this thesis are carried out using Apache Spark engine for large-scale data processing. Thus, before going into details of spatio-temporal mining, we overview the main concepts of Apache Spark technology. Apache Spark is designed to provide fast processing and scalability to deal with Big Data. To achieve this goal Spark extensively uses the main memory. We can compare Spark with other frameworks for processing large scale datasets, such as Apache Hadoop MapReduce. The main difference is that MapReduce involves many disk I/O especially in iterative jobs, while Spark stores and processes data in the main memory. Thus, Spark reaches processing speed way faster then Apache MapReduce.

The fundamental building blocks of Spark are the so called Resilient Distributed Datasets (RDDs). RDDs are stored in main memory (when possible) or local disk, they provide a distributed, fault-tolerant collection of data that can be processed in parallel across a cluster of computers. They are automatically rebuilt on machine failure.

Spark programs are written in terms of operations on RDDs. In particular Spark computing framework provides a programming abstraction and transparent mechanisms to execute code in parallel on RDDs. The complexities coming from fault-tolerance, scheduling and synchronization is hidden and managed automatically by the framework.

There are two types of operations that can be performed on RDDs: transformations and actions. **Transformations** take one or more existing RDDs and produce a new RDD. In order to optimize the execution plan Spark employees **lazy evaluation**, transformations on RDDs are not executed immediately when called. Transformations are lazy, they are not carried out right away. Instead, they are saved as a set of instructions to be carried out in response to a trigger for an action. Operations such as *map*, *flatMap*, *groupByKey* are examples of transformations employed in our applications.

Actions are procedures that trigger transformations, return results to the driver program, or send data to an external storage device. Actions are eager, which means they start the real calculation when called and are executed instantly. Actions are used to extract data from RDDs or to trigger side effects like saving data to disk or printing results. Some examples of actions are *collect* and *saveAsTextFile*.

Another fundamental concept in Spark's execution model is the Directed Acyclic Graph (**DAG**). It represents the order in which RDDs are transformed in a Spark application. Spark creates a DAG as a logical execution plan when transformations are applied to RDDs. The steps and actions necessary to compute the outcome are listed in the DAG. The DAG enables Spark to decide how data should be moved between stages and optimize the execution strategy.

Beyond support for RDDs operations, the Spark environment provides many other features. The framework actually contains many components (showed in figure 2.2).

Spark Core contains the basic functionalities of Spark exploited by all other

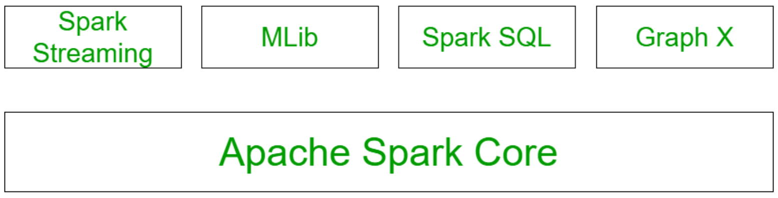

Figure 2.2. Spark main components

components (e.g. task scheduling, fault recovery, memory management) and provides APIs to create RDDs and to apply transformations and actions on them. Spark SQL is a component of Apache Spark that provides a unified interface for working with structured and semi-structured data. It combines the capabilities of a relational database management system (RDBMS) with the distributed computing power of Spark. Spark SQL provides support to manage DataFrames, distributed collections of data organized into named columns, similar to tables in a relational database. DataFrames provide a higher-level abstraction compared to RDDs. Spark SQL provides simple and expressive APIs for data manipulation, filtering, grouping, aggregation, and other common data operations. However for the algorithms we develop in this thesis, it's enough to work with RDDs.
Other components of the Spark framework (e.g Spark Streaming, MLlib, GraphX) are extremely powerful, but are not helpful for the applications discussed in this thesis.

Chapter 3

Spatio-temporal Co-location Mining

Spatio-temporal data mining aims at unraveling intricate patterns and insights present in datasets that evolve over both space and time. This approach empowers researchers, analysts, and decision-makers to discover hidden relationships, predict future trends, and make informed decisions across a spectrum of domains. For example in the field of urban planning and transportation management, spatio-temporal data mining allows for the analysis of traffic patterns over time, aiding in the identification of congested zones and peak hours. This information can be leveraged to optimize traffic flow, reduce commuting times, and enhance overall urban mobility. In epidemiology, spatio-temporal analysis plays a vital role in tracking the spread of diseases. By examining the geographical progression of outbreaks alongside temporal data, health authorities can allocate resources effectively, implement timely interventions, and mitigate the impact of contagions.
One of the way to find relationship between spatio-temporal data is co-location mining. The concept of spatial co-location mining discussed in the previous chapter can be extended to perform mining of spatio-temporal co-location patterns. In chapter 2 we dealt with data that have only a geographic locations without any temporal information. To perform spatio-temporal co-location mining we consider two kinds of spatio-temporal kind of data with two different purposes:

1. ***People trajectories***: the main goal is to analyze human behavior and find correlation among people and some other points of interest (shop, restaurant). The trajectory of each person is sampled, each sample is

represented by a dataset row that contains the people identifier, the geographic coordinates and the timestamp.

2. ***Traffic and weather events***: each row of the dataset represents an occurrence of a particular episode concerning traffic or weather domains. Each event is represented by a category name(e.g. accident, congestion, rain) a geographic location, a start and a final timestamp. The objective of the analysis is to find common interactions between different categories of traffic events and their correlation with weather events.

[9] developed an algorithm to perform mining of spatio-temporal data, in particular this application has been developed to mine people trajectories data, our contribution aims at improving the algorithm to make it more efficient and scalable to work with a huge volume of data. Then we'll also check the effectiveness of the new algorithm when working with traffic and weather events.

3.1 People trajectories

Starting from a set of trajectories that represent the movement of a group of people we want to develop a technique that is able to find people that are correlated. The idea is that if two or more people have some kind of relationship they will be found together in the same geographical area during same time. The trajectories of the people are represented by a set of sampled points, each point is a row of the dataset with the following structure:

$$< persond\_id, (latitude, longitude), timestamp >$$

For example in figure 3.1 are showed the trajectories of two people (P1 and P2) that have some spatially and temporally close points. These two people may be correlated, if they have enough close points the co-location pattern $[P1, P2]$ may be prevalent and would be spotted by the application. The representation of the data is different with respect to the structure used for spatial co-location mining. We can consider the $person\_id$ as if it corresponded to the feature of the spatial dataset. The objective of the spatial co-location mining was to find correlations among **categories** of entities, while in spatio-temporal co-location mining of trajectories we want to find correlation between people represented by $person\_id$. The main differences between spatio-temporal and spatial co-location mining are summarized in table 3.1.

	Spatial mining	**Spatio-temporal mining**
Feature	The descriptor of the entity (plant type)	The entity itself (person identifier)
Instance	An entity located in space	An entity located in space in a certain time
Goal of the co-location mining	Find correlation among categories	Find correlation among entities

Table 3.1. Differences between spatial and spatio-temporal mining of people trajectories

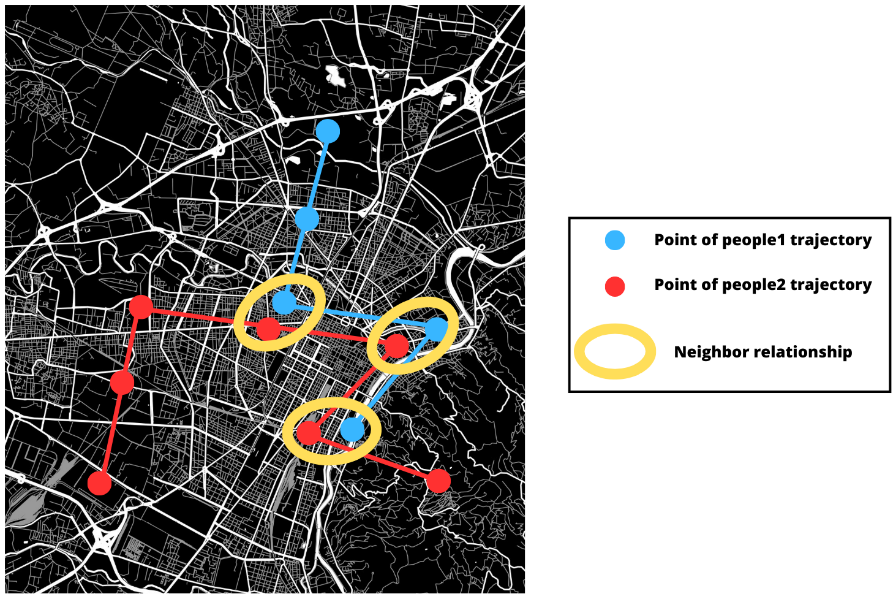

Figure 3.1. Trajectories of 2 people that may form a spatio-temporal pattern

In section 2.1 we introduced the concept of spatial neighbor relationship. Here we need to define the concept of ***spatio-temporal neighbors***. Two instances o_i and o_j are considered spatio-temporal neighbors if their distance is lower then a given threshold and they respect a time constraint for which the difference between their timestamp is under a certain threshold (*max_difference_time*). So, the spatio-temporal neighbor relationship is referred as:

$$\begin{aligned} R(o_i, o_j) \implies &\text{distance}(o_i, o_j) \leq \text{min\_distance} \\ &\text{AND } |o_i.ts - o_j.ts| \leq \text{max\_difference\_time} \end{aligned} \tag{3.1}$$

where $o.ts$ is the timestamp of the instance.

To perform spatio-temporal co-location mining we can exploit the technique "Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation" (described in chapter 2) with some updates. In particular the neighbors computation phase should be changed in order to consider also temporal information. Once the entire set of spatio-temporal neighbors has been computed we can employ parallel computation based on neighbor dependency partition and column calculation to extract co-location patterns. This idea is used in [9] to perform co-location mining of people trajectories. The architecture of this new technique is showed in figure 3.2.

As we can observe in figure 3.2, the application is composed of two main phases: the first one is responsible to find all the spatio-temporal neighbors of each point; the second phase discovers all the prevalent co-location patterns. This second phase exploits the parallel approach that has already been developed by [1] and presented in chapter 2 of this thesis. On the other hand, the algorithm to compute spatio-temporal neighbors is an innovation. This algorithm should find groups of instances that satisfies spatial and temporal neighbor constraints in an efficient and parallelizable way. To perform this task we may check all the pairs of points in the dataset, but this is not very efficient. Given a dataset of N timestamped points, to check each pair of points we need to do Cartesian product and this result in a complexity of $O(N^2)$. This operation is very expensive and may be not feasible when working with huge volume of data. To address this problem search data structures are needed.

[9] developed a parallelizable solution to compute spatio-temporal neighbors that exploits ballTree structures as showed in figure 3.3. This solution is developed exploiting Apache Spark framework, in particular the computation is expressed in terms of operations on RDDs. The idea of this architecture is to divide the original dataset into groups and perform neighbor search for each group. In order to group the instances, the temporal dimension is considered, the time is split into days, all the data points of a day form a group, which is taken as input form the BallTree. The **boundary points** between two consecutive days are grouped all together and are used to build another

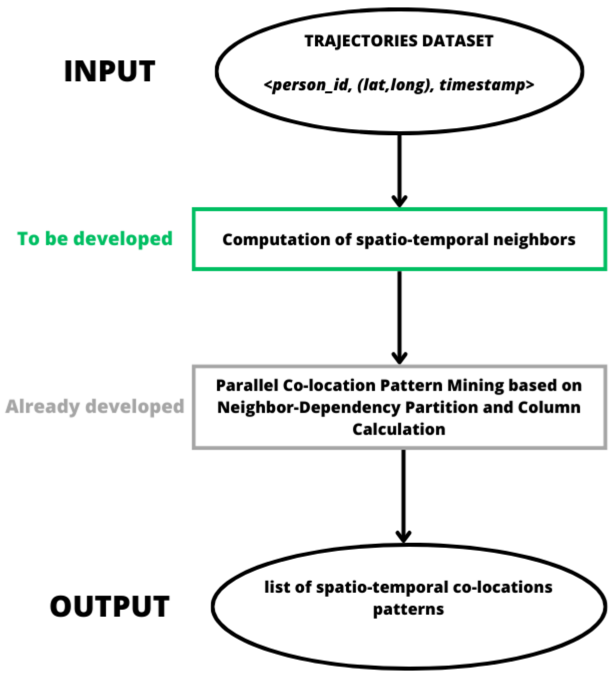
INPUT
TRAJECTORIES DATASET
<person_id, (lat,long), timestamp>
To be developed
Computation of spatio-temporal neighbors
Already developed
Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation
OUTPUT
list of spatio-temporal co-locations patterns

Figure 3.2. Design of spatio-temporal co-location mining algorithm

BallTree.

The BallTree is a space partitioning data structure used to perform efficient nearest neighbor search in multi dimensional spaces. In particular the data points are partitioned into nested balls and search operations are performed over these balls. In our algorithm the BallTree is used to find spatial neighbors. To compute the spatial distance the Haversine distance is used. The BallTree implementation of *skcit-learn* Python module is employed. This structure offers simple methods to specify a distance threshold to find neighbors. After applying the BallTree in parallel manner, for each point of the dataset we get the list of spatial neighbors. The RDD obtained as output of the BallTree has the form:

$$< o_i, [\mathit{list\ of\ spatial\ neighbors\ of\ } o_i] >$$

Then a *map* function is applied to check the time constraint, discarding all the couples that do not have the spatio-temporal neighbor relationship

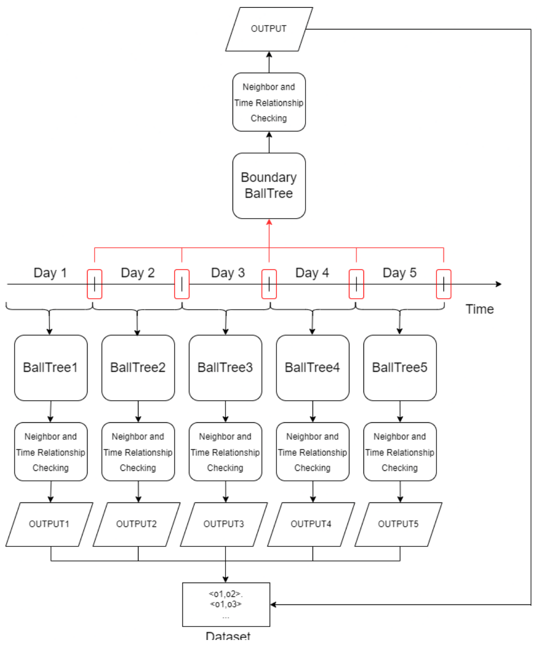

Figure 3.3. Parallel ballTree design to extract spatio-temporal neighbors

(3.1) . The spatial neighbors that do not respect the time constraint are simply removed from the list. After this operation we get an RDD with this structure:

$$< o_i, [\textit{list of spatio-temporal neighbors of } o_i] >$$

From this list the neighbor database is generated. Then the neighbor database is fed to the next phase of the algorithm which is able to find co-location patterns exploiting neighbor dependency partitions and column calculation.

3.2 Improving spatio-temporal neighbor computations

The original algorithm developed by [9] to find spatio-temporal neighbors is summarized in figure 3.4. This solution works good but can be optimized. In this section we present some alternative approaches to compute spatio-temporal neighbors in order to improve the efficiency of the process.
We first explore some different search data structure in place of BallTree. We experiment with R-tree, which is commonly used to index spatial dataset and geographic coordinates. The R-tree works with similar principles of BallTree,

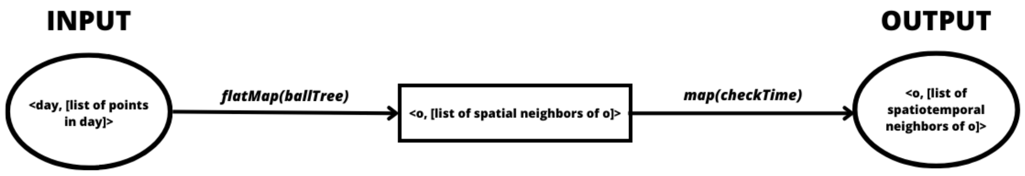
INPUT
OUTPUT
<day, [list of points in day]>
flatMap(ballTree)
<o, [list of spatial neighbors of o]>
map(checkTime)
<o, [list of spatiotemporal neighbors of o]>

Figure 3.4. Structure of algorithm to compute spatio-temporal neighbors

the main difference is that the R-tree partitions the data points in rectangular bounding box. This new solution is schematized in figure 3.5.

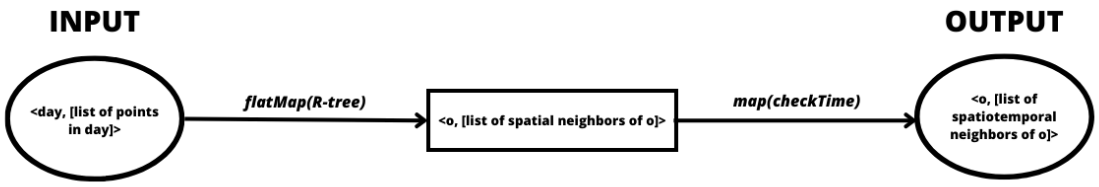
INPUT
OUTPUT
<day, [list of points in day]>
flatMap(R-tree)
<o, [list of spatial neighbors of o]>
map(checkTime)
<o, [list of spatiotemporal neighbors of o]>

Figure 3.5. Structure of algorithm to compute spatio-temporal neighbors with R-tree

Another optimization come from a simple observation. In the original algorithm (figure 3.4) the time constraint check is made through a *map* function. We can avoid this step and use just one single *flatMap* function that cares of filtering the neighbors that do not respect the time constraint. This solution is showed in figure 3.6.

In figure 3.7 we use two different trees data structure: one is used to optimize and search for spatial neighbors of each instance, another is used to compute the temporal neighbors. Then the two neighbor lists are "joined" to find the final spatio-temporal neighbor list. In this case, considering an instance o_i, another instance o_j is a spatio-temporal neighbor of o_i if it appears both in spatial and temporal list of neighbors of o_i.

Finally this idea of joining the result obtained with two different trees is taken in account also by the algorithm described by figure 3.8, but with a different approach. In this case the two trees are created and joined inside the

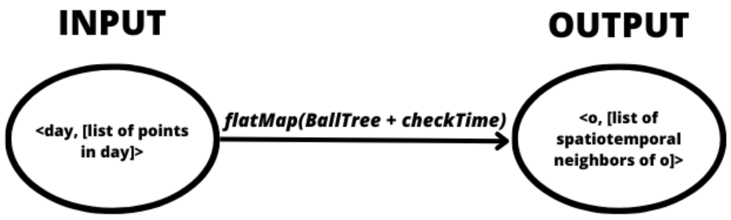
INPUT
OUTPUT
<day, [list of points in day]>
flatMap(BallTree + checkTime)
<o, [list of spatiotemporal neighbors of o]>

Figure 3.6. Structure of simplified algorithm to compute spatio-temporal neighbors

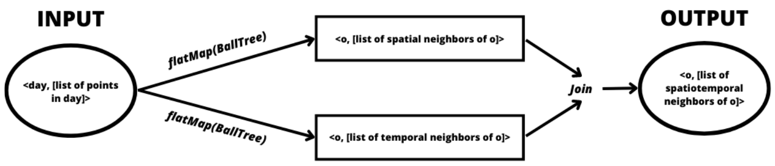
INPUT
OUTPUT
<day, [list of points in day]>
flatMap(BallTree)
flatMap(BallTree)
<o, [list of spatial neighbors of o]>
<o, [list of temporal neighbors of o]>
Join
<o, [list of spatiotemporal neighbors of o]>

Figure 3.7. Structure of algorithm to compute spatio-temporal neighbors joining results obtained with two trees

flatMap function, which emits directly the list of spatio-temporal neighbors. While in the previous version (3.7) two different RDDs were generated and then joined to find the list of spatio-temporal neighbors of each instance.

3.3 Evaluation and comparison of algorithms for spatio-temporal neighbors computation

In this section we make a comparison among the original algorithm for spatio-temporal mining developed by [9] and the new proposed solutions.

3.3.1 Experiment setup and dataset

To carry out our experiments we consider a GPS trajectory dataset called Geolife developed by Microsoft Research Asia ([12]). A GPS trajectory is composed by a set of points associated with latitude, longitude, a timestamp and an identifier of the person. 182 people have been involved to register 17,621 trajectories. Most of them have been captured in Beijing, China from

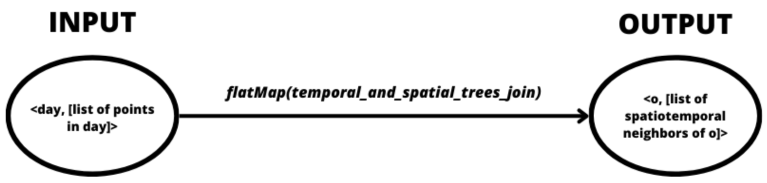
INPUT
OUTPUT
<day, [list of points in day]>
flatMap(temporal_and_spatial_trees_join)
<o, [list of spatiotemporal neighbors of o]>

Figure 3.8. Structure algorithm to compute spatio-temporal neighbors joining results obtained with two trees created inside flatMap

April 2007 to August 2012. The sampling does not follow a regular rate, the points are logged every 1 ∼ 5 seconds or every 5 ∼ 10 meters. In figure 3.9 the people with the highest number of sampled points are reported.

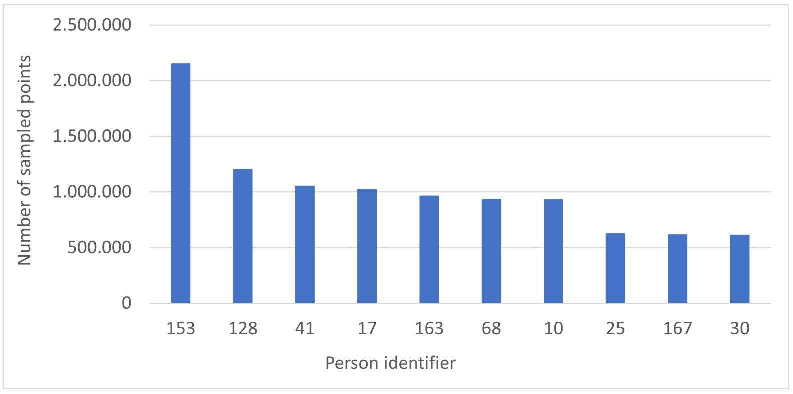

Figure 3.9. Number of points labeled with spatial and temporal location for people with highest sampled points

Many trajectories of Geolife dataset have been labeled with *transportation mode* (e.g. walk, driving, bus) however in our evaluation we focus only on correlation between people without considering *transportation mode*. In the study of [9] they considered a reduced version of Geolife, they used only one-sixth of the original dataset with a total number of 3.3M points. In this work we show that actually the algorithm is able to scale with the entire Geolife dataset which contains almost 24 M points. All the applications described in this thesis have been implemented using PySpark, a Python library for Apache Spark, and run on BigData@Polito Cluster [13]. This cluster is operated by SmartData@Polito group, it was originally composed of 36 nodes with a maximum capacity of 220 TB of data (around 650 TB of raw disk space). The complete infrastructure now has more than 1,700 CPU cores, 19 TB of RAM memory and 8 PB of raw storage available, however these resources are shared and split between many users. The BigData@Polito Cluster provides support to store and process huge quantities of data and allows the execution of many common open-source frameworks. In particular, we exploit Hadoop Distributed File System (HDFS) that allows distributed storage and replication of data. We make use of Hadoop Spark to efficiently process in parallel each of the small pieces of a huge volume of data. We employ JupyterLab to debug and submit our applications. Spark jobs are submitted to the BigData@Polito Cluster, then results are saved in the logs which are accessed through Hadoop User Experience (HUE).

3.3.2 Results

The described algorithm is able to find co-location patterns of frequently close people. The number of detected patterns depends on the values of some parameters: *min_participation_index* (prevalence threshold), *distance* (spatial threshold) and *max_difference_time* (temporal threshold). A deep discussion and interpretation of these results is carried out in [9]. In figure 3.10 we just report a summary of these results, we show the number of discovered co-locations patterns with different prevalence thresholds.

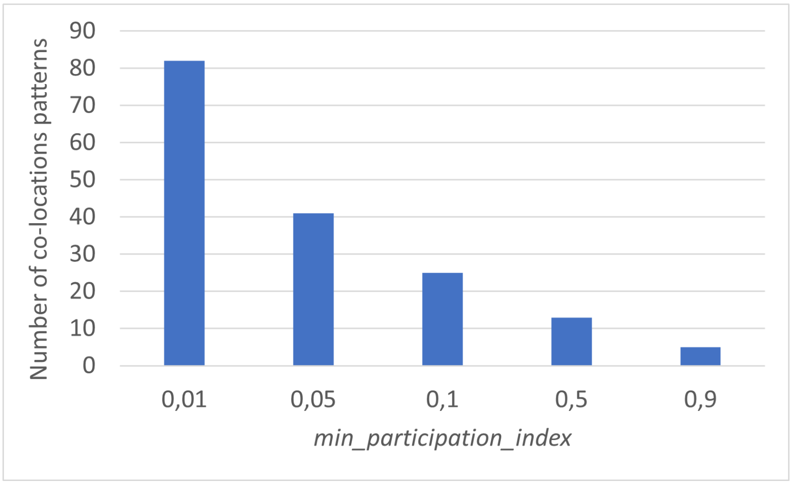

Figure 3.10. Number of co-locations patterns discovered with fixed values of *distance=0.5 km* and *max_difference_time=5 minutes*

In this thesis we want to establish how the different approaches to compute spatio-temporal neighbors affect the performance of co-location mining algorithm. In figure 3.11 we report the time required by the algorithm for co-location mining of trajectories when using different solutions to compute spatio-temporal neighbors. In particular we considered all the solutions described in the previous section 3.2, each of these solutions is labeled with the number of the figure that represents it.

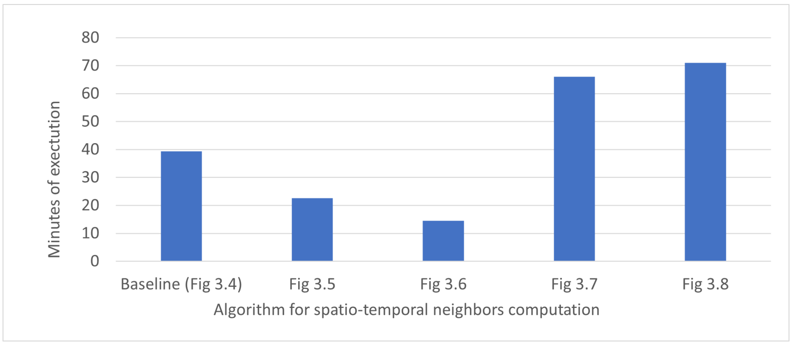

Figure 3.11. Execution time required by spatio-temporal co-location mining using different approaches to compute spatio-temporal neighbors

The solution 3.6 considerably improves the baseline, it requires 41 % less time. This new solution to compute spatio-temporal neighbors does not disrupt the original baseline idea, however it needs less operations in terms of numbers of RDDs generated. The algorithm in 3.6 allows co-location mining to scale to bigger datasets. On the other hand, the approaches in 3.7 and 3.8 are not helpful, they introduce some overhead due to join computations.

3.4 Analysis of traffic and weather events

The algorithm developed for spatio-temporal mining of people trajectories can be simply adapted for spatio-temporal mining of traffic events. The types of data that represent traffic and weather events have the following structure:

$$< event\_type, (latituted, longitude), startTimestamp\_endTimestamp >$$

event_type is a category type of events concerning traffic, urban or weather domain (e.g. Accident, Construction, Snow). The main difference of this traffic events with data that represent trajectory is that traffic events have an instant of start and end, while trajectory samples are associated with a single timestamp. However this dissimilarity is simply filled considering *startTimestamp* as the reference time to check time constraints when mining traffic and weather events data. We apply the new algorithm, originally developed for spatio-temporal mining of people trajectories, also on a dataset containing millions of traffic and weather events located in space and time. We perform this analysis to check the effectiveness of the algorithm in a different field and to evaluate if our application is actually able to disclose some important correlation among these kinds of events.

3.4.1 LSTW dataset

To perform our analysis we consider Large-Scale Traffic and Weather (LSTW) Events Dataset published in [11]. This dataset contains about 30.8 million traffic and 5.6 million weather events that cover the contiguous USA from August 2016 to the end of Dec 2020. This data are retrieved from different sources (the US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors within the road-networks). Traffic event is a spatio-temporal entity, it is associated with location and time and a category type. The event types available in LSTW are:

- **Accident (A)**: a common type, which may involve one or more vehicles, and could result in fatality.
- **Broken-Vehicle (BV)**: refers to the situation when there is one (or more) disabled vehicle(s) in a road.
- **Congestion (CG)**: refers to the situation when the speed of traffic is slower than the expected speed.
- **Construction (CS)**: an on-going construction or maintenance project on a road.
- **Event (E)**: situations such as sports event, concerts, or demonstrations, that could potentially impact traffic flow.
- **Lane-blocked (LB)**: refers to the cases when we have blocked lane(s) due to traffic or weather condition.
- **Flow-incident (FI)**: refers to all other types of traffic entities. Examples are broken traffic light and animal in the road.

The distribution and relative frequency of the event categories is showed in 3.2.

The weather events in LSTW dataset are collected using the measures captured by weather stations located in airports. The taxonomy is defined as:

Entity Type	**Raw Count**	**Relative Frequency**
Accident	2,638,226	8.6%
Broken-Vehicle	645,713	2.1%
Congestion	24,541,194	79.7%
Construction	728,138	2.3%
Event	47,497	0.2%
Lane-Blocked	501,583	1.6%
Flow-Incident	1,692,486	5.5%
Total	30,794,837	100%

Table 3.2. Distribution of traffic events in LSTW dataset

- **Severe-Cold**: extremely cold condition, with temperature $\leq -23.7°$.
- **Fog**: low visibility condition as a result of fog or haze.
- **Hail**: solid precipitation including ice pellets and hail.
- **Rain**: rain of any type, ranging from light to heavy.
- **Snow**: snow of any type, ranging from light to heavy.
- **Storm**: the extremely windy condition, where the wind speed is at least 60 km/h.
- **Precipitation**: any kind of solid or liquid deposit, but different from snow or rain. This was a generic label we frequently observed in raw weather data.

Weather events unlike traffic events can't be located in a specific geographic point. Each weather event is associated to the airport code where it has been registered. Then to compare weather and traffic data, the airport code of the closest airport station is assigned also to traffic events. A traffic and weather events are "spatially close" if they have the same airport code. This is a simple way to find correlations between weather and traffic data, however it's not so accurate. The main limitation is that airport stations are distributed in a sparse way over the territory. Each airport covers a wide geographic area, it's not sure that all the points associated with an airport actually undergo the same weather events.

3.4.2 Summary of results

The algorithm for traffic and weather co-location mining can be run with different parameter values:

- *distance*: the spatial distance expressed in kilometers within which two events are considered close.
- *max_difference_time*: the maximum difference expressed in minutes between the starting moment of 2 events within which these events are considered close.
- *min_Participation_Index*: the threshold to filter not relevant co-locations. Co-location patterns with Participation Index lower than *min_Participation_Index* are discarded.

We first check how many co-locations are obtained considering both traffic and weather events (LSTW dataset). In figure 3.12 number of discovered co-location patterns is reported with different values of *distance* and *min_Participation_Index* keeping fixed the value of *max_difference_time* to 30 minutes.

The results of figure 3.12 show that the values of *distance* affect the number of co-locations found only when the value of *min_Participation_Index* is small (≈ 0.01) as showed in details in figure 3.13. Furthermore we can observe that there is not any co-location pattern that has $Participation\_Index > 0.07$. While when we considered GeoLife trajectories dataset we found patterns with *Participation_Index* up to 0.9 (figure 3.10). This gap is due to the fact that LSTW dataset is composed of few generic categories of events (about 20) which contains a huge number of instances (in the order of 1 M). It's unlikely to found patterns with high values of *Participation_Index* considering this kind of dataset. An interesting dataset for co-location mining should have more specific categories of events with less events. For example the category "Event (E)" represents a generic public event that may affect traffic flow. It would be preferable if this category was split into many categories according the specific event (e.g. concert, public demonstration, sport event) as well as the number of people involved.

In table 3.3 the co-location patterns with highest value of participation index are shown (*distance*=1.5 km; *max_difference_time*=30 min). All these patterns are composed by a traffic event category and a weather event category. Co-locations pattern among traffic events category have lower values

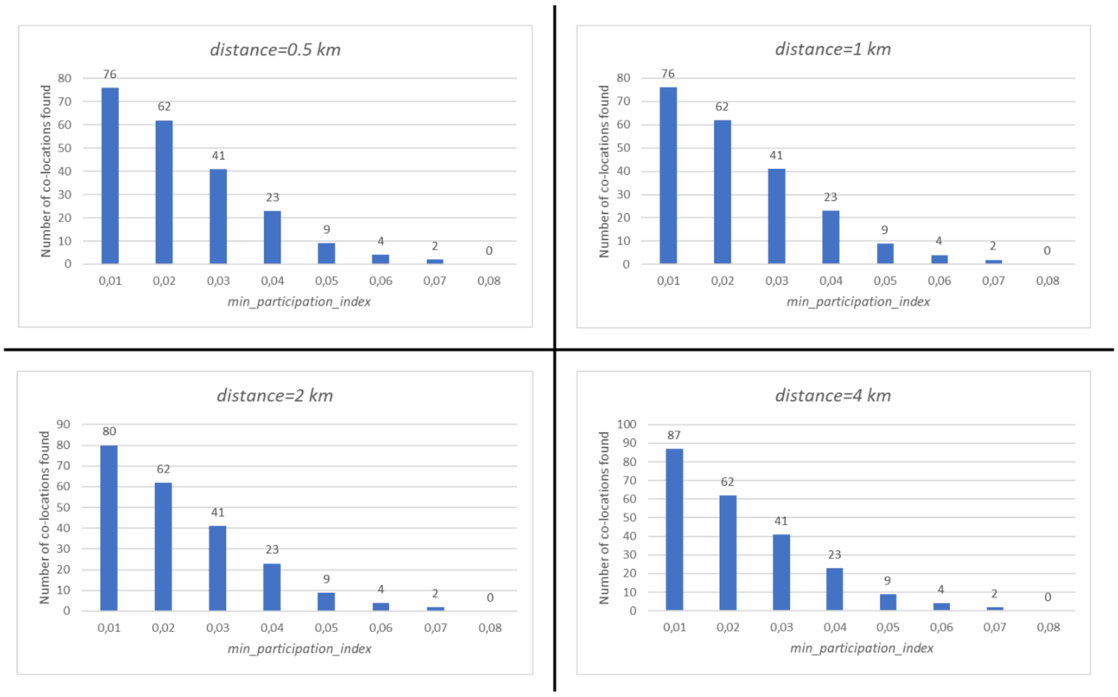

Figure 3.12. Traffic and weather: number of co-locations found with different values of distance and minimum participation index

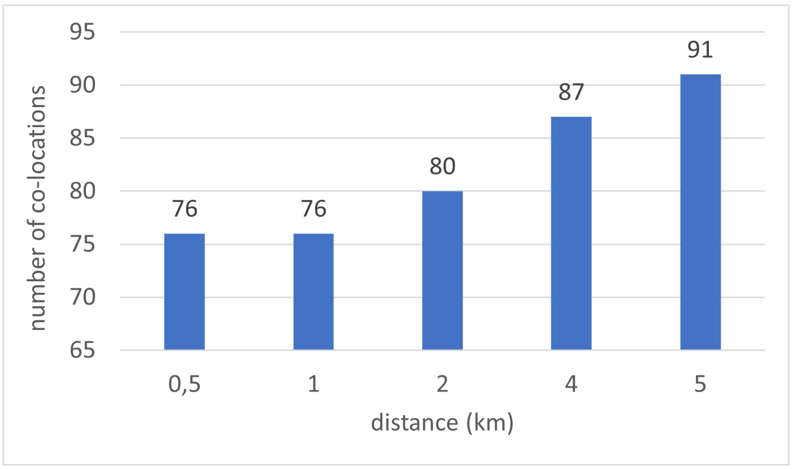

Figure 3.13. Traffic and weather: number of co-locations found with different values of distance and fixed value of minimum participation index = 0.01

of participation index. Not all the co-location patterns in this list are significant. For example *[Construction, Severe cold]* it's difficult to explain, while

[Accident, Light rain] may be meaningful. Rain usually makes roads slippery increasing the probability of an accident in the area. We can also notice that all the patterns in the table 3.3 are of size 2. As expected co-location patterns with bigger size are less likely and have lower values.

Co-location pattern	**Participation index**
[Construction, Severe cold]	0.075
[Flow incident, Severe cold]	0.074
[Event, Severe cold]	0.067
[Construction, Light snow]	0.059
[Congestion, Severe cold]	0.058
[Flow incident, Light snow]	0.057
[Flow incident, Storm]	0.056
[Construction, Storm]	0.055
[Accident, Severe cold]	0.047
[Accident, Light rain]	0.044

Table 3.3. Top 10 traffic-weather co-location patterns mined with *distance*=1.5 km; *max_difference_time*=30 min

In table 3.4 size-3 co-locations patterns with highest participation index are reported (algorithm run with *distance*=1.5 km; *max_difference_time*=30 min). As expected the participation indexes of size-3 patterns are lower than participation index of size-2 patterns. However some of these patterns make sense.

As we discussed in section 3.4.1 the weather events of LSTW dataset cover a wide geographic area. Each weather event has a huge neighbor set, this could lead to misleading co-location patterns. Thus, in order to obtain more accurate results, we carry out spatio-temporal co-location mining considering only traffic events from LSTW dataset. We expect that traffic co-locations have lower values of participation index with respect to traffic-weather co-location patterns. In figure 3.14 are showed the numbers of co-location patters discovered for different values of *min_participation_index* when considering only traffic events.

For *min_participation_index* set to 0,01 we got only two traffic co-location patterns. While for the same value of *min_participation_index* we observed 80 traffic-weather co-locations patterns. This is due to the fact that the weather events have a huge number of neighbors and this lead to co-location patterns with higher values of participation index.

Co-location pattern	**Participation index**
[Congestion, Flow Incident, Light Rain]	0.0102
[Congestion, Flow Incident, Moderate Fog]	0.0101
[Congestion, Flow Incident, Precipitation]	0.0061
[Congestion, Construction, Light Rain]	0.0034
[Congestion, Construction, Moderate Fog]	0.0025
[Congestion, Construction, Flow Incident]	0.0024
[Construction, Event, Severe Fog]	0.0018
[Congestion, Construction, Severe Fog]	0.0012
[Congestion, Construction, Precipitation]	0.0011
[Construction, Event, Moderate Snow]	0.0010

Table 3.4. Top 10 size-3 traffic-weather co-location patterns mined with *distance*=1.5 km; *max_difference_time*=30 min

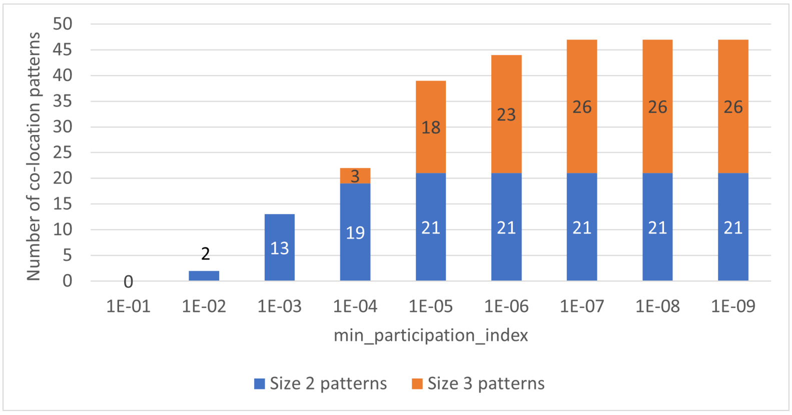

Figure 3.14. Traffic only: number of co-locations found with different values of participation index threshold. *distance* fixed to 1.5 km; *max_difference_time* fixed to 30 minutes

Again, co-locations patterns with size 3 are discovered only for lower values of *min_Participation_Index*.
In table 3.5 are reported the most prevalent traffic-only co-location patterns ordered by participation index.

To complete the analysis we compare the execution time when changing input parameters. *min_Participation_Index* acts as a threshold to filter the discovered co-locations, thus it doesn't affect the execution time. While, we expect that the execution time increase as *distance* and *max_difference_time*

Co-location pattern	Participation index
[Congestion, Flow-Incident]	0.00561
[Accident, Congestion]	0.00100
[Accident, Flow-Incident]	0.00079
[Broken-Vehicle, Flow-Incident]	0.00032
[Lane-Blocked, Flow-Incident]	0.00024
[Accident, Lane-Blocked]	0.00019
[Construction, Flow-Incident]	0.00018
[Broken-Vehicle,Congestion]	0.00017
[Accident,Broken-Vehicle]	0.00016
[Broken-Vehicle,Lane-Blocked]	0.00015

Table 3.5. Top 10 traffic-only co-location patterns mined with *distance*=1.5 km; *max_difference_time*=30 min

growth. Indeed, higher values of these parameters lead to bigger neighbor database increasing the supports of co-location patterns. This is confirmed in figure **??**.

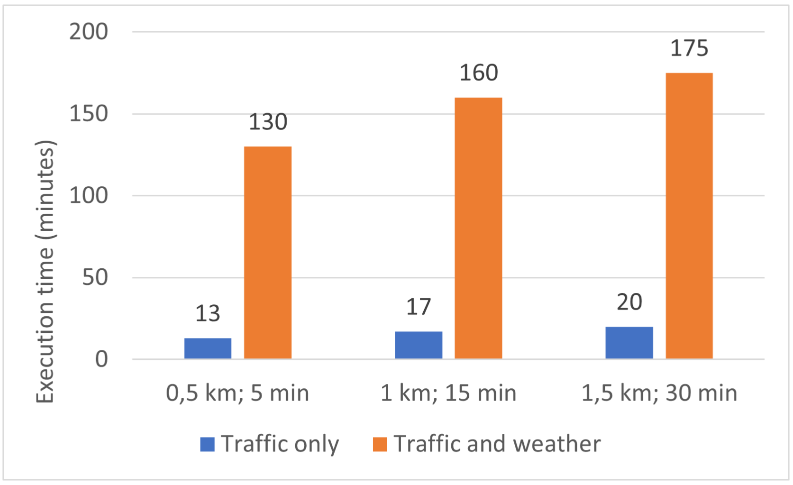

Figure 3.15. Execution time required for co-location mining with different values of *distance* and *max_difference_time*

The execution is slower for the algorithm that considers both traffic and weather events. This is because, as discussed before, weather events are associated with a large spatial area and so they have a huge neighbor set. Neighbor database generated mining both traffic and weather events is way bigger than neighbor database obtained mining only traffic events. However the developed algorithm is able to scale up and is able to handle even this huge amount of data.

Chapter 4

Co-location Sequence Mining

In this chapter we dive into the core contribution of this work. We introduce the problem of "Co-location Sequence Mining". First we provide a formal definition of Co-location Sequences and what do they represent. Then we define some new metrics to measure the relevance of the sequences, underlying the pros and cons of all the new measurements. Later we propose an algorithm for mining these kinds of sequences and computing the suitable metrics. The application is developed to be highly scalable and efficient since it should be able to work with a huge volume of data. Finally the main results of co-location sequence mining applied on a dataset containing traffic and weather events are reported.

4.1 Problem definition

To introduce the concept of co-location sequence, we consider two simple co-locations $[A, B]$ and $[A, C]$ that follow the definition given in 2.1. These 2 co-locations inform us that events of type A are usually spatially and temporally close to events of type B and type C. We aim at finding a new concept to express the correlation between these two co-locations. We introduce the sequence of co-locations that is denoted as:

$$[A, B] \Rightarrow [A, C] \tag{4.1}$$

Any instance of the sequence of co-location respect these constraints:

- $[A, B]$ and $[A, C]$ are spatial or spatio-temporal co-location patterns (as defined in 2.1)
- The event of type A is referred as ***aggregator***, it is the same event in the first and second co-location.
- The event of type C should begin after the event of type B
- The event of type C starts after the event of type B but before a certain amount of time defined by a parameter *max_difference_time*: $C.t - B.t < max\_difference\_time$

With this definition we are affirming that when an event of type A is close temporally and spatially to an event of type B is likely to observe an event of type C in the same geographical area that starts immediately after B. In other words the fact that an event A is close to an event of type B implies that an event of type C is going to happen in the same area. We consider valid also sequences of type:

$$[A, B] \Rightarrow [A, B] \tag{4.2}$$

if the two instances of type B represent different events. For example considering the situation in figure 4.1 at time 1 we can observe a "Construction" (maintenance, or re-paring project in a road) that is close temporally and spatially to a "Congestion" event. Than immediately after at time 2 the same "Construction" will be close spatially and temporally to "Accident" that has started immediately after the "Congestion". If there are many cases of "Construction" near to "Construction" and then followed by "Accident" we'd observe a co-location sequence of type $[Construction, Congestion] \Rightarrow [Construction, Accident]$. To determine if the co-location sequence is actually meaningful we use some metrics discussed in 4.2.
The event of type "Congestion" and "Accident" are close to the aggregator event "Construction", however they could not be close to each other. Thus, the standard co-location mining described in the previous chapters may not be able to discover any kind of relationship between "Congestion" and "Accident". The co-location sequence mining is necessary to find this hidden correlations.

It's important to remark that a sequence can be composed of more than 2 co-locations. We define the sequence ***size*** as the number of co-locations in the sequence. For example a size-3 co-location sequence is:

$$[A, B] \Rightarrow [A, C] \Rightarrow [A, D] \tag{4.3}$$

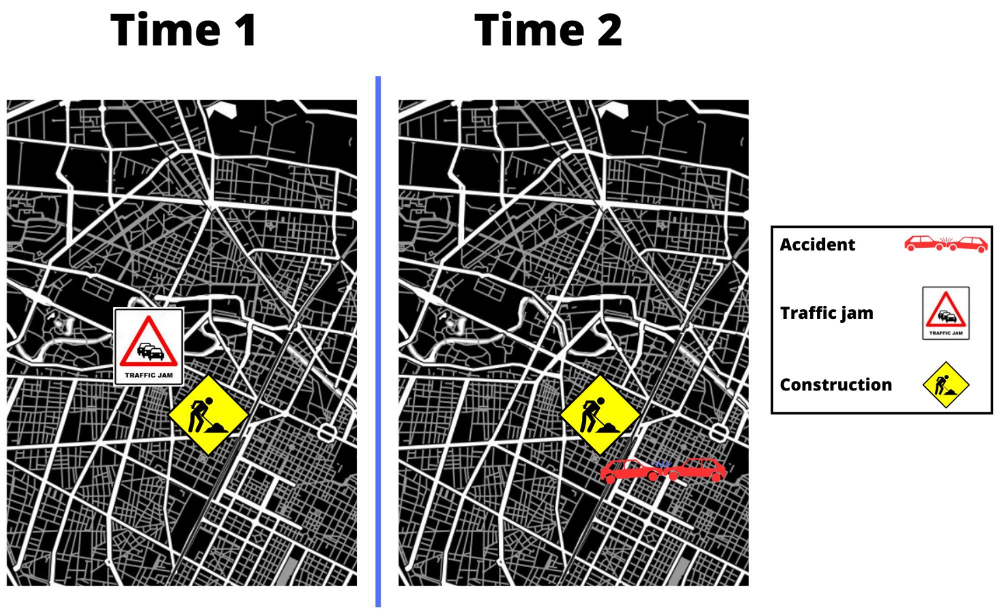

Figure 4.1. Traffic events that may generate a co-location sequence

In this case the event A works as aggregator, the event C should start immediately B and, in turn, the event D should start immediately after C. Up to here we considered only examples of sequences composed of size-2 co-locations, however the definition is still valid if the sequence is composed of co-locations with size>2. For example:

$$[A, B, C] \Rightarrow [A, D] \tag{4.4}$$

is a possible co-location sequence. The event of type A is the aggregator and is the same in both co-locations. The event of type D should start immediately after both B and C.

Given a dataset of events located in time and space, we can find all possible co-location sequences starting from a set of spatio-temporal co-locations retrieved using some algorithm (e.g the one described in chapter 3) . From this set we generate all possible candidate co-location sequences. For each candidate sequence we check if it is matched by some group of instances in the original dataset and we compute some metrics. So, in our idea co-location sequence mining should be implemented as post processing with respect to spatio-temporal mining.

4.2 Metrics

In order to understand which sequences are meaningful we introduce some new metrics to take into account different aspects. Since there is not any previous study in the field of co-location sequence mining, we propose new various approaches to compute support and confidence of the sequences. Actually we propose our custom definition of support and confidence, which may be different from the standard definition. In 4.4 we'll discuss about actual interpretability and significance of these new metrics.

4.2.1 Metrics 1

Considering a co-location sequence $[A, B] \Rightarrow [A, C]$ we define **support1** as the number of *distinct* groups of events that respect the rule imposed by sequence. The *distinct* refers to the first co-location of the sequence ($[A, B]$), if there are 2 or more groups with the same couple of events for the first co-location of the sequence, these will contribute to increase support just once. For example suppose that there are 4 groups of instances that match the sequence $[A, B] \Rightarrow [A, C]$:

- $[A.1, B.1] \Rightarrow [A.1, C.2](support + 1)$
- $[A.1, B.2] \Rightarrow [A.1, C.2](support + 1)$
- $[A.1, B.2] \Rightarrow [A.1, C.7](support + 0)$
- $[A.1, B.1] \Rightarrow [A.1, C.3](support + 0)$

$support1([A, B] \Rightarrow [A, C]) = 2$
support1 for $[A, B] \Rightarrow [A, C]$ defines how many times the couple of neighbor events $[A, B]$ is followed by the couple of events $[A, C]$ where the instance of A should be the same e the event C should start after B with a difference of time that does not exceed the constraint *max_difference_time*

Considering the sequence $[A, B] \Rightarrow [A, C]$, we define **confidence1** as the ratio between $support1[A, B] \Rightarrow [A, C]$ divided by the support of the first co-location (total number of *distinct* couples of instances [A,B] that are spatially and temporally close)

$$\textbf{confidence1}\ ([A, B] \Rightarrow [A, C]) = \frac{\text{support1}\ ([\mathbf{A}, \mathbf{B}] \Rightarrow [\mathbf{A}, \mathbf{C}])}{\text{support}([\mathbf{A}, \mathbf{B}])} \tag{4.5}$$

where *support*$([A, B])$ is the number of neighbor couples $[A, B]$. As well as in the co-location mining phase even in the co-location sequence mining phase two or more instances of an event are considered neighbors if respect the 2 following criteria:

- The two instances are **spatially** close: the distance between A and B is lower than a parameter referred to as *distance_threshold*
- The two instances are **temporally** close: the absolute value of the difference between the beginning of the two events is lower than a parameter referred to as *max_difference_time*

High value of confidence1 for sequence $[A, B] \Rightarrow [A, C]$ means that is likely that a couple of event $[A, B]$ is followed by $[A, C]$ where A is the same event in the two couples and the event C starts immediately after B. Although it's not true that all the sequences with high values of confidence1 are meaningful, in fact there may be cases where a sequence has support1 equals 1 or another similar small number and however has a confidence1 $\approx$ 1. In this situation the sequence should not be considered significant, the value of confidence1 is misleading. For this reason we explore other metrics to measure the importance of the sequences.

4.2.2 Metrics 2

Considering the sequence of co-location $[A, B] \Rightarrow [A, C]$ we introduce a new definition of support and confidence. To define the new support we consider only the instances that are not working as aggregator: B and C . The **support2** is the number of couples $[B, C]$ that are spatially and temporally close (C starts after B, and their time difference in terms of C.start-B.start is lower than *max_difference_time)*.

$$\begin{aligned} \textbf{support2}([A, B] \Rightarrow [A, C]) &= \text{number of [B,C] spatially close} \\ \textbf{s.t} \quad & \text{start C - start B} < \textit{max\_difference\_time} \end{aligned} \tag{4.6}$$

We define **confidence2** as the ratio between support2 and the number of instances of the first type:

$$\textbf{confidence2} \ ([A, B] \Rightarrow [A, C]) = \frac{\text{support2}([A, B] \Rightarrow [A, C])}{\text{number of events}(B)} \tag{4.7}$$

The confidence2 $([A, B] \Rightarrow [A, C])$ is simply the likelihood that an event of type B is followed by an event of type C with respect to the total number of events of type B. For example suppose that there are 4 groups of instances that match the sequence $[A, B] \Rightarrow [A, C]$:

- $[A.1, B.1] \Rightarrow [A.1, C.2](support + 1)$
- $[A.2, B.1] \Rightarrow [A.2, C.2](support + 0)$
- $[A.3, B.1] \Rightarrow [A.3, C.7](support + 1)$
- $[A.1, B.1] \Rightarrow [A.1, C.3](support + 1)$

$support2([A, B] \Rightarrow [A, C]) = 3$
This metric has a big limitation: different sequences can have the same value of confidence2. For example if we consider two sequences:

$$([A, B] \Rightarrow [A, C])$$
$$([D, B] \Rightarrow [D, C])$$

these 2 sequences have the same values of confidence2 because they only differ for the aggregator event feature, that do not contribute to support2. Thus confidence2 as it is, is not suitable to evaluate the importance of the sequences. Although confidence2 is useful because can be compared with other metrics. An interesting comparison is the ratio between confidence1 and confidence2:

$$\frac{\text{confidence1}}{\text{confidence2}} \tag{4.8}$$

To explain how this ratio can be interpreted let's make an example. If we consider the simple co-location sequence $([A, B] \Rightarrow [A, C])$ a high ratio between confidence1 and confidence2 means that the event of type A increases the probability that an event of type B followed by C happen.

4.2.3 Metrics 3

Confidence1 aims at measuring the likelihood of a co-location pattern to be followed by another pattern. We introduce a new confidence with a different purpose. We define **confidence3** as the ratio between support1 and the number of instances of the aggregator feature type.

$$\textbf{confidence3}\ ([A, B] \Rightarrow [A, C]) = \frac{\text{support1}([A, B] \Rightarrow [A, C])}{\text{number of events}(A)} \tag{4.9}$$

We can interpret confidence3 ($[A, B] \Rightarrow [A, C]$) as the likelihood that the event A implies that there is an event B in its neighborhood followed by an event of type C. A limitation of this metric is that the denominator may be a large number when there are lot of events from the same feature label. In these cases the ratio of confidence3 results in very small number that may be difficult to interpret.

In the next section we introduce an algorithm for co-location sequence mining which is able to compute all these new metrics.

4.3 Algorithm

In this section we present a new algorithm to make co-location sequence mining in a scalable, parallelizable and efficient manner. Our solution works as post processing with respect to the co-location pattern mining phase. The algorithm for co-location sequence mining needs as input the results of the co-location pattern mining, in particular the entire list of retrieved co-location patterns and the neighbor database are needed. The algorithm for co-location sequence mining we developed is composed of different stages:

1. **Generate candidate sequences**: starting from the pre-computed list of all the co-location patterns, this phase produces the list of all potential co-location sequences.
2. **Find instances that match candidate sequences**: starting from the list of all potential co-location sequences and the neighbor set, this phase searches and finds all the groups of instances that respect the sequence rule. It's the core and most demanding phase.
3. **Compute metrics**: once we found all co-location sequences with the list of instances that match them, the algorithm provides as output all the required metric values.

The design structure of the algorithm is reported in figure 4.2. In the following sections we describe in details the implementation of the application.

4.3.1 Generate candidate sequences

This algorithm takes as input the list of spatio-temporal co-locations generated by the process described in chapter 3 and produces a list of sequences of

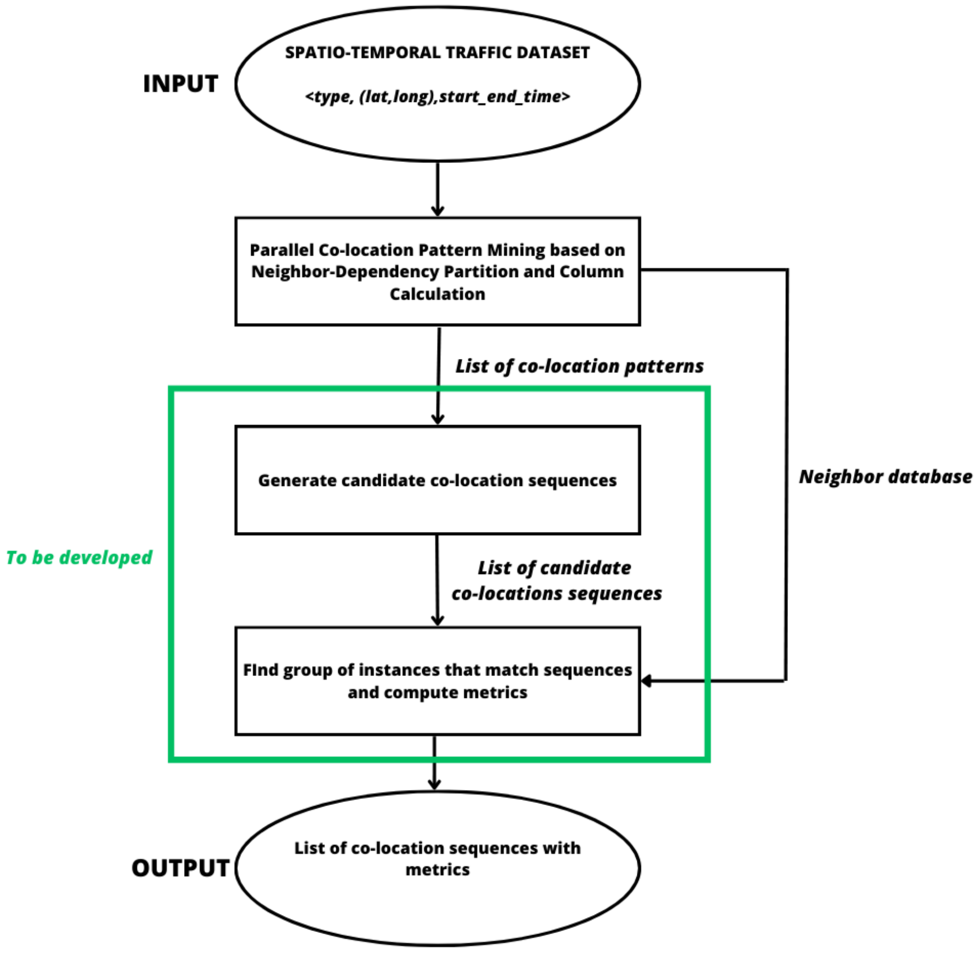
INPUT
SPATIO-TEMPORAL TRAFFIC DATASET
<type, (lat,long),start_end_time>
Parallel Co-location Pattern Mining based on Neighbor-Dependency Partition and Column Calculation
List of co-location patterns
Neighbor database
To be developed
Generate candidate co-location sequences
List of candidate co-locations sequences
FInd group of instances that match sequences and compute metrics
OUTPUT
List of co-location sequences with metrics

Figure 4.2. Architecture of the application for co-location sequence mining

co-locations as defined in the previous section 4.1. The candidate sequences are generated considering all the possible combination of all the co-locations with each other. The algorithm takes as input a list of co-locations grouped around each feature type (f). Each row of the list has this format:

$$<\text{f}, [\textit{list of co-locations starting with } f]>$$

For example if we consider a set of feature type *{A,B,C,D}* the input list of co-locations may be:

(A, [[A, B], [A, C], [A, B, C]])
(B, [[B, C], [B, D], [B, C, D]])
(D, [])
(C, [[C, D]])

To generate all the candidate sequences we can work separately for each input row (for each group of co-locations). All the co-location patterns inside a group has at least one feature in common with all the other patterns inside the group. For example considering the group centered around feature A:

(A, [[A, B], [A, C], [A, B, C]])

all the three co-location patterns have the feature A in common. The candidate sequences are generated by the Cartesian product among all the three patterns, the result would be:

$$([A, B] \Rightarrow [A, C])$$
$$([A, B] \Rightarrow [A, B, C])$$
$$([A, C] \Rightarrow [A, B])$$
$$([A, C] \Rightarrow [A, B, C])$$
$$([A, B, C] \Rightarrow [A, B])$$
$$([A, B, C] \Rightarrow [A, C])$$

However it's not enough to compute the Cartesian product in each group of co-location pattern. Before starting with the Cartesian product we need to make a consideration: if we take a co-location pattern $[A, B]$ there is not any temporal concept of "before" and "after". $[A, B]$ rule simply says that the feature A and B are usually temporally and spatially close. So, in co-location mining there is not any difference between $[A, B]$ and $[B, A]$. While in co-location sequence mining this difference matters. So we define a flatMap function *full_colocations* that works in this way. It takes as input the rows of the co-locations. For example for the group centered around feature A:

(A, [[A, B], [A, C], [A, B, C]])

the output of *full_colocations* is the set of rows:

(A, [A, B])
(B, [B, A])
(A, [A, C])
(C, [C, A])
(A, [A, B, C])

(B, [B, A, C])
(C, [C, A, B])

Then a *group_by* function is used to recreate a set of groups with the following structure:

<f, [*full list of co-locations that starts with f*]>

Considering again the set of input co-locations:

(A, [[A, B], [A, C], [A, B, C]])
(B, [[B, C], [B, D], [B, C, D]])
(D, [])
(C, [[C, D]])

The result after applying *full_colocations* and *group_by* would be:

(A, [[A, B], [A, C], [A, B, C]])
(B, [[B, A], [B, A, C], [B, C], [B, D], [B, C, D]])
(C, [[C, A], [C, A, B], [C, B], [C, B, D], [C, D]])
(D, [[D, B], [D, B, C], [D, C]])

Once we get the list of full co-locations grouped by feature we can apply *generate_candidate* that works parallelly for each group of co-location patterns and produce all the possible combination between co-locations. Considering the same example, the output of candidate sequences would be:

(['B', 'A'], ['B', 'A']) (['B', 'A'], ['B', 'A', 'C']) (['B', 'A'], ['B', 'C']) (['B', 'A'], ['B', 'D']) (['B', 'A'], ['B', 'C', 'D']) (['B', 'A', 'C'], ['B', 'A'])

...

(['D', 'B', 'C'], ['D', 'B']) (['D', 'B', 'C'], ['D', 'B', 'C']) (['D', 'B', 'C'], ['D', 'C']) (['D', 'C'], ['D', 'B']) (['D', 'C'], ['D', 'B', 'C']) (['D', 'C'], ['D', 'C'])

The algorithm to find the candidate sequences has been developed with PySpark, the main steps of the process are reported in figure 4.3.

Pseudocode for this phase of the algorithm is the following:

```
fullColocations=input.flatMap(full_colocations).groupByKey()
output=fullColocations.flatMap(generate_candidate)
```

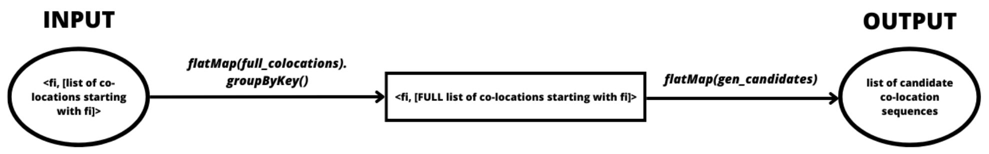
INPUT
<fi, [list of co-locations starting with fi]>
flatMap(full_colocations).groupByKey()
<fi, [FULL list of co-locations starting with fi]>
flatMap(gen_candidates)
OUTPUT
list of candidate co-location sequences

Figure 4.3. Architecture of the application to generate candidate co-location sequences

4.3.2 Find instances that match candidate sequences

This algorithm takes as input the list of **candidate co-location sequences** and the **neighbor dependency partition** of all the features. The neighbor dependency partitions are simply extracted from the neighbor database which is computed during the co-location mining phase. While the list of candidate sequences is the result of the *generate candidate sequences* algorithm described in previous section 4.3.1. The goal of this new phase is to find all the groups of instances that match the candidate sequences in order to extract the required metrics. The instances are events represented by a tuple:

('feature type', 'start timestamp_end timestamp')

To make clear the purpose of the algorithm, it should goes through the neighbor dependency partition and find all the groups of instances that respect the constraints for one of the candidate sequences. This process should be repeated for all the candidate sequences. We can process each candidate sequence independently. In order to visualize the expected output of the algorithm, let's suppose we are mining the candidate sequence $([A, B] \Rightarrow [A, C])$. The result of *Find instances that match candidate sequences* would be a tuple with this structure:

$$< s, [\textit{list of group of instances that match } s] >$$

For example:

$$< ([A, B] \Rightarrow [A, C]), [\textit{list of instance groups that match } ([A, B] \Rightarrow [A, C])] >$$

Once we got the list of instance groups that match the co-location sequence, we can simply compute all the metrics defined in previous section 4.2.

The general idea of this application is reported in the schema 4.4. To build the algorithm we make an **assumption**: the list of candidate sequences is small enough to be contained in some kind of local variable. So as first step we define a function *group_candidate_sequences* that starts from the list of

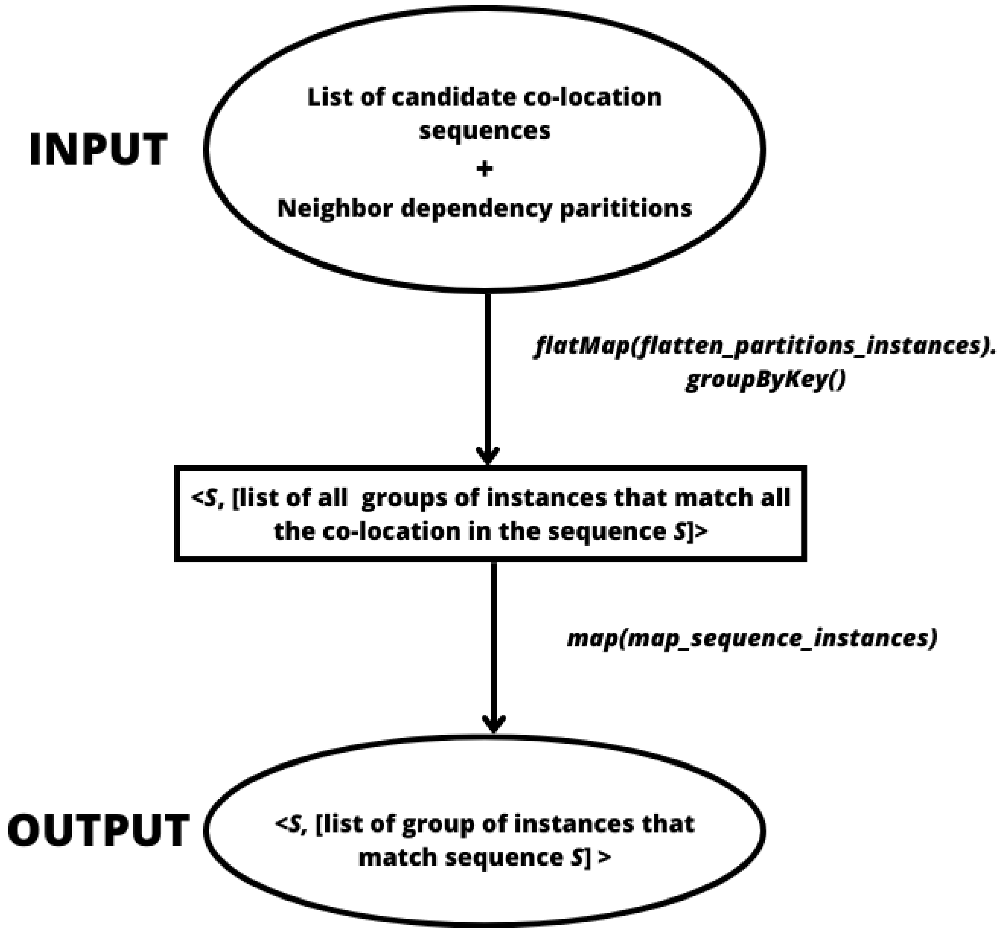
INPUT
List of candidate co-location sequences
+
Neighbor dependency parititions
flatMap(flatten_partitions_instances).
groupByKey()
<S, [list of all groups of instances that match all the co-location in the sequence S]>
map(map_sequence_instances)
OUTPUT
<S, [list of group of instances that match sequence S] >

Figure 4.4. Architecture of the algorithm *Find instances that match candidate sequences*

candidate sequences and generate a dictionary that has as key the feature (f) and as value the list of candidate sequences that contains f in one of the co-locations of the sequence. For example if we suppose that we have only two candidate sequences

$$([C, D] \Rightarrow [C, B])$$
$$([C, A] \Rightarrow [C, B])$$

The ***sequence__dictionary*** generated by *group__candidate__sequences* would be:

$$\{$$
$$A : [([C, A] \Rightarrow [C, B])]$$

$$B : [([C, D] \Rightarrow [C, B]), ([C, A] \Rightarrow [C, B])]$$
$$C : [([C, D] \Rightarrow [C, B]), ([C, A] \Rightarrow [C, B])]$$
$$D : [([C, D] \Rightarrow [C, B])]$$
$$\}$$

The *sequence_dictionary* is sent to the cluster as a shared variable and then can be accessed by all the functions in the PySpark code. The reason why the list of candidates has been organized in this kind of dictionary will be clear later. Once the candidate sequences has been handled and converted into the suitable dictionary, the algorithm starts to manipulate the RDD representing the neighbor dependency partition (*partitions_RDD*). This RDD is computed during the co-location mining phase, each row represents a partition centered around feature type (f) and has the format:

$$< f, [< o, [\textit{list of neighbor of instance } o] >] >$$

where o is an instance of type f or f is a prefix of o. The first operation on the *partitions_RDD* is a flatMap referred as *flattern_partitions_instances*. Each row of the *partitions_RDD* has a feature f as key. We retrieve all the candidate sequences that contain f simply extracting the list of sequences from the pre-defined *sequence_dictionary* using f as key. For each neighbor dependency partition (for each row of *partitions_RDD*) centered around f the function *flattern_partitions_instances* emits a set of rows with this format:

$$< s, [\textit{list of groups of close instances that match one or more co-location inside sequence } s] >$$

where s is the candidate sequence. In the list of neighbors are considered only the groups where at least one instance is of type f (center of the partition). This is done in order to avoid to overload the data structures used in the intermediate steps with redundant information. To make an example let's suppose that the list of candidate sequences for feature B is:

$$\textit{sequence\_dictionary[B]} = [([C, D] \Rightarrow [C, B]), ([B, A] \Rightarrow [B, C]), ([D, A] \Rightarrow [D, B])]$$

the function *flatten_partitions_instances* that takes as input the neighbor partition centred around B would emits 3 rows:

1. (($[C, D] \Rightarrow [C, B]$), [list of instances of close (C,B) found inside B-dependency partition])

2. ($([B, A] \Rightarrow [B, C])$, [list of instances of close (B,A) and list of instances of close (B,C) found inside B-dependency partition])
3. ($([D, A] \Rightarrow [D, C])$, [list of instances of close (D,B) found inside B-dependency partition])

After the *flatten_partitions_instances* a function to group all the instances is applied. A *groupByKey* is employed using the co-location sequence as key. After the *groupByKey* we get an RDD that has the rows composed like this:

$$< s, [list\ of\ all\ the\ groups\ of\ instances\ that\ match\ ALL\ the\ co\text{-}locations\ in\ the\ sequence] >$$

Following the example, after the *groupByKey* we get:

1. ($([C, D] \Rightarrow [C, B])$, [list of instances of close (C,B) and (C,D) found inside **all dependecy partitions**])
2. ($([B, A] \Rightarrow [B, C])$, [list of instances of close (B,A) and (B,C) found inside **all dependecy partitions**])
3. ($([D, A] \Rightarrow [D, C])$, [list of instances of close (D,A) and (D,C) found inside **all dependecy partitions**])

At this point the function *map_sequence_instances* is applied independently for each row (each row has a sequence as key and a list of groups of close instances as value). This function generates all the possible combinations comparing each group of close instances with all the other groups. If the compared groups respect the sequence (the feature type of the considered groups are the same of the sequence) it means that they potentially match the co-location sequence. Although this is not enough, since the function should also check the temporal constraints given by the definition of co-location sequence itself. Let's see how *map_sequence_instances* is working with an example. We suppose that we are mining the sequence:

$$([A, B] \Rightarrow [A, C])$$

This sequence is matched by couple of group of close instances such as:
$[('A', '2016-08-14T00:07:00\_2016-08-16T00:08:00'), ('B', '2016-08-14T00:07:01\_2016-08-16T00:09:00')] \Rightarrow [('A', '2016-08-14T00:07:00\_2016-08-16T00:08:00'), ('C', '2016-08-14T00:08:02\_2016-08-16T00:07:30')]$
This set of instances respect the constraints given by the definition of co-location sequence:

- The instance of aggregator element A is the same in the 2 groups: ('A', '2016-08-14T00:07:00_2016-08-16T00:08:00') belongs to both first and second group of close instances.
- The instance of type C in the second group starts immediately after the instance of type B: ('C', '2016-08-14T00:08:01_2016-08-16T00:07:30') starts 1 minute after ('B', '2016-08-14T00:07:01_2016-08-16T00:09:00')

Figure 4.5 summarizes how the data have been partitioned for this phase of the algorithm. The algorithm "Find instances that match candidate sequences" takes as input the neighbor database for a set of feature $F = \{f_1, f_2, ..f_n\}$ and a set of candidate co-location sequences $S = \{s_1, s_2, ..s_n\}$. The neighbor database is divided into N partitions centered around each feature of the feature set F. For each partition centered around f the flatMap function *flattern_partitions_instances*. This function finds a portion of cliques that match the co-locations contained inside the candidate sequences. Only the co-location sequences that include f are considered. Then a *groupByKey* is needed to gather the entire list of cliques that matches all the co-locations of each sequence. Afterwards a map function (*map_sequence_instances*) can operate independently for each sequence. It compares each cliques with all the other to find group of cliques that match the entire sequences. Once we found all the group of instances matching the sequences, we can simply compute all the metrics introduced in section 4.2.

4.4 Experimental evaluation: co-location sequence mining of traffic events

In this section we discuss the configuration and the results we got experimenting the algorithm for co-location sequence mining. Our main goal is to check whether is possible to discover some interesting rules and correlation between traffic and weather events applying the algorithm for co-location sequence mining. We also need to verify that the metrics we introduced are useful and coherent.

4.4.1 Experimental setup

The dataset considered is the LSTW dataset, already described in section 3.4.1. We also create three new dataset as subset of LSTW:

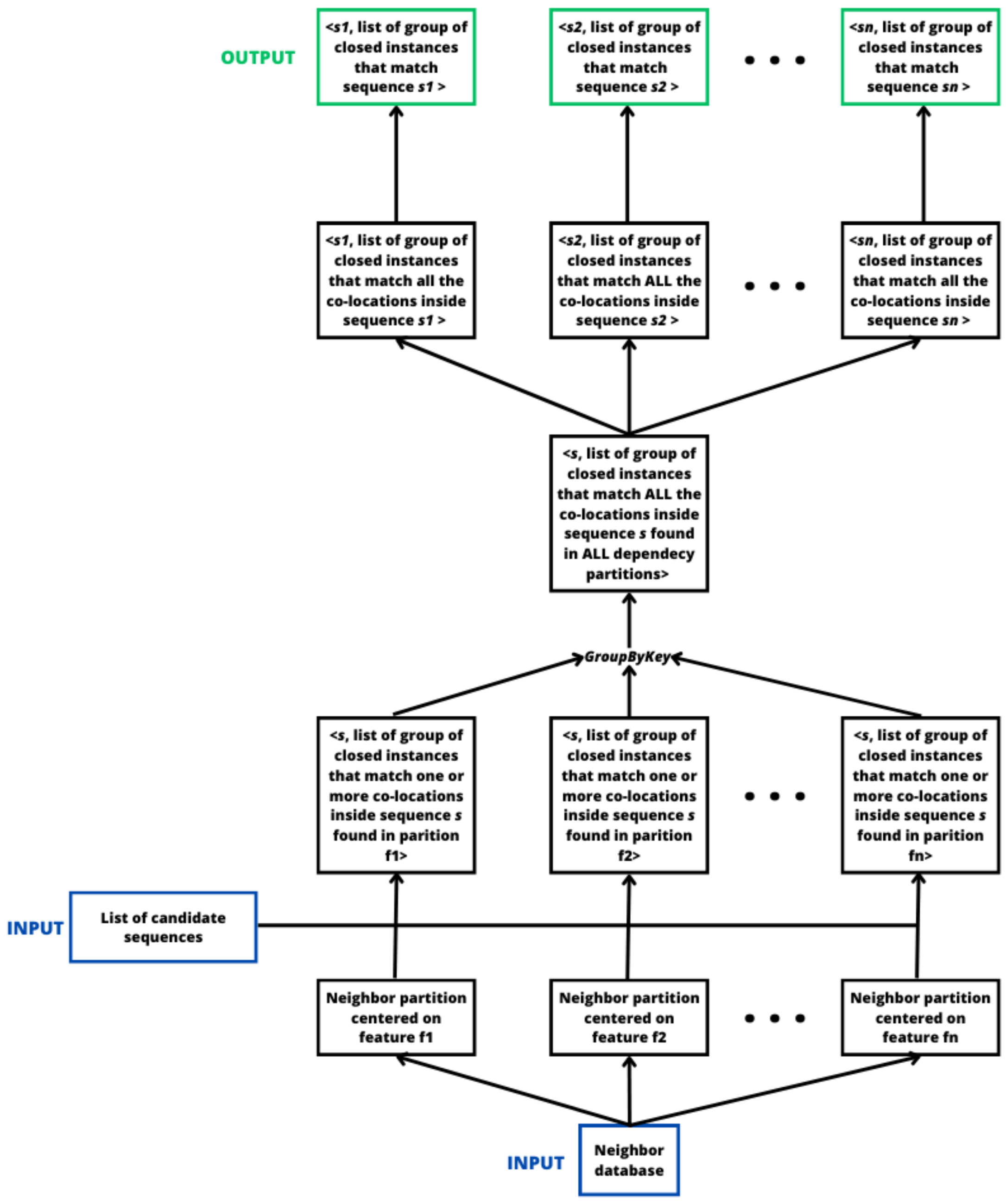
OUTPUT
<s1, list of group of closed instances that match sequence s1 >
<s2, list of group of closed instances that match sequence s2 >
<sn, list of group of closed instances that match sequence sn >
<s1, list of group of closed instances that match all the co-locations inside sequence s1 >
<s2, list of group of closed instances that match ALL the co-locations inside sequence s2 >
<sn, list of group of closed instances that match all the co-locations inside sequence sn >
<s, list of group of closed instances that match ALL the co-locations inside sequence s found in ALL dependecy partitions>
GroupByKey
<s, list of group of closed instances that match one or more co-locations inside sequence s found in parition f1>
<s, list of group of closed instances that match one or more co-locations inside sequence s found in parition f2>
<s, list of group of closed instances that match one or more co-locations inside sequence s found in parition fn>
INPUT
List of candidate sequences
Neighbor partition centered on feature f1
Neighbor partition centered on feature f2
Neighbor partition centered on feature fn
INPUT
Neighbor database

Figure 4.5. "*Find instances that match co-location sequences*" algorithm: manipulation of data

- **California**: contains all the events of the LSTW dataset registered in the state of California (5.4 M events)
- **Texas**: contains all the events of the LSTW dataset registered in the state of Texas (2.7 M events)
- **New York**: contains all the events of the LSTW dataset registered in New York state (2.9 M events).

We applied our analysis in these three datasets and in the original LSTW dataset that covers the whole territory of USA. The original LSTW dataset that contains events from all the contiguous United states is referred as "**All states**". The algorithm has been developed using PySpark and run on the BigData@Polito Cluster following the approach previously described in section 3.3.1. The algorithm can be run changing different parameters both in co-location mining and sequence co-location mining phases. The main parameters that can be set to run the algorithm are:

- *distance*: the spatial distance expressed in kilometers within which two events are considered close.
- *max_difference_time*: the maximum difference expressed in minutes between the starting moment of 2 events within which these events are considered close.
- *min_Participation_Index*: the threshold to filter not prevalent co-locations. Co-location patterns with Participation Index lower than *min_Participation_Index* are discarded.
- *sequence_length*: the number of co-locations that are considered to compose a sequence.
- *only_traffic*: if set true only traffic events are considered otherwise also weather events are taken in account.

For the aforementioned problems of the LSTW weather data (see 3.4.1), we decided to consider only traffic events for co-location sequence mining (i.e. we set *only_traffic=true*) In 3.4 we showed that is difficult to obtain co-locations with high value of *participation index* when mining LSTW traffic events. For experimental purpose we decided to set *min_Participation_Index*=$1^{-}10$ in order to obtain a significant number of co-location patterns as input of the co-location sequence mining phase. Due to the limitations in term of computational and memory resources we were able just to perform mining of 2-size sequences (*sequence_length=2*). Regarding *distance* and *max_difference_time* we consider reasonable values in the range of respectively: 0.5km - 1.5 km and 5min - 30 min.

4.4.2 Results

Figure 4.6 summarizes the analysis and result we performed. The figure shows the number of co-location sequences discovered, when changing *distance* and *max_difference_time*. The found sequences are divided in three bands: the sequences in the first blue band have support1<10; the sequences in the second yellow band have $100 < \text{support1} \geq 10$; the sequences in the third black band have support1>100 . The analysis is performed for all the 4 datasets. In general, the algorithm is able to extract a significant number of sequences. As expected the number of supported sequences grows as the distance and temporal parameters get bigger. Nevertheless most sequences have a limited number of support1. To observe sequences with support1>100, large values of *distance* and *max_difference_time* are needed (1.5 km, 30 minutes).

The trends are similar for all the 4 considered datasets. Though, higher

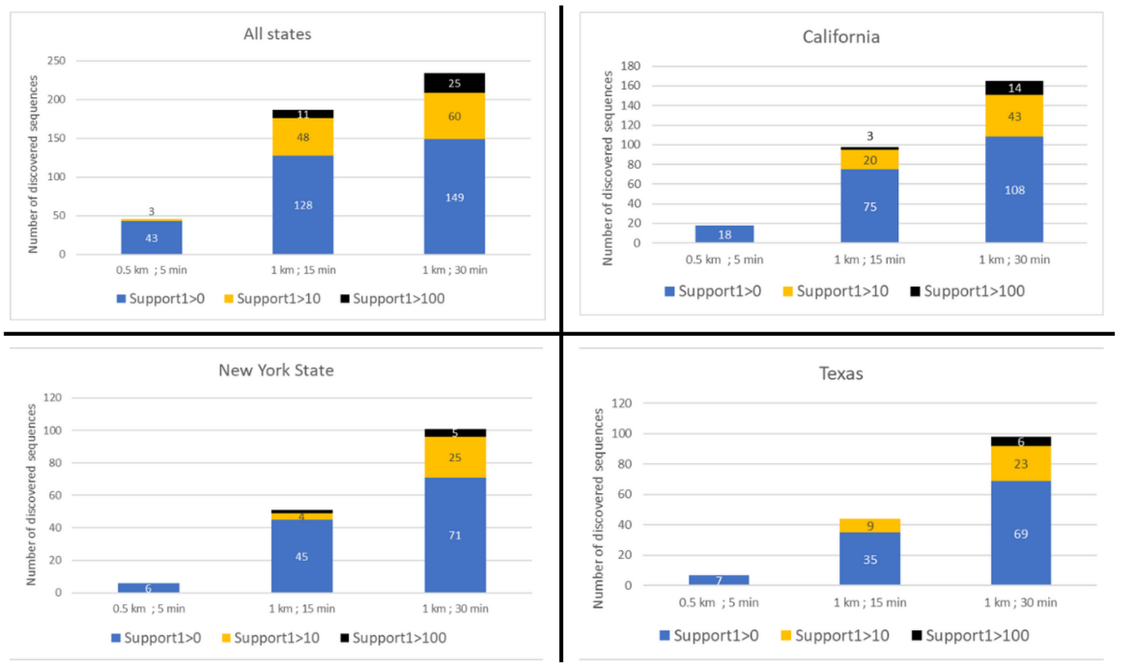

Figure 4.6. Summary of obtained results

numbers of co-locations sequences are found considering "All states" as dataset, since it contains a greater number of events. In order to better visualize the type of results that are found by the co-location sequence mining, in table 4.1 we report an extract of the outcomes of the algorithm run with (*max_difference_time*=15 min ; *distance*= 1 km) and All states dataset. In the table are reported the top 20 sequences ordered by value of confidence1 in descending order.

sequence	**Support1**	**Confidence1**	**Confidence2**	**Confidence3**
$([FI, CG] \Rightarrow [FI, CG])$	10944	0,314311152	0,010035005	0,006466228
$([FI, BV] \Rightarrow [FI, CG])$	47	0,255434783	0,002236288	2,78E-05
$([CS, BV] \Rightarrow [CS, CG])$	1	0,25	0,002236288	1,37E-06
$([BV, FI] \Rightarrow [BV, CG])$	42	0,24852071	0,020572696	6,50E-05
$([CS, FI] \Rightarrow [CS, CG])$	19	0,240506329	0,020572696	2,61E-05
$([FI, CS] \Rightarrow [FI, CG])$	18	0,24	0,000652349	1,06E-05
$([FI, A] \Rightarrow [FI, CG])$	177	0,231372549	0,003174103	0,00010458
$([A, FI] \Rightarrow [A, CG])$	155	0,218002813	0,020572696	5,88E-05
$([LB, A] \Rightarrow [LB, CG])$	21	0,21	0,003174103	4,19E-05
$([FI, E] \Rightarrow [FI, CG])$	3	0,2	0,003263364	1,77E-06
$([LB, FI] \Rightarrow [LB, CG])$	24	0,19047619	0,020572696	4,78E-05
$([E, A] \Rightarrow [E, CG])$	1	0,166666667	0,003174103	2,11E-05
$([FI, LB] \Rightarrow [FI, CG])$	20	0,157480315	0,002133246	1,18E-05
$([E, FI] \Rightarrow [E, FI])$	9	0,155172414	0,004642874	0,000189486
$([E, CG] \Rightarrow [E, CG])$	22	0,141935484	0,010035005	0,000463187
$([BV, A] \Rightarrow [BV, CG])$	19	0,137681159	0,003174103	2,94E-05
$([BV, LB] \Rightarrow [BV, CG])$	5	0,135135135	0,002133246	7,74E-06
$([CG, FI] \Rightarrow [CG, FI])$	4116	0,125836926	0,004642874	0,000167718
$([E, FI] \Rightarrow [E, CG])$	7	0,120689655	0,020572696	0,000147378

Table 4.1. Co-locations results with 1km and 15 min thresholds

A complete description of the event labels is provided in section 3.4.1 .
We can observe that the most supported pattern is$([Flow-Incident, Congestion] \Rightarrow [Flow-Incident, Congestion])$ (referred as $([F, CG] \Rightarrow [F, CG])$) which also has the highest value of confidence1. This means that this is the most significant sequence found. It' important to note that not all the sequences in the table with high values of confidence1 are so meaningful. For example the sequences $([CS, BV] \Rightarrow [CS, CG])$ and $([E, A] \Rightarrow [E, CG])$ are supported just once, so they can't be considered meaningful even if they have an high value of confidence1. Thus, as expected, confidence1 is a good metric but

is not enough to judge if a co-location sequence is relevant. It's not easy to define a threshold in terms of confidence1 that discriminates interesting and not interesting co-location sequences.

The same configuration with *max_difference_time*=15 min and *distance*= 1 km is considered also to compare confidence1 and confidence2. Figure 4.5 shows a scatter graph, where each point in the chart is a co-location sequence, the horizontal axis represents confidence1 values, while vertical axis contains confidence2 values. Confidence2 values are much lower than confidence1.

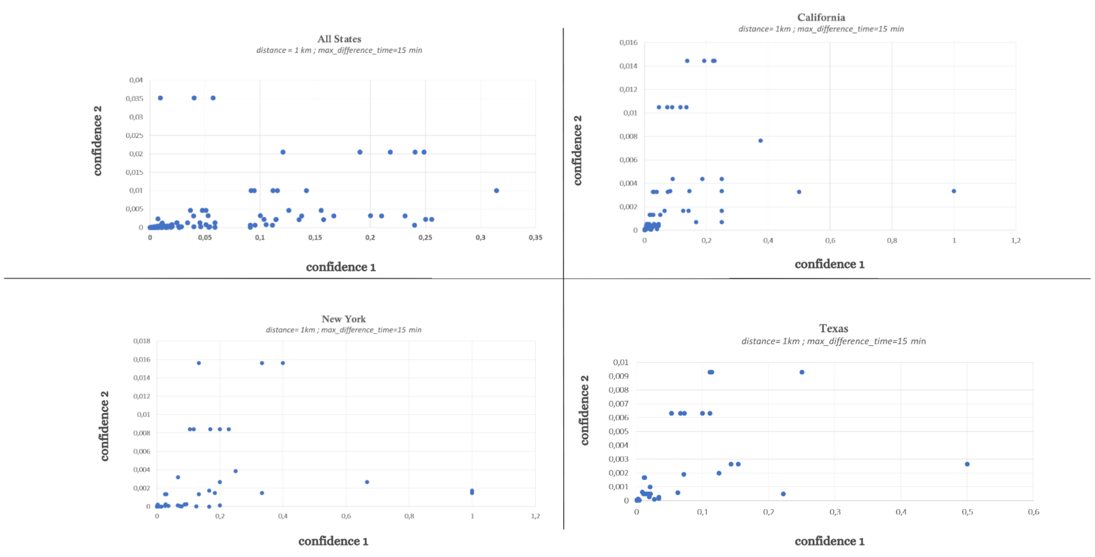

Figure 4.7. Confidence1 vs Confidence2 for all sequences extracted considering "All states" dataset; *distance*=1km and *max_difference_time*=15 min

There are not sequences with a confidence2 > 0,04. While we can observe values of confidence1 up to 1, although most sequences have a confidence1 value lower than 0,1. The results in figure 4.7 confirms that confidence2 metric cannot be used to discriminate which co-location sequences are important. This metric is useful to be compared with other confidences.
In figure4.8 are reported the values of the ratio between confidence1 and confidence2 for some of the sequences found considering *All states* dataset (*distance*=1km and *max_difference_time*=15 min). The sequence ($[Broken - Veichle, Lane - Blocked] \Rightarrow [Broken - Veichle, Accident]$) ($[BV, LB] \Rightarrow [BV, A]$) has a considerable value of this ratio, this means that the event of

type $Broken-Veichle$ increase the probability of finding an event of type $Lane-Blocked$ followed by an event of type $Accident$ in the same area. Not all the sequences with an high value of the ratio between confidence1 and confidence2 can be considered meaningful.

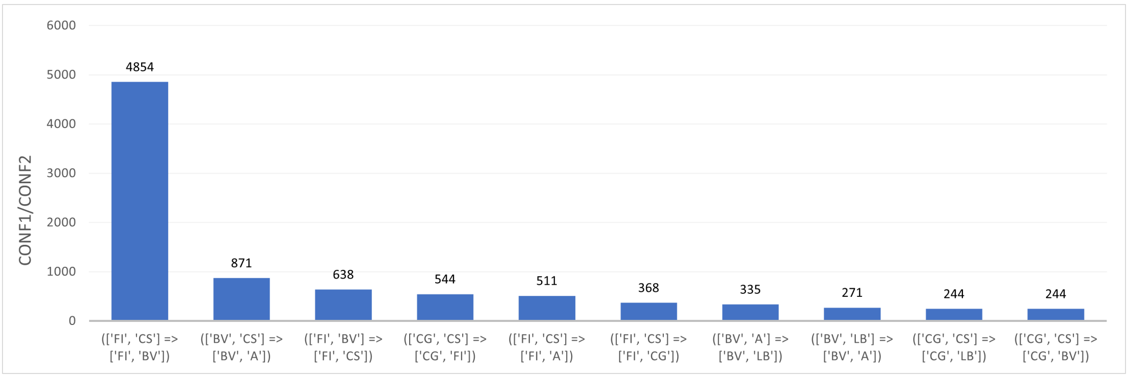

Figure 4.8. Confidence1/Confidence2 for some sequences extracted considering "All states" dataset; *distance*=1km and *max_difference_time*=15 min

Similar trends are observed in the scatter graph that compares confidence1 and confidence3 (4.9). Confidence3, similarly to confidence2, does not reach high values and always remain lower than 0.006.

To complete the analysis we check the execution time of the algorithm for co-location sequence mining. In figure 4.10 are reported the execution times obtained with different configuration of input parameters (*distance* ; *max_difference_time*). The co-location mining phase is negligible if compared with the time required by co-location sequence mining. As we can expect, the execution time is lower when the algorithm is run with small values of *max_difference_time* and *distance*. The computation depend on the dimension of the neighbor database, when the constraints on *distance* and *max_difference_time* are strict (smaller values) the number of neighbors for each spatio-temporal event is reduced and the execution of the algorithm is faster since there are less data to consider.
To summarize, in this section we verified the effectiveness of the algorithm for co-location sequence mining, showing that the application is able to find many sequences of correlated categories of traffic events. However it's hard to extract some general association rules, considering our definitions and metrics. We found many sequences with low values of support that may alter the value of other metrics. This issue is probably due to the intrinsic structure of the considered dataset. We expect more clear results if the algorithm would

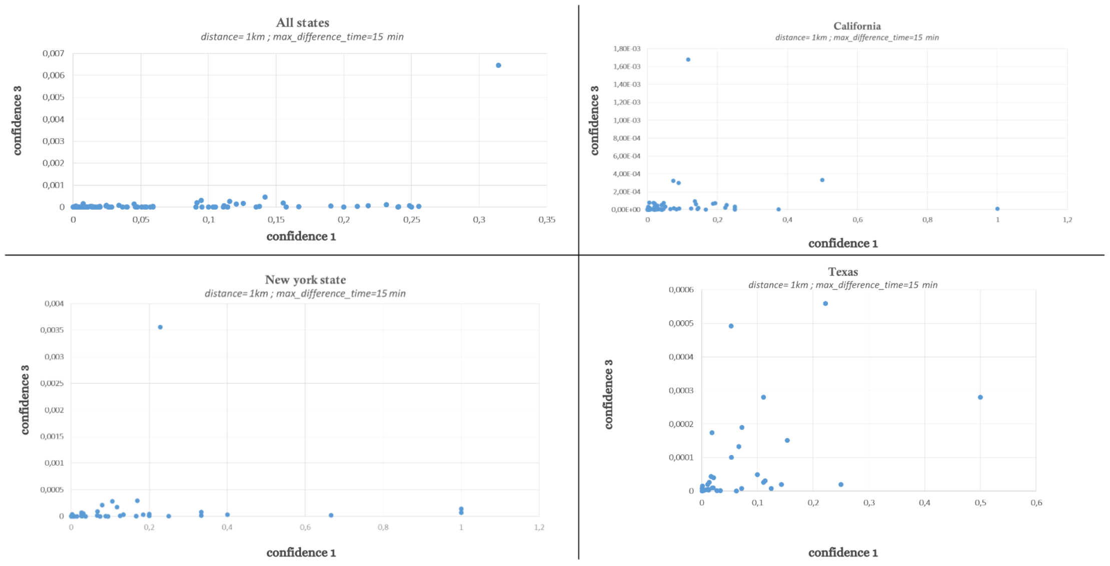

Figure 4.9. Confidence1 vs Confidence3 for all sequences extracted considering "All states" dataset; *distance*=1km and *max_difference_time*=15 min

be applied to a dataset that contains more specific traffic labels and so less instances for each event type.

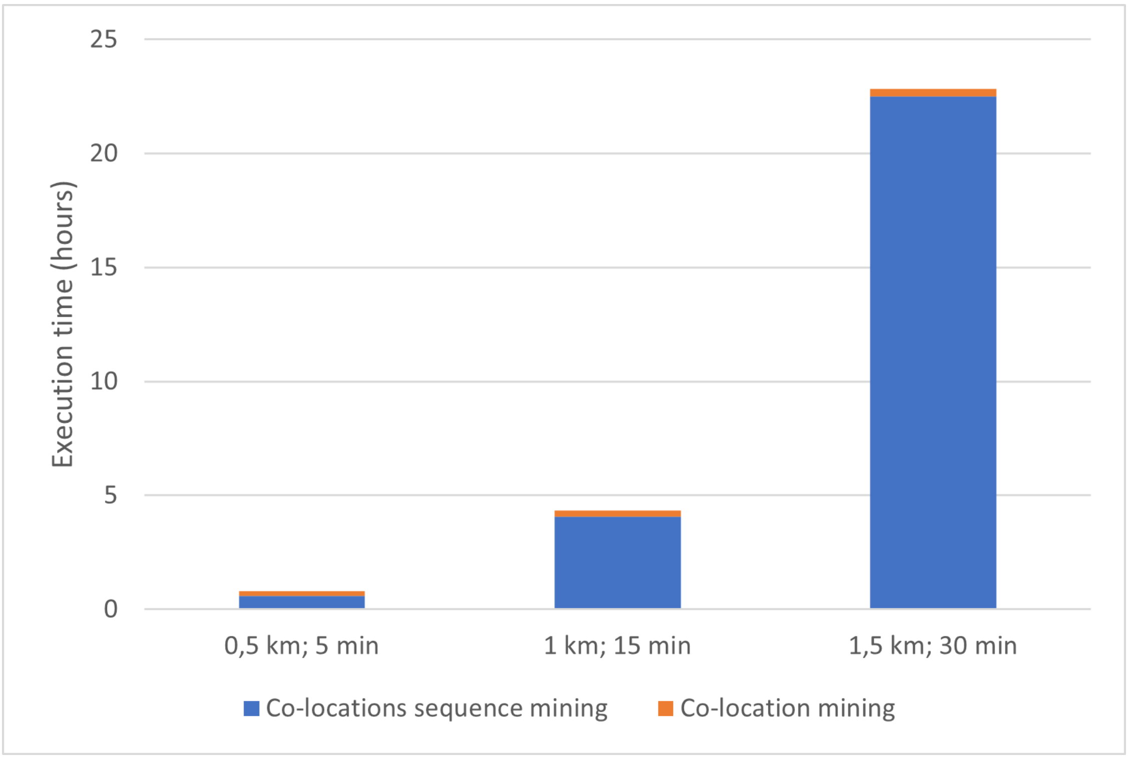

Figure 4.10. Execution time (in hours) of algorithm for co-location sequence mining with different values of *distance* and *max_difference_time*

Chapter 5

Conclusion

This thesis highlights the importance of co-location patterns. Co-location pattern mining is an innovative technique that can be used to discover relationships and correlations between categories of events in different contexts: urban field, traffic management, scientific researches. These patterns are discovered when instances of certain attributes or categories tend to co-occur or appear together within the same geographic area and at the same time. In this thesis we show that co-location pattern mining can be effectively applied to huge datasets (more than 30 M instances). Many state-of-art algorithms have been developed to perform co-location pattern mining task. Most of these technique have a big issue: they are not scalable and struggle when employed with huge datasets. The recent introduction of parallel co-location pattern mining based on column calculation and neighbor-dependency partition is extremely interesting for our purpose. However this algorithm manages data that only have spatial dimension. In this thesis we aim to deal with data that have both spatial and temporal information. We witness the lack of an algorithm able to mine spatio-temporal co-location patterns in the literature. Nevertheless, we show that the idea of parallel co-location pattern mining based on column calculation and neighbor-dependency partition can be employed also for an application that handles spatio-temporal events. There are many different approaches to develop the algorithm for spatio-temporal co-location mining. The critical phase of this algorithm is the computation of spatio-temporal neighbors among instances. As far as we know there is not a standard procedure to perform the calculation of spatio-temporal neighbors. Thus, we explore different solutions and we compare the results of them in terms of required execution time to find out the

most efficient. Overall, the algorithm we develop is able to successfully perform spatio-temporal co-location mining over a huge dataset that contains more than 30M of events located in space and time. Our best solution for spatio-temporal co-location mining outperforms the algorithm we considered as baseline, requiring 41 % less time.
We apply our analysis on 2 different types of datasets: Geolife (people trajectory data) and LSTW (traffic and weather events). The results we get in terms of co-location patterns are interesting. Nevertheless we try to give our contribution to the field of co-location pattern mining, introducing the new concept of sequence of co-location patterns.
A sequence of co-location patterns is a set of related co-locations that share an aggregator event and respect some spatio-temporal constraints. This new concept should help to find occult relationships among categories event that simple co-location mining is not able to discover. We provide a formal definition and suitable metrics to perform mining of co-location patterns sequences. Furthermore we show an implementation of an algorithm able to perform this task in a parallel manner exploiting Apache Spark environments. We test the new application on a dataset containing more than 30M events related to traffic and weather (LSTW). The algorithm is able to deal with huge amount of data, and the results are encouraging, but our analysis has also some limitations. The considered dataset contains few general category labels and many events for each category, this kind of structure likely alter some results. We can't find other datasets containing a sufficient number of spatio-temporal events. Furthermore, due to the limited computational resources available, we can't perform mining of co-location sequences with size > 2. Finally, the algorithm for co-location sequence mining works as "post-processing" with respect to the simple co-location mining phase. As a future work, the integration between co-location sequence mining and simple co-location mining may be investigated. This hypothetical application should be tested on other new datasets containing more detailed labels with less instances.

Bibliography

[1] Peizhong Yang, Lizhen Wang, Xiaoxuan Wang, Lihua Zhou and Hongmei Chen. *Parallel Co-location Pattern Mining based on Neighbor- Dependency Partition and Column Calculation.* School of Information Science and Engineering, Yunnan University. 2021.

[2] S. Shekhar and Y. Huang. *Co-location rules mining: a summary of results.* Proceedings of SSTD, pages 236–256. 2001.

[3] Y. Huang, S. Shekhar, and H. Xiong. *Discovering colocation patterns from spatial data sets: a general approach*

[4] Xin Hu, Guoyin Wang. *Mining Maximal Dynamic Spatial Co-Location Patterns.* IEEE. 2019

[5] Y. Ge, Z. Yao and H. Li. *Computing Co-Location Patterns in Spatial Data with Extended Objects: A Scalable Buffer-Based Approach.* IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 2, pp. 401-414. 2021.

[6] X. Yao, L. Chen, L. Peng, T. *A co-location pattern-mining algorithm with a density-weighted distance thresholding consideration.* Inf. Sci., 396: 144-161. 2017.

[7] Xuguang Bao, Lizhen Wang. *A clique-based approach for co-location pattern mining.* Information Sciences, Volume 490, Pages 244-264. 2019.

[8] Lizhen Wang, Yuzhen Bao, Joan Lub, Jim Yip. *A New Join-less Approach for Co-location Pattern Mining.* IEEE International Conference on Computer and Information Technology. 2008.

[9] Liturri. *Temporal co-location pattern discovery in spatiotemporal data through parallel computing.* 2023.

[10] Jin Soung Yoo, Shashi Shekhar, Mete Celik. *A Join-less Approach for Co-location Pattern Mining: A Summary of Results.* Data Mining, Fifth IEEE International Conference. 2005.

[11] Sobhan Moosavi, Mohammad Hossein Samavatian, Arnab Nandi, Srinivasan Parthasarathy, and Rajiv Ramnath. *Short and Long-term Pattern*

Discovery Over Large-Scale Geo-Spatiotemporal Data. The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19). 2019.

[12] *https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/*

[13] *https://smartdata.polito.it/computing-facilities/*