View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UPCommons. Portal del coneixement obert de la UPC

Computing and visualizing informative trajectories in temporally annotated data

Martí Zamora Casals

July 4th, 2017

Director: Ricard Gavaldà Mestre
Department of Computer Science

Master in Innovation and Research in Informatics
Specialization in Data Science

Facultat d'Informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC) – BarcelonaTech

Abstract

A trajectory in the medical world, is the sequence of events that occur during the life of a patient. In the recent years, these trajectories have been stored in the Electronical Health Records and many of the health organizations have databases with the clinical history of all their patients. The trajectories can be summarized in a *trajectory graph* which shows the different paths the trajectory of a patient may take. The graph contains events on its nodes and the edges contain the temporal relations. Previous works focused in the exploration of *trajectory graphs* only allow one event at each node, thus losing information and potentially mixing different groups of patients. In this work, we have developed a new procedure to extract the *trajectory graphs* that allows having several events in a single node of the graph. This procedure has been tested in two real world datasets: one related to diagnostics at hospital admissions, and the other on prescriptions in intensive care units.

Acknowledgements

I want to thank Ricard Gavaldà for proposing me such an exciting project and helping me with ideas and advice. Also, my gratitude to Julianna Ribera for providing medical expertise and help when reviewing the results and all the team Esther Limón, Ester Amado and Sílvia Cordomí for helping in the definition of the project. Finally, my family for their unconditional support.

Contents

Chapter 1

Introduction

Chronic diseases and polymedication are common in developed countries. As a person grows older it is normal to be diagnosed and prescribed with an increasing number of diseases and drugs. With them, it also increases the risks associated with the side effects of combining several drugs due to unknown interactions.

In the last years, health-care centers and institutions have been collecting and storing Electronic Health Records (EHRs). These records, can be an ordered sequence of events that occurred to the patient (diagnostics, medical prescriptions, procedures or laboratory tests) - what is known in the medical world as the *trajectory* of the patient. With a computer science perspective combining trajectories from several patients it can be obtained the *trajectory graph*, a graph where the nodes are states or stages of a disease and the edges are the transitions from one state to another. These states can be seen as distributions indicating the probability of an event.

Some studies have been done to explore the trajectories from a temporal point of view [14][17]. However, they only allow a single event in the nodes of the graph therefore, losing all the information on the events that can co-occur at the same time. Moreover, they only allow each event to appear once in the graph potentially mixing different types of trajectories into one. Other studies, [6] [21] use Hidden State Models to internally model the *trajectory graph* but they are difficult to represent and interpret and are not used for data exploration but for prediction tasks or patient clustering.

In this work, a new procedure has been developed in order to be able to visualize temporal trajectories of events considering the events that can co-occur when the patient is in a state. The procedure has been developed having in mind the structure of EHRs datasets but it could be used in different domains as well. This procedure can absorb different types of temporal data such as diagnostics, prescriptions or medical procedures. It also can scale to large data sets with thousands of events and has been tested in real world data.

This work is structured as follows. Chapter 2 explains the state of art exposing previous works in the same line as well as typical approaches that could be used to solve the problem. Chapter 3 justifies and explains the proposed solution. Chapter 4 explains how the solution has been tested in two real world datasets. Finally, Chapter 5 exposes the conclusions and proposes some future work lines.

Chapter 2

Related work

Many works in the past have analyzed event sequences from a temporal perspective. They can be classified in many ways:

- Some solutions generate a graph as output, they are normally synthesizes the information more whereas some others simply extract important sets of sequences or patterns.
- Sometimes the states are labelled with observable quantities (e.g. diagnostics) and sometimes they are hidden (latent in the data, but not directly observable).
- The transitions can be events meaning that when some event happens the state changes or the events can be in the states which could be formed by one or more events.
- Some works have a notion of time, something happening after one second can be different of something happening after an hour even if it is the same event, some others only consider the order of the events.
- They can be also classified depending on the goal: (exploration, clustering or prediction) and it may be needed a similarity function for clustering or a loss function for prediction.

One of the articles with more impact both in the medical and computer science worlds [14] uses the data from the whole population from Denmark (6.2 million people) during 14.9 years to find temporal graphs between diseases. They group the trajectories and center them in some key diagnoses such as diabetes, chronic obstructive pulmonary disease or cardiovascular disease see Figure 2.1. The result are acyclic graphs with a single disease on each node with an edge between two nodes if there is a temporal relation between each other. Having a large dataset allows them to use very strict significance tests. However, each disease can only appear once in a graph losing information about groups of patients sharing a disease; for example, two groups of patients one following a trajectory $a \to b \to c$ and the other one being $a' \to b \to c'$ end mixed in the graph and it is not possible to know if there are two groups or some patients follow the trajectory $a \to b \to c'$. They also ignore non-temporal relationships losing the information about secondary diseases occurring at the same time with the primary disease and again potentially mixing different groups of patients.

In [17] they build a temporal disease graph with the notion of time between events, the relations are penalized if more time happens between two events. Their system can obtain data from different types (they test it with diagnoses and drugs) and predict future conditions and also generate graph representations of the learned trajectories. They test their method with 1,000 synthetic generated samples and two sets of patients with 430 and 1,127 patients.

Many works use hidden state models to predict future states for a patient. [6] uses Generalized Linear Dynamic Models to predict the mortality inside Intensive Care Units combining heterogeneos data from the MIMIC-II dataset using 15,000 EHRs. In [21] the program learns a hidden

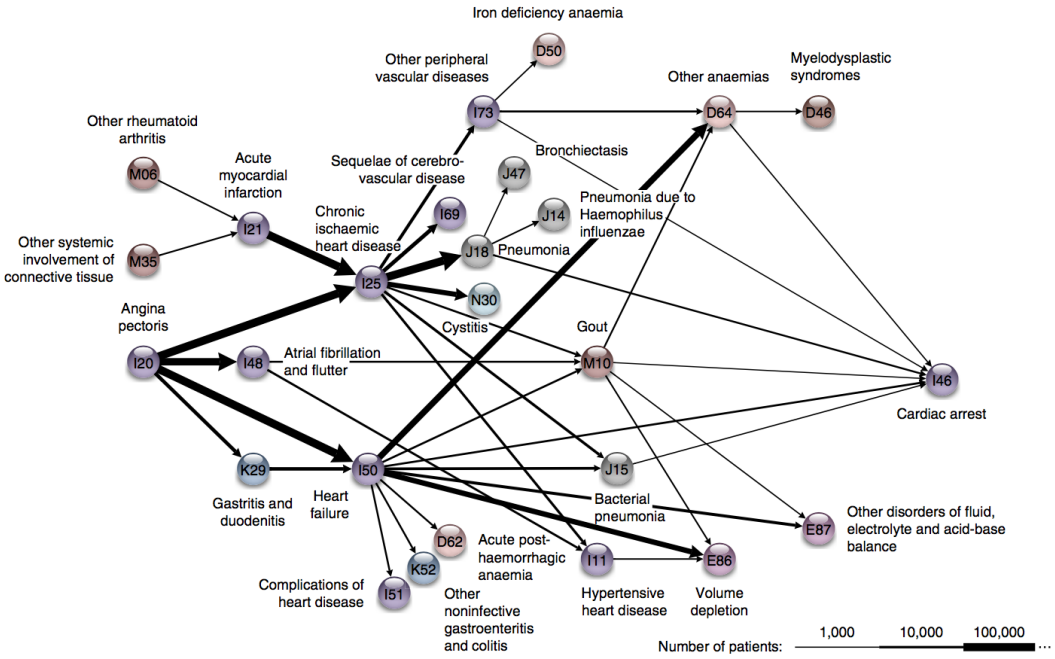


Figure 2.1: Cardiovascular disease *trajectory graph*. Source: [14]

state model that can predict new stages of a disease. They test it with a diagnostics dataset of 300,000 patients during 4 years.

Frequent sequential pattern mining consists in finding frequent occurring subsequences and is one of the first approaches one can use when the input data consists in sequences. They can be obtained using algorithms such as PrefixSpan [19][12] that creates *database projections* over sequence prefixes or SPADE [23] that uses efficient lattice search and joins with only three passes over the database. The output of these algorithms is a list of frequent subsequences, their support and optionally the set of sequences containing this subsequence. To transform the frequent subsequences to a *trajectory graph* it would be needed to merge the subsequences based on some similarity measure. In [18] they use *frequent sequential pattern mining* to generate features to predict the survival period for patients with a type of cancer (Glioblastoma Multiforme) and discover which combinations of genomic factors combined with treatment patterns influence longevity. In [5] the temporal relations between two events (e_1 before e_2, e_1 after e_2, e_1 overlaps e_2, e_1 contains e_2, ...) are taken into account for the sequential pattern mining algorithm which they use to predict the risk of patients treated with heparin of developing heparin induced thrombocytopenia.

Finding patterns in *temporally annotated sequences*, that is, a sequence of pairs *(event-id, time)* has not been approached as frequently as the *frequent sequential pattern mining*. In [22] they find, in addition to the frequent sequential patterns, the time intervals between events which they call *delta patterns*. Some other approaches [11] also include doing a kernel density estimation to find the distribution of the time between events which can lead, for example, to two patterns with the same events $a \to b \to c$ but with the transitions $a \to b$ and $b \to c$ having different associated distributions.

The problem using sequential pattern mining with or without temporal annotations is that from large number of output sequences many of them will be redundant and when using medical data they might be also incomplete if the data does not span for many years which is not often the case.

Another solution might be learning *finite state machines* (FSM) using algorithms such as RPNI [9] that learns a probabilistic deterministic finite atuomata or ALERGIA [7] which is an extension of the former. Both build a *Prefix Tree Acceptor* (PTA) from the training sample and merge the the states when they are equivalent within a statistical uncertainty. One problem with these

approaches is that generating the initial PTA is too expensive when the number of sequences and events grows. Another problem is that events happen only one at a time. If two events A and B are related but there isn't a temporal relation between them, two different states will potentially be gererated (the state "A but not yet B" and the state "B but not yet A").

In [20], J. Schmidt proposes several FSM learning algorithms capable to treat multivariate data are presented. The author proposes four algorithms is PRTA which is the basic version, SPRTA, CPRTA and OPRTA that are designed to be able to scale to large datasets. The methods use clustering to merge the states. Specially worth to mention is OPRTA that is the online version of the PRTA and manages to avoid creating the initial and memory consuming PTA and the first clustering step. It does so adding each sequence incrementally to an existing PRTA. OPRTA also supports concept drift hence, the strategy consists in generating first a PRTA with a small subset of data that fits in memory and then start consuming the stream of new data. Finally, the algorithms are tested with both synthetic data and real-world data consisting in some medical datasets very similar in structure to the ones used in Chapter 4 mainly consisting in diagnoses. The other datasets are composed by laboratory test results which requires that the algorithms should be able to treat with several variables at the same time and genetic data. The final FSA can be explored in a state graph or *trajectory graph* where the transitions between states correspond to events.

Chapter 3

Methodology

The approaches seen in Chapter 2 that generate *trajectory graphs* from the data have the problem that do not keep the information about the events happening at the same time each node can only contain one element. Moreover, each new event implies a state change. Finally, each event can only appear once in the graph, this implies that some information is lost if different groups of patients converge in the same node even if they go to totally different nodes later. For example, the chronic kidney disease may be caused, among others, by diabetes [2], hypertension [16] or glomerulonephritis [3]; merging these three kinds of trajectories into the chronic kidney disease means losing the information in the different trajectories these groups of patients might have.

When talking to medical experts the model we think they have in mind, without using the same vocabulary, is a model where the trajectory of a patient is formed by states, each state is formed by a central event, normally a diagnostic and a set of secondary events that co-occur at the same time with the central event.

More formally, each patient will evolve during her life through a trajectory formed by a sequence of unobservable states $s_1, s_2, ...s_n$. Each state i has an associated distribution $D_i(x)$ for an event x (which may be a diagnostic, prescription, procedure, laboratory analysis, etc.). During the period the patient is in the state s_i, she generates the event x following that distribution $D_i(x)$ randomly and independently one of the other. If the probability of generating x_j changes conditioned that x_i has been generated before it means the patient is in a different state. The distribution of time for a patient being in a state and the probabilities of changing to a different state will also be characteristic of the state.

The *trajectory graph* can be cyclic. For example, a patient that breaks a leg will transition from the state "healthy" to the state "broken leg"; the "broken leg" state will have associated "pain killers", "plaster the leg", etc. After this, the patient will transition again to the "healty" state. For chronic diseases though, the resulting graph should be acyclic as a patient with a chronic disease will not transition back to the "sane" state and the disease will evolve to complications.

Using EHRs data involves taking into account an added set of requirements which the developed solution should take into account:

- **Limited time span of the dataset.** Usually the provided dataset will not be for long periods of time. However, the trajectories of chronic diseases may evolve during long periods of time, longer than the time span of the dataset. Then, no patient will follow the whole trajectory but the procedure should be able to merge similar small trajectories to be able to find all the trajectory.
- **Data heterogeneity.** The data available might be from different sources (diagnostics, prescriptions, procedures, ...) the proposed solution must be flexible enough to absorb different types of data. Even more, some demographic data (age, gender) may or may not be available and it should be able to use it when it is available but also work when it is not.

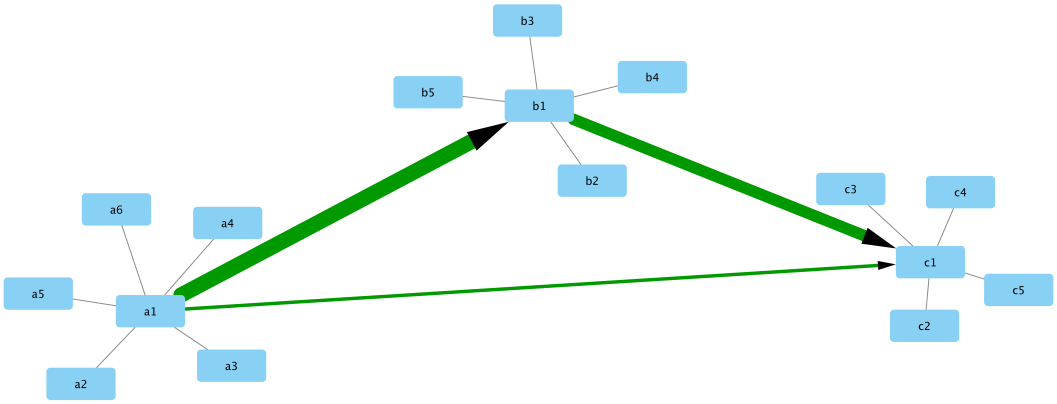


Figure 3.1: Example of a *trajectory graph* of the kind we would like to build. Green arrows indicate temporal relationships and their thickness the importance of the relation. Black edges connect each node with the central node of its state.

- **Irregular data.** The data are not taken at regular intervals, if the data is coming from hospital admissions will be very different compared to the data coming from regular patient visits. Even in the same dataset the differences might be important.
- **Misannotations.** If the data is taken by hand and later digitalized there can be different ways to annotate the same event. Even some doctors might annotate differently the same diagnostic. The annotated data in the databases is not frequently used for the day-to-day use and it is only used for analysis meaning that the data quality can be questionable and may require intense preprocessing.

This section explains the procedure to generate a trajectory graph from medical events (e.g. diagnostics, prescriptions or procedures) that have a date and a patient id. Many clinical databases are compatible with the patient-id, event-name, event-time structure hence they are compatible. However, as the following procedure is not tied to only medical data, the used nomenclature will be: sequence when we are referring to an individual of the database (a patient) and they will be identified with an id and event for the different events happening in the sequence and it will have an event-id and a time. Optionally, the sequences may have non-temporal data, usually demographic data such as age or gender.

The *trajectory graph* built after the procedure will consist of states in the nodes and temporal relations as edges indicating the transitions between elements in the graph. Each state will be composed by a set of events with non-temporal relation and one of them will be the central event. A path in the *trajectory graph* can be seen as a possible trajectory for a patient.

The states and the temporal relations will have associated a set of sequences. It is important to notice that for a sequence belonging to a state, the sequence, does not need to have all the events and it is not necessary that the sequence follows the whole trajectory of states either. Figure 3.1 shows an example of a *trajectory graph*.

The procedure can be divided in five phases:

1. The **preprocessing** phase sorts the sequences and removes those events with low support.
2. The **counting** phase counts the pairs of events appearing on each sequence.
3. The third phase computes the **importance and temporality** of each pair of events.
4. In the fourth pahse the **states are generated**.
5. In **graph cleansing** pahse the graph is simplified.

The procedure has a worst case scenario time cost of $O(|E|^2 + \sum_{s \in S} |s|^2)$ where $|E|$ is the number of different events in all the dataset, $|s|$ is the number of different events in the sequence s and S the set of all input sequences. This is a worst case scenario however, in the experiments done in real datasets the distribution of events in sequences tends to be stable in future works this cost could be transformed to $O(|E|^2 + |S|)$ if we ensure that $|s|$ does not grow too much.

Regarding the memory, the cost is simply $O(|E|^2)$ and does not depend on $|S|$. In the clinical databases scenario, this is good because the number of patients can grow but the number of different events tends to stabilize at certain number of patients (the probability of having a disease does not depend on the number of patients). The experiments in the MIMIC-III dataset show that when seeing the first 5,000 patients the number of seen event pairs increases up to 300,000 whereas seeing the rest of 35,000 patients only increases the event pairs count in 120,000.

3.1 Input data

The input data set will be a set of sequences each sequence will have a set of events and optionally some non-temporal variables such as, age and gender. In this first work, there won't be associated time distributions in the edges thus, the specific time of the events does not have to be stored if the sequences are sorted and one keeps track of the events happening at the same time.

Optionally, those events with low-support (i.e. not happening enough times in the DB) can be filtered-out to speed-up further procedure.

3.2 Counting events

The first step consists in counting events. In later steps, it will be needed to use this data as sets to obtain the cardinality of unions and intersections. This means that in addition to counting the sets should also be maintained. In Section 3.6 some optimizations will be introduced in order to avoid the cost of keeping the sets in memory.

Three different counts are needed:

1. Count when event a happens before event b.
2. Count when events a, b happen at the same time.
3. Count the total times each event happens.

Computing the three different counts is simple, for each event in the sequence update the third counter and for each pair of events a, b in a sequence update the first counter if a happens before b or update the second one otherwise. This takes only one run over the whole database of sequences.

Optionally, if there are demographic variables, a matrix of counters with the different profiles (combinations of variables) can be used.

3.3 Computing the importance and temporality of event pairs

The next step consists in computing how the event pairs are related. Two values are computed for each pair of events, one for assessing the importance of the relation and the other to know how clear the temporal relation is.

To measure the importance of the relation we have used the lift.

$$\mathit{lift}(a,b) = \frac{p(a,b)}{p(a)\cdot p(b)}$$

The lift is typically used in Association Rule Mining and it can be used to measure the relation of events. A lift greater than 1 indicates a strong relation between the two events whereas a lift near to 1 indicates that the two events do not tend to appear together more than what is expected for random chance. In that case, the events are sets and the operation would be:

$$\mathit{lift}(a,b) = \frac{|a\cap b|\cdot|S|}{|a|\cdot|b|},$$

where $|S|$ is the number of sequences. Using the lift can be misleading if two events obtain a high lift when they are actually conditionally independent to some other variable. In the clinical field this can happen with diseases tied to a certain age or gender.

$$p(a,b|z) \approx p(a|z)\cdot p(b|z)$$

One way to avoid that is to compute a different lift for each profile (one lift for each combination of the demographic variables). Then, they can be averaged. Define

$$\mathit{lift}(a,b;Z) = \frac{|a\cap b\cap Z|\cdot|S\cap Z|}{|a\cap Z|\cdot|b\cap Z|}$$

where Z is the set of sequences of a profile, and then

$$\mathit{lift}(a,b) = \sum_{Z\in P}\frac{\mathit{lift}(a,b;Z)\cdot|Z|}{|S|}$$

where P is the set of different profiles.

To measure the temporal relations the p-value of a happening before b with probability 50% is computed. A binomial test has been used for that purpose: The null hypothesis is that the two events happen at the same time then the subsequence a,b should be as frequent as the subsequence b,a. The alternative hypothesis is that a happens before b then the subsequence a,b will be more frequent than the subsequence b,a. In the binomial test the cardinality of the sequences with the subsequence a,b will account for successes whereas the cardinality of the union of the sets of sequences with the subsequence b,a and the sequenences with the subsequence a,b happening at the same time account for failures.

The null hypothesis can be rejected if the p-value from the test is lower than a threshold, typically 0.01. Bonferroni correction [8] can be added to that threshold to reduce the probability of an incorrect rejection due to the large number of hypothesis tests being made. However, this is only possible if the data is large enough to do so.

3.4 States and trajectories computation

This step consists in grouping pairs of events and building temporal relations between them. The main idea of the algorithm below is trying to join those states that share events and share sequences but do not have a temporal relationship between them. The algorithm takes as input the list of pairs of events a,b sorted by lift up to a *min_lift*. Each pair of events $(evt.a, evt.b)$ also has

associated three sets, the set of sequences with the subsequence $a, b(evt.ab)$, the set of sequences with the subsequence $b, a(evt.ba)$ and the set with the sequences where a, b happens at the same time *evt.same* It outputs a list of states and a list of temporal relations. Each state contains a set of events that can happen in it and a set with the sequences belonging to that set.

The algorithm has two helper functions: **check_merge(s1, s2)** checks if two states are similar enough to be merged, the conditions are that they share at least one event and that they share a ratio of sequences big enough specified by *min_similarity* which can also be seen as the Jaccard Similarity [13] $|set1 \cap set2|/|set1 \cup set2|$ and **merge(s1, s2)** that actually merges the two sets. It also tries to merge the resulting set with the other sets calling recursively the *merge* function.

Algorithm 1 Generate states

```
states ← list()
temp_relations ← list()
event_pairs ← filter(event_pairs, evt : evt.lift > min_lift)
event_pairs ← sort(event_pairs, evt : evt.lift, DESC)
for all evt ∈ event_pairs do
    if evt.pvalue < min_pval then
        temp_relations.add(evt)
    else
        state ← (set(evt.a, evt.b), evt.ab ∪ evt.ba ∪ evt.same)
        states.add(state)
        for all s ∈ states do
            if check_merge(s, state) then
                merge(s, state, states)
                states.delete_if_exists(state)
            end if
        end for
    end if
end for
```

Algorithm 2 Check merge

```
function check_merge(s1, s2)
    if s1 == s2 then return False
    end if
    (evts1, set1) ← s1
    (evts2, set2) ← s2
    if |evts1 ∩ evts2| == 0 then return False
    end if
    if |set1∩set2| / |set1∪set2| < min_similarity then return False
    end if
return True
end function
```

The central event of a state S will be the most common event e in the state:

$$\underset{e \in S}{\operatorname{argmax}} |S \cap e|$$

Other options to choose the central event could be using "expert advice" to decide which are the important events for example serious illnesses and which are not or modulate the frequency of the event in the state by how discriminative is among the states like the factor idf does in the tf-idf model.

The result of this step obtains a graph of trajectories, however the states tend to have many connections between them and it is difficult to understand. For this reason, the next step tries to make the view simpler.

Algorithm 3 Merge states

```
function merge(s1, s2, states)
    (evts1, set1) ← s1
    (evts2, set2) ← s2
    s1 ← (evts1 ∪ evts2, set1 ∪ set2)
    for all s ∈ states do
        if check_merge(s, s1) then
            merge(s, s1, states)
            states.delete_if_exists(s1)
        end if
    end for
end function
```

3.5 Graph cleansing

This final step intends to simplify the states merging the states that share the central event and removing some of the temporal relations. The merging is done like in Algorithm 3. This operation vastly reduces the number of states but the information is kept and it could be shown in a specialized user interface.

The second simplification consists in hiding the edges that do not go from a central event to another central event. This avoids to many edges appearing and disturbing the analysis.

Usually, at this step the resulting graph is acyclic, if it is not it should be transformed to an acyclic graph removing the less important edges causing a cycle. Once the graph is acyclic the longest path distance $d_{a,b}$ between each pair of states a, b can be computed. The importance of the edge a, b will be proportional to the number of sequences going directly from the state a to state b. To do this, the sequences following an indirect path to go from a to b should be subtracted from the computation of the importance a, b. That is, removing the elements arriving to b via another edge x, b where $d_{x,a} < d_{a,b}$.

$$importance(a,b) = \left| set(a,b) \setminus \bigcup_{x;(x,b)\&d_{x,b}<d_{a,b}} set(x,b) \right|$$

3.6 Optimizations

In this section some optimizations are described to make the computations fast and the memory footprint low enough to be used in real datasets.

3.6.1 HyperLogLog

HyperLogLog (HLL) [10] is a probabilistic data structure for counting the number of distinct elements in a multiset. Calculating the exact cardinality of a set requires linear memory. HLL estimates the cardinality with a reasonable estimation error bound while keeping the memory usage very low. The error ϵ of the counting depends on the size m of the data structure: $1.04/\sqrt{m}$ a larger size implies less error. The memory size of the HLL is $O(\epsilon^{-2} \log \log n + \log n)$ being n the size of the set. For example, for an error of less than 1% m should be 2^{14}.

The main operations of the HLL are **add** and **count** it also has a **create** operation and a **merge** operation to compute the union of two sets which adds a lot of power to the data structure such as easy parallelization or adaptation for *sliding window* usage. Additionally, some new operations can be derived combining the former operations and using the *inlusion-exclusion principle* like

difference_count to estimate the cardinality of the difference of two sets and **intersection_count** which can estimate the cardinality of the intersection.

These set operations fit in the operations necessary to compute the *trajectory graph* described previously allowing to reduce the necessary memory to $O(|E|^2)$ (being $|E|$ the number of different events in the dataset) and not dependent on the number of sequences.

Implementing the explained consists in using the HLL operations instead of the set operations. For example,

For computing the lift of two HLLs hll_A, hll_B we would use the *inlusion-exclusion principle* to compute the cardinality of the intersection of A and B.

$$|A \cap B| = |A| + |B| - |A \cup B|$$

then,

$$|A \cap B| \approx count(hll_A) + count(hll_B) - count(merge(hll_A, hll_B))$$

$$lift(hll_A, hll_B) \approx \frac{(count(hll_A) + count(hll_B) - count(merge(hll_A, hll_B)) \cdot N)}{count(hll_A) \cdot count(hll_B)}$$

being N the total number of sequences.

To compute the Jaccard Similarity between two states A, B with associated HLLs hll_A, hll_B:

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$$Jaccard(hll_A, hll_B) \approx \frac{count(hll_A) + count(hll_B) - count(merge(hll_A, hll_B))}{count(merge(hll_A, hll_B))}$$

To compute the importance of an edge (a,b) (which has an associated $hll_{a,b}$) we can take the same approach:

$$|A \setminus B| = |A| - |A \cap B| = |A \cup B| - |B|$$

$$hll_t = merge(hll_{x_1,b}, hll_{x_2,b}, ..., hll_{x_m,b}) \quad for\,all\,edges(x_i, b)\,with\,d_{x_i,b} < d_{a,b}$$

$$importance(a, b) \approx count(merge(hll_{a,b}, hll_{a,t})) - count(hll_t)$$

It is important to notice that the error of the intersection count is not relative to the intersection but to the union of the sets. Meaning that a doing a 1% error in the HLL does not mean having a 1% error in the intersection, it could be higher. It is important to create the HLLs with a size big enough to overcome this.

3.6.2 Parallelization

The computations described in Sections 3.2 and 3.3 are the more costly operations in the procedure. They can be easily parallelized splitting the counting in different machines or storing the event pairs in disk and doing several passes through the sequences database updating a subset of the event pairs each time.

Chapter 4

Experiments

The capabilities of the procedure explained in the previous section have been tested in two different datasets. The first dataset, is from the MIMIC-III database [15]. MIMIC is an open dataset developed by the MIT Lab for Computational Physiology, it contains deidentified health data from 40,000 patients. Among other, it includes demographics, vital signs, medications and laboratory tests. The second dataset is from Hospital de Sant Pau in Barcelona and it contains the diagnoses from hospital admissions between years 2008 to 2014 from patients with heart diseases.

4.1 Implementation

The procedure has been implemented in Python 3.4 using the Python library HLL[1] for the HyperLogLogs. No parallelization has been used. The tests have been done in a machine using an Intel Core i5 at 2.7 GHz, 16GB of memory and an SSD disk. The resulting graphs are visualized with the desktop version of Cytoscape[2].

4.2 MIMIC-III - Prescriptions

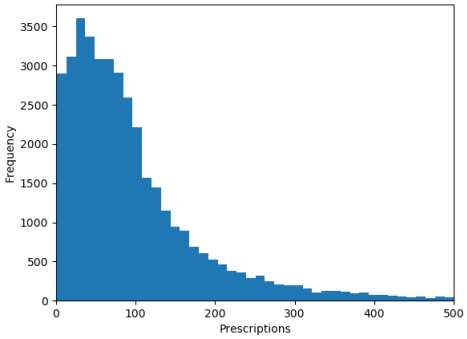
The MIMIC dataset contains several tables with different tables of data. For the experiment, it has been used the table with prescriptions. This table contains a subject id, an identifier for the drug and the date of administration of the drug making it compatible with the procedure. The table also contains the dose administrated but it has not been used.

Reading the input data takes 138 seconds, computing the lifts and p-values 152 seconds, computing the states 11.3 seconds and cleaning the graph only 1.3 seconds. In total 302 seconds and the memory footprint is 11.9GB. It can be seen that the most computer intensive parts are when the program reading the input data and counting the pairs of events and when the lifts and the p-values are computed. These two steps are the two steps that can be easily parallelized.

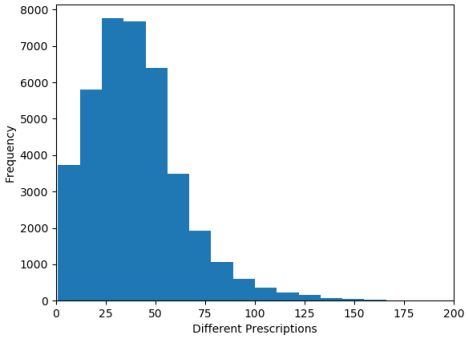
The table contains 4.15 million prescriptions and 40,000 different patients. On average each patient has 105 prescriptions but the range goes from 1 to 2,379 prescriptions (figure 4.1a). If only the different prescriptions are taken into account the average is 39 going from 1 to 221 (figure 4.1b). The patients stay 12 days in the hospital on average (accounting re-admissions) (figure 4.1c). There are 4,500 different events (prescriptions) however the distribution is very skewed. Some events are much more frequent than others only 761 have a support larger than 100 (figure 4.1d).

Only the events with a support larger than 100 have been considered. The number of pairs

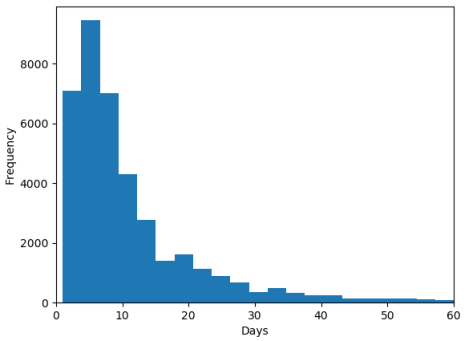
[1] https://github.com/ascv/HyperLogLog
[2] http://www.cytoscape.org/



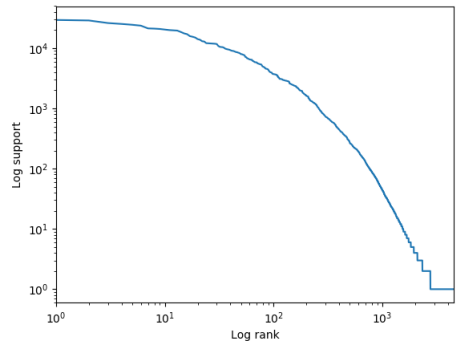
(a) Number of patients with a certain number of prescriptions.



(b) Number of patients with a certain number of different prescriptions.



(c) Number of days in hospital.

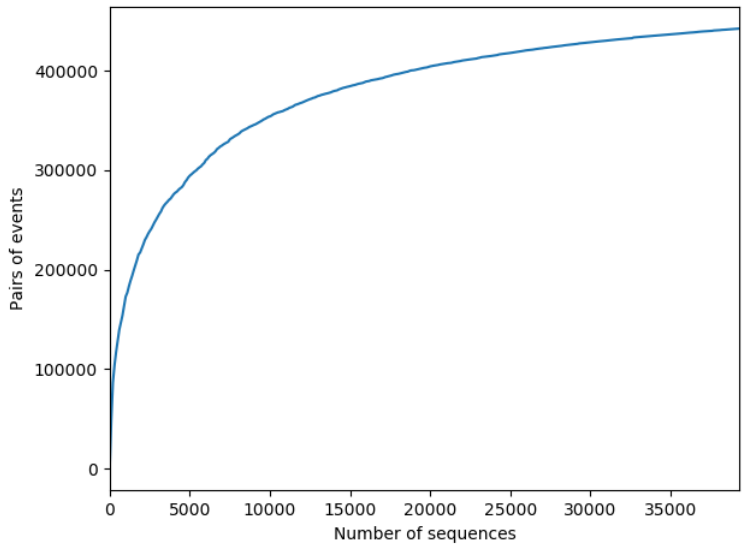


(d) Log-log plot of the event support sorted by support.

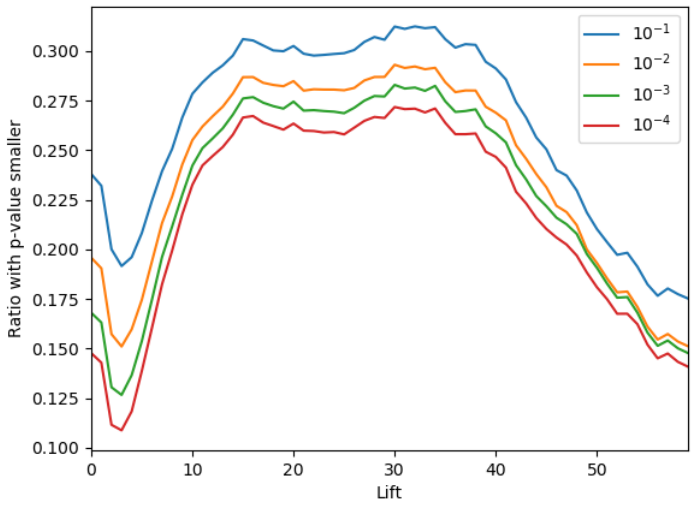
Figure 4.1: Distribution of the MIMIC-III prescriptions

of events is bounded by the square of the number of events. The number of events considered is 761 then the number of pairs appearing in the sequences should be about 580,000 as the number of patients grows. Figure 4.2a shows that this growth tends to slow down very quickly almost stabilizing around the 430,000 pairs of events having only seen 35,000 different sequences. The support for the pairs of events is also very skewed: only 110 pairs of events have a support larger than 100. The pairs of events with a support lower than 100 have not been considered. Some of the lifts are large being 600 the largest. 2,500 pairs have a lift larger than 10 and in figure 4.2b it can be seen the relation between the ratio of p-values and the lift. The ratio of significant p-values is lower when the lift is low and when the lift is very high (larger than 40). The interval between with lift between 15 and 40 contains a ratio of significant p-values larger.

The number of states depends of the minimum similarity (percentage of shared patients) allowed to merge to states. Having a minimum similarity of 0% (which is equal to merge two states if they share an event) 341 states are obtained. On the other hand, only allowing a minimum similarity of 100% for merging 976 states are obtained.



(a) Pairs of events against sequences. The growth of newly seen pairs of events tends to slow down as the number of sequences grows.



(b) Ratio of significant p-values against the lift.

Figure 4.2: Counts, lifts and pvalues for the pairs of events

Figure 4.3 shows the *trajectory graph* with medication given to premature newborn babies. Green arrows indicate a temporal relation the thickness of the arrow indicates how many patients are following that trajectory and black edges indicate a non-temporal relation in a superstate. Remember that not all the drugs in a superstate are given to the patients belonging to that superstate. Some of the secondary drugs are the same from one state to the other, this is the expected behaviour explaining that some elements from the combination of drugs have changed but not all.

The medication given to premature newborns is the one with larger lifts. Many of the combinations include Heparin, which is an anticoagulant, caffeine that is given to avoid apnea in premature newborns and several combinations of vitamins. Some of the temporal relation are between the same drug but with different application: from intravenous to syrup for example. The trajectories have been extracted with a minimum lift of 10, a minimum state similarity of 0.8 and a maximum p-value of 0.001.

Figure 4.4 contains the drug trajectories for transplanted patients. This trajectory graph contains mainly imunosupressors like Mycophenolate mofetil or Tacrolimus or Prednisolone which are used to prevent organ rejection after a kidney, liver or heart transplants. The trajectories have been extracted with a minimum lift of 6 and a minimum state similarity of 0.8 and a maximum p-value of 0.001.

Many of the relations observed are between the same drugs administrated in different ways, this could be prevented manually checking and correcting the names of the medication or if the database used a standard codification like ATC codes[3] which encodes the active principles and can be aggregated in a hierarchical way.

[3]The Anatomical Therapeutic Chemical (ATC) Classification System are the classification codes for active ingredients of drugs [1]

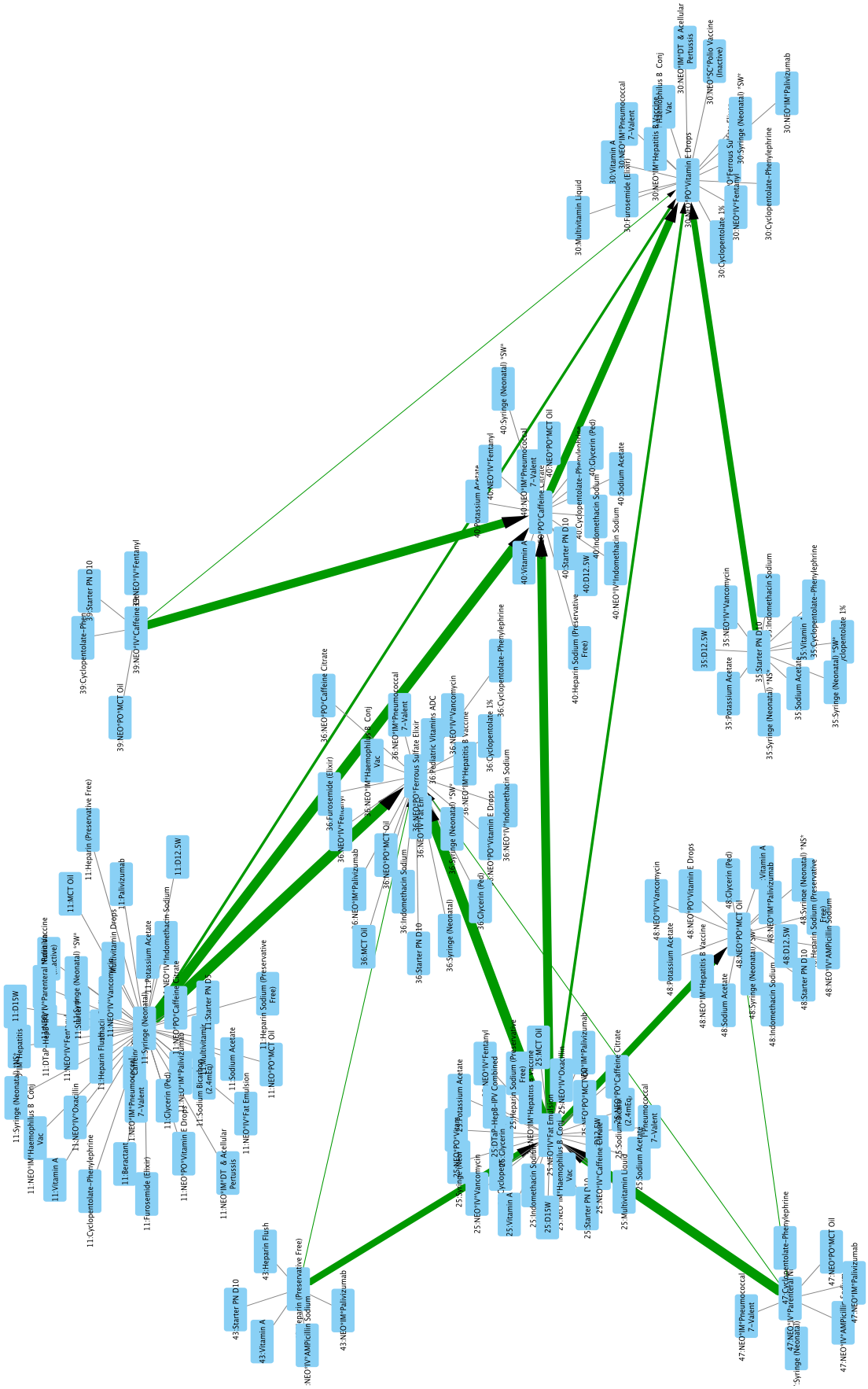


Figure 4.3: Medication for newborn patients

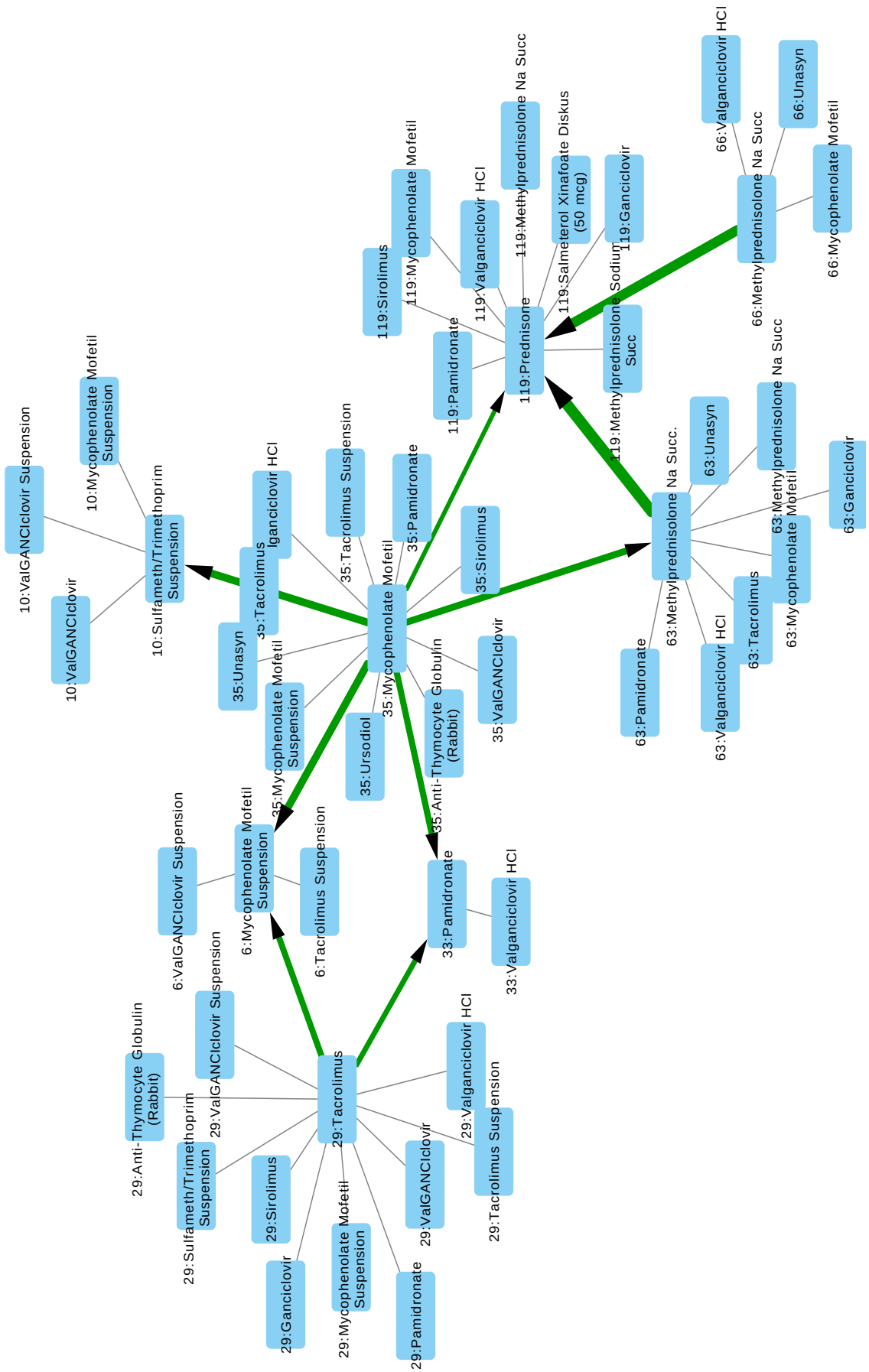


Figure 4.4: Medication for trasplanted patients

4.3 Hospital de Sant Pau - Heart diseases

The second dataset from Hospital de Sant Pau contains hospital admissions data from 2008 to 2014 from patients with heart problems. In the data are encoded for each admission the patient id, the date and the list of diagnoses in ICD-9 [4]. As there is only one date for each admission only the patients with two admissions or more can be used to study trajectories. 4,000 patients have more than one diagnostic which add up to 110,000 events having 3.2 different events each patient on average and 544 different events in total.

Having less sequences compared with the other dataset and with the average number of events for sequence also lower makes that the number of pairs of events hasn't been stabilized yet (figure 4.5a). The lifts are also lower with a maximum lift of 15 and the p-values are bigger (figure 4.5b). There are a total of 48,015 event pairs with 2,118 of them with a support greater than 100 only these have been considered.

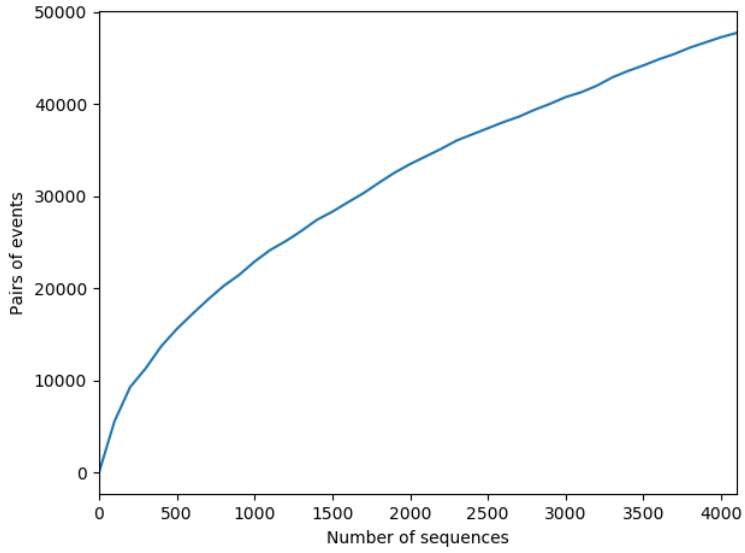
Reading the input data takes 3.9 seconds, computing the lifts and p-values 10.5 seconds, computing the states 11.1 seconds and cleaning the graph only 0.4 seconds. In total 25.9 seconds and the memory footprint is 1.43GB.

Figure 4.6 contains the trajectories found using the dataset from Sant Pau. As the number of patients is lower and the number of different dates for each sequence is also lower is much more difficult to find temporal and non-temporal relations. The minimum lift has been reduced to 1.5, the maximum p-value to 0.025 and the minimum similarity to 0.9.

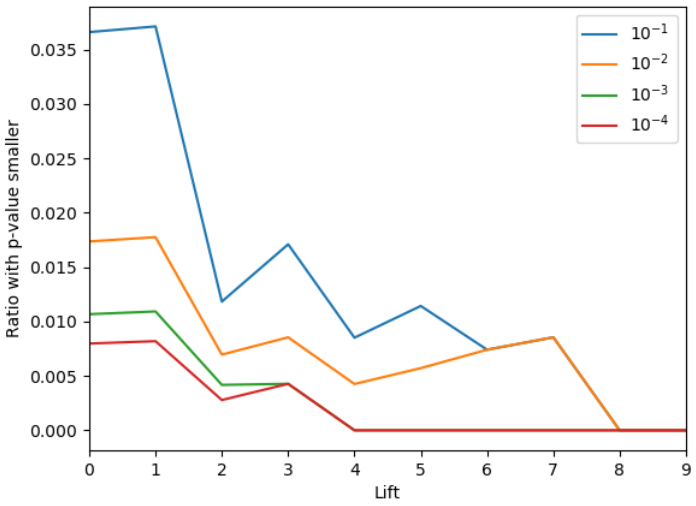
The state in the center contains many of the most common chronic diseases like cardiac insufficiency and diabetes and points to metabolism disorder, obesity and nephropathy. There is also a temporal relation between heart attack and old heart attack. Old heart attack is an anotation that the hospital uses to annotate when a patient has had a previous heart attack which obviously appears in the trajectories.

Figure 4.7 shows a state with diabetes (D.M.) in the center with a temporal relation with chronic kidney disease. Chronic kidney disease is known to occur in time to patients with diabetes [2]. In the graph it also appears chronic bronchitis on a temporal relation with secondary respiratory problems and chronic respiratory obstructions that also are related with diabetes.

[4]The International Classification of Diseases (ICD) is the international "standard diagnostic tool for epidemiology, health managment and clinical purposes" [4]



(a) Pairs of events against sequences.



(b) Ratio of significant p-values against the lift.

Figure 4.5: Counts, lifts and pvalues for the pairs of events

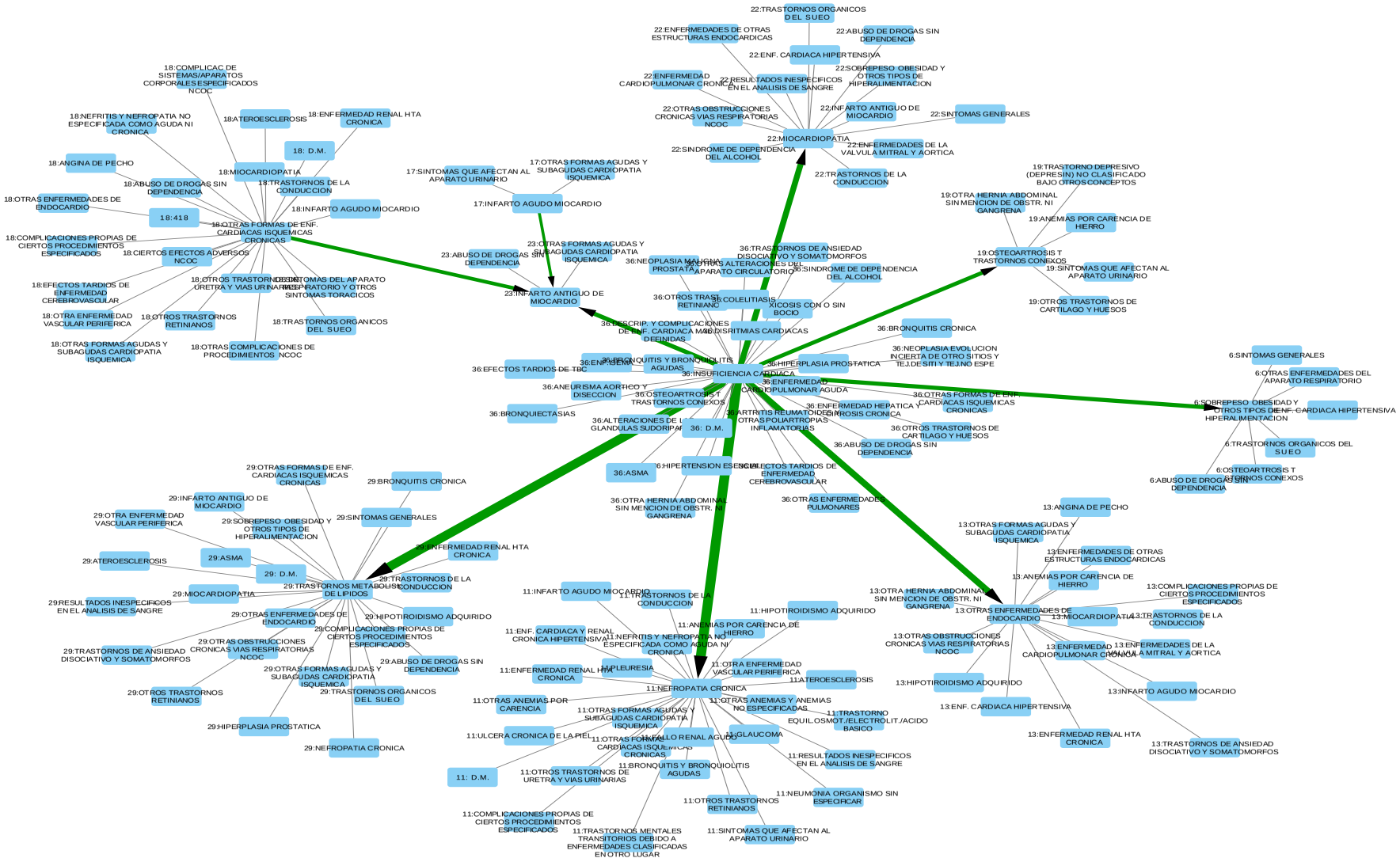


Figure 4.6: Diagnoses for heart diseases

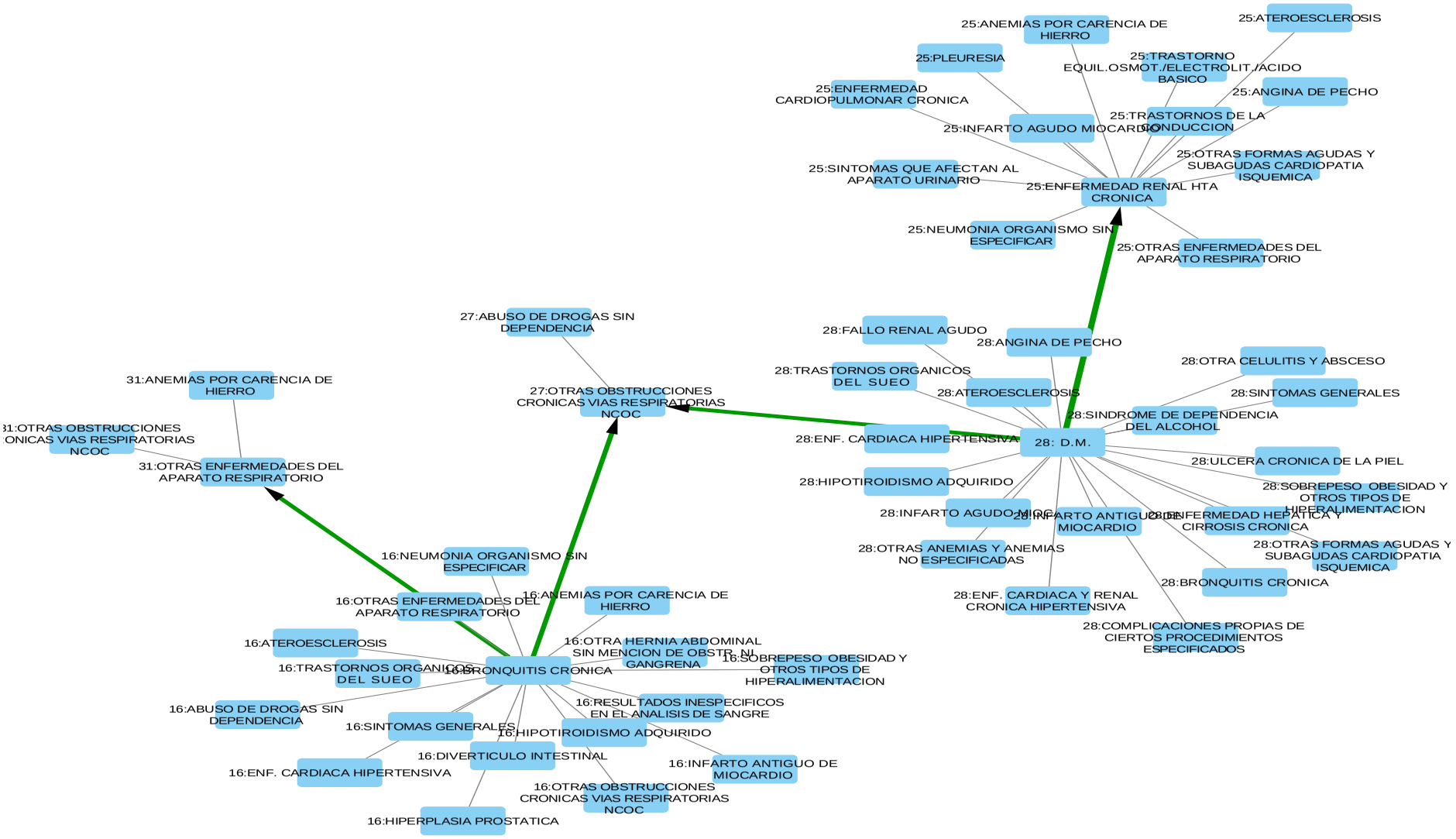


Figure 4.7: Diagnoses for diabetes and respiratory problems

Chapter 5

Conclusions & Future work

5.1 Conclusions

In this work, a new procedure for extracting *trajectory graphs* from temporally annotated sequences has been developed. The procedure has been designed to work with Electronic Health Records (EHRs) containing only the patient-id, event-id and the event-time. This is compatible with the majority of EHRs databases used in present days. The procedure is agnostic in the kind of data is fed which means it can absorb data from diagnostics, prescriptions or procedures and even combinations of them.

Previous works used in exploration of trajectories only allowed one single event in each node of the graph and each node could only appear once in the graph losing information about different groups of patients sharing a single event. To overcome this, the proposed procedure considers a state to be a set of events and each event can be repeated several times in the graph.

The procedure has been tested with data from two data sets the prescriptions table from MIMIC-III and a dataset formed by admissions from patients with heart diseases in Hospital de Sant Pau. For the MIMIC-III dataset one of the trajectory graphs that appears is composed by the prescriptions given to the premature newborn babies. It mainly consists in vitamins and some drugs to prevent coagulation of the blood and apnea. The other trajectory graph that appears is formed by the prescriptions given to the patients having some organ transplanted. The trajectories are mainly immunosupressors to prevent organ rejection. In the trajectories appear many relations with different forms of the same drug, which indicates that the dataset should be preprocessed. As the Sant Pau dataset is smaller, the lift has been decreased and the p-values and similarity to merge states increased in order to make the relations appear. The relations appearing make sense from a medical point of view but the trajectories are shorter because not many temporal relationships appear.

Regarding the performance, it can scale to real world datasets as it is easy to parallelize, the memory cost does not depend on the number of sequences. The number of events makes the cost grow quadratically as it increases because it needs to compute the counts for each pair of events. However, for the tested real-world datsets, the number of event pairs tends to stabilize rapidly as more sequences are seen. In both datasets used, the running time for the input phase is within one order of magnitude of the time to simply read the data from disk. More experiments should be performed to determine whether this statement holds in general in practical cases.

5.2 Future work

Evaluating the quality of the resulting *trajectory graphs* is a key point in order to guarantee that the procedure is working the procedure needs more work from a theoretical point of view. Also, it would be interesting to use synthetically generated data to encode some *trajectory graphs* and try to recover them after.

More work should be done together with medical experts to be able to build a tool useful for them in the exploration of clinical databases. In that line, a user interface should be created to be able to explore effectively all the computed information that remains hidden after the *cleansing* step.

It would be interesting developing a classifying technique to decide at which state may belong a patient. Knowing this would help predicting which could be the future states of his or her disease.

Chapter 6

Bibliography

[1] Anatomical therapeutic chemical classification system . https://en.wikipedia.org/wiki/Anatomical_Therapeutic_Chemical_Classification_System. Accessed: 2017-06-20.

[2] Diabetes and kidney disease. https://medlineplus.gov/ency/article/000494.htm. Accessed: 2017-06-20.

[3] Glomerulonephritis. https://medlineplus.gov/ency/article/000484.htm. Accessed: 2017-06-20.

[4] International statistical classification of diseases and related health problems - icd-9. https://en.wikipedia.org/wiki/International_Statistical_Classification_of_Diseases_and_Related_Health_Problems#ICD-9. Accessed: 2017-06-20.

[5] Iyad Batal, Hamed Valizadegan, Gregory F Cooper, and Milos Hauskrecht. A temporal pattern mining approach for classifying electronic health record data. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4(4):63, 2013.

[6] Karla L Caballero Barajas and Ram Akella. Dynamically modeling patient's health state from electronic medical records: A time series approach. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 69–78. ACM, 2015.

[7] Rafael C Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *International Colloquium on Grammatical Inference*, pages 139–152. Springer, 1994.

[8] Olive Jean Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.

[9] Pierre Dupont. Incremental regular inference. *Grammatical Interference: Learning Syntax from Sentences*, pages 222–237, 1996.

[10] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Analysis of Algorithms 2007 (AofA07)*, pages 127–146, 2007.

[11] Fosca Giannotti, Mirco Nanni, and Dino Pedreschi. Efficient mining of temporally annotated sequences. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 348–359. SIAM, 2006.

[12] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and MC Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *proceedings of the 17th international conference on data engineering*, pages 215–224, 2001.

[13] Paul Jaccard. The distribution of the flora in the alpine zone. *New phytologist*, 11(2):37–50, 1912.

[14] Anders Boeck Jensen, Pope L Moseley, Tudor I Oprea, Sabrina Gade Ellesøe, Robert Eriksson, Henriette Schmock, Peter Bjødstrup Jensen, Lars Juhl Jensen, and Søren Brunak. Temporal disease trajectories condensed from population-wide registry data covering 6.2 million patients. *Nature communications*, 5, 2014.

[15] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3, 2016.

[16] Eric Judd and David A Calhoun. Management of hypertension in ckd: beyond the guidelines. *Advances in chronic kidney disease*, 22(2):116–122, 2015.

[17] Chuanren Liu, Fei Wang, Jianying Hu, and Hui Xiong. Temporal phenotyping from longitudinal electronic health records: A graph based framework. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 705–714. ACM, 2015.

[18] Kunal Malhotra, Duen Horng Chau, Jimeng Sun, Costas Hadjipanayis, and Shamkant B Navathe. Temporal event sequence mining for glioblastoma survival prediction. In *KDD 2014 Workshop on Health Informatics (HI-KDD 2014)*, 2014.

[19] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on knowledge and data engineering*, 16(11):1424–1440, 2004.

[20] Jana A Schmidt. *Machine Learning of Timed Automata*. PhD thesis, Johannes Gutenberg Universität Mainz, 2013.

[21] Xiang Wang, David Sontag, and Fei Wang. Unsupervised learning of disease progression models. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 85–94. ACM, 2014.

[22] Mariko Yoshida, Tetsuya Iizuka, Hisako Shiohara, and Masanori Ishiguro. Mining sequential patterns including time intervals. In *AeroSense 2000*, pages 213–220. International Society for Optics and Photonics, 2000.

[23] Mohammed J Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1):31–60, 2001.