European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases
ECMLPKDD 2013

Proceedings

Workshop on Languages for Data Mining and Machine Learning

LML 2013

September 23, 2013
Prague, Czech Republic

Workshop Organization

Workshop Co-Chairs

Bruno Crémilleux *(Université de Caen, FR)*
Luc De Raedt *(KU Leuven, BE)*
Paolo Frasconi *(Università di Firenze, IT)*
Tias Guns *(KU Leuven, BE)*

Program Committee

Hendrik Blockeel
Jean-Francois Boulicaut
James Cussens
Saso Dzeroski
Elisa Fromont
Noah Goodman
Kristian Kersting
Lars Kotthoff
Nadjib Lazaar
Yuri Malitsky
Mirco Nanni
Barry O'Sullivan
Sergei O. Kuznetsov
Dino Pedreschi
Jean-Marc Petit
Avi Pfeffer
Salvatore Ruggieri
Lakhdar Sais
Sameer Singh
Arnaud Soulet
Guy Van Den Broeck
Christel Vrain

Preface

Research in Data Mining and Machine Learning has progressed significantly in the last decades, through the development of advanced algorithms and techniques. In the past few years there has been a growing attention to the development of languages for use in data mining and machine learning. Such languages provide common buildings blocks and abstractions, and can provide an alternative interface to advanced algorithms and systems that can greatly increase the utility of such systems.

Goals The workshop aims to bring together researchers and stimulate discussions on languages for data mining and machine learning. Its main motivation is the believe that designing generic and declarative modeling languages for data mining and machine learning, together with efficient solving techniques, is an attractive direction that can boost scientific progress.

Program We received 16 paper submissions among which we accepted 6 for long presentation and 3 for short presentation.

Additionally, two papers were additionally accepted for short presentation in the special oral-only track (not included in these proceedings):

- A query language for constraint-based clustering
  Antoine Adam and Hendrik Blockeel
- ParaMiner: A generic pattern mining algorithm for multi-core architectures
  Benjamin Negrevergne, Alexandre Termier, Marie-Christine Rousset and Jean-François Méhaut

The program also includes the following invited talk:

- Dino Pedreschi *(Università di Pisa)*:
  On mobility mining query languages

Website The website of the workshop contains all individual papers and links to the webpages corresponding to the papers when available:

- `http://dtai.cs.kuleuven.be/lml/`

Table of Contents

Declarative In-Network Sensor Data Analysis

George Valkanas[1], Ixent Galpin[2], Alasdair J. G. Gray[3], Alvaro A. A. Fernandes[3], Norman W. Paton[3], and Dimitrios Gunopulos[1]

[1] Dept. of Informatics & Telecommunications, University of Athens, Greece
{gvalk,dg}@di.uoa.gr
[2] Universidad Jorge Tadeo Lozano, Colombia
ixent.galpin@utadeo.edu.co
[3] School of Computer Science, University of Manchester, United Kingdom
{a.gray,alvaro,norm}@cs.man.ac.uk

Abstract. Bridging data analysis techniques with classic query processing has long been of interest in the database community. Most approaches, however, are usually developed with a specific domain in mind, e.g. relational, streaming etc., use their own query language, or focus on specific techniques. In this paper, we propose a simple, yet effective, extension to standard or commonly used declarative processing languages to support data mining. Our approach is independent of a particular domain, and by utilizing a *query refactoring* technique, optimization issues are taken care of by the underlying query processing engine, which is already in place and knows best the setting's particularities. Therefore, our approach promotes ease of programmability, development, and use of the data mining techniques, with minimal modifications in the query processing stack. We demonstrate our technique through an experimental evaluation, using our prototype system *SNEE-A*, that runs in-network data analysis given a sensor network deployment, a setting with several critical constraints.

1 Introduction

Bridging classic query processing with data analysis and mining techniques has long been of interest in the database community [15, 17, 24, 27]. Following the widespread attention that data streams and sensor networks have received in the past few years, due to their potential benefits, e.g., environmental monitoring, automated facility control etc., several approaches have been proposed [21, 29, 32]. The main advantages are an in-place infrastructure, and a higher ease of programmabing data mining techniques.

However, such approaches typically suffer from the following shortcomings: i) they propose their unique query language, ii) focus on specific algorithms, and iii) are developed having a single domain in mind. Most rely on User Defined Functions (UDFs), which have been generally criticized for being "black-boxes" and not optimization-friendly [12, 23]. Sub-optimal plans, however, impact differently one domain from the other, making these techniques unfit to port between domains. Sensor networks, for instance, are constrained on resources, face a distributed setting and dynamic environment, and are substantially different from a relational database setting. A poor execution plan that runs in a relational database simply increases running time of the query, leaving the user to wait. On the contrary, a poor plan destined to run in a sensor network could completely drain nodes from their energy and render the network useless. Therefore, it is important to optimize the code for each setting of application, but doing so manually is error prone and requires significant technical effort.

To successfully integrate query processing and data analysis, we identify the following desiderata:

i) The primary objective is to support data analysis and mining tasks, such as clustering, classification, outlier detection, etc., in a consistent way across domains, e.g. relational databases, distributed databases, or streaming sensor networks.
ii) *Efficiency* is still a major concern, although its definition is highly dependent on the domain. For instance, relational databases are interested in reducing response time, whereas for a sensor network energy consumption is a first-class citizen.
iii) Ease of programmability and development of data analysis techniques is an additional goal, as it increases a programmer's productivity. Note that the integration of new techniques needs to satisfy the *efficiency* constraint we mentioned above.
iv) Maximize adoption and ease of use by the end-users.

To address these concerns, in this paper we propose an approach that allows users to express data mining tasks through a high-level declarative language, e.g. SQL. Our **contributions** in this paper can be summarized as follows:

i) We give an approach to define and execute data analysis techniques using declarative queries. The main advantages are speed in development and deployment, a simple syntax and a system that takes care of correctness implications.
ii) We conceptualize data analysis techniques as *intensional* extents, i.e. sources whose data do not have to be acquired or stored, as opposed to *extensional* ones, and derive a flexible framework where we can combine these two types within the same query.
iii) We propose a *query refactoring* approach, motivated by the idea that several data mining algorithms can be expressed as algebraic operators. Therefore, unlike UDFs, we can leverage the optimizers that query processing engines already have.

We showcase our approach through two entirely different data analysis techniques, applied in the in-network setting and provide experimental evidence of our prototype system SNEE-A, a sensor network query processing engine that supports data mining using the discussed methodology. Our example techniques are:
Online correlation: If we know there is a correlation between the values of two measured variables, e.g. temperature and humidity, is it possible to predict humidity knowing only temperature readings and can this be done efficiently?
Outlier detection: Given a sensor network deployment that measures humidity and temperature, we want to be informed of anomalies in readings.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 shows the necessary language extensions to integrate data analysis with standard query processing, whereas Section 4 the needed modifications to the query engine. Experiments are in Section 5, and Section 6 concludes the paper.

2 Related Work

We first focus on general approaches that bring together data mining tasks and classic query processing, and then focus in more specific approaches for the in-network setting, as our application domain is such.

Unifying query execution engines with data mining techniques under a common query language has been a research topic since the mid 1990s. Support for association rule mining [15, 17, 26] and classification [24, 27] at the language level has been examined. These approaches, however, were developed for the relational setting and were too narrow on their supported algorithms. For example, both classification techniques dealt with decision trees. More importantly, though, they all employ UDFs to achieve their goals. UDFs have long been critized as effectively being "black boxes" and not optimization-friendly [12, 23]. However, the impact of non-optimizable query operators may largely vary from setting to setting. For instance, a poor execution plan in the relational setting results in longer execution times and lower user satisfaction. On the other hand, poor optimization for in-network processing is detrimental, because it can drain node energy very quickly, rendering the sensor network practically useless.

The work in [25] employs SQL queries to perform K-Means clustering through the use of triggers and vendor-specific SQL scripting extensions. However, the objective of that work is not to integrate data mining with a query language, but rather to use (relational) database technologies to perform K-Means clustering. Also note that the utilities used therein (i.e., triggers) operate differently in relational and streaming environments.

Moreover, when moving from the classic relational domain to more complex ones, e.g. streaming environments, the declarative language itself is constrained in expressiveness. To overcome this, most proposed systems and techniques introduced their own declarative query language, which is usually an SQL variation for that setting, such as CQL [7] (Continuous Query Language) and variants [18], ACQP (Acquisitional Query Processing) [22] and SNEEql [9] (SNEE query language). ESL, proposed in [21], employs User Defined Aggregates (UDAs), a subset of UDFs, thereby inheriting their drawbacks. These languages, however, for the most part, do not focus on data analysis and mining support. MMDL [29], an ESL extension, is a step forward in this direction for the streaming setting. However, as pointed out in [32], memory requirements of UDAs (therefore ESL and MMDL) cannot be clearly estimated from their syntactic structure, which does not fit well the resource-limited sensor network setting.

We now briefly discuss query execution engines for sensor network, as our application domain of choice is such. Typically, there are two lines of work in this area: i) Gather all sensed data to a central node, the *sink* and perform operations in a centralized environment or ii) view the network as a distributed processing query engine, and (partially or entirely) evaluate queries in-network.

Systems that fall under this category include STREAM (Stanford Stream Data Manager) [6], Aurora [2], Borealis [1], TelegraphCQ [20] and the more recent SMM (Stream Mill Miner) [29]. SMM is the only one among them to target specifically at data mining support. It uses UDAs, thus inheriting their drawbacks which we already discussed.

Works under the second category include the Cougar project [34], which introduced database concepts in sensor networks, as well as some in-network aggregation. Madden *et al.* developed one of the most well-known in-network query processing frameworks, TinyDB [22]. Sensor readings are represented by a relational table, optimization is limited to operator reordering and the same load is distributed among the nodes in the participating set, disregarding their position in the network topology. Despite their novelty in in-network query processing, none of them considers data mining tasks.

SNEE (Sensor NEtwork Engine) [13], is a sensor network query execution engine, optimizing queries submitted in a declarative language, *SNEEql*. SNEE considers multiple parameters that affect network efficiency, e.g. network topology, node availability, energy consumption of operators. By default, SNEE optimizes node power consumption, and maximizes network longevity. Quality of service requirements may also be imposed (e.g. delivery constraints), which effectively alter the optimization goal.

A hybrid approach is adopted by the recently presented *AnduIN* [18]. *AnduIN* uses a declarative, streaming language variant and supports data analysis techniques through UDFs at the query level. Another difference is that we model data analysis as algebraic operators and leverage the execution engine's optimizer, whereas *AnduIN* uses UDFs and evaluates code performance offline, through simulations.

Regarding in- and out of network custom data analysis and mining techniques, there is a large body of literature [4], not to mention for classic settings. However, these are stand alone solutions and not integrated with a query processing engine, which we aim for. It has also been discussed that they sometimes contradict established notions of relational databases [11], let alone streaming environments. Furthermore, in these cases, optimization issues are a responsibility of the algorithm's designer, despite the existence of optimizers in the processing engines, which we would like to take advantage of.

3 In-Network Data Analysis with a Declarative Language

Towards fulfilling our goals, we follow a holistic methodology that involves:

a) extending the declarative language appropriately, so that data analysis techniques are supported at the query language level.
b) implementing them as extensions to the query optimization stack, building on the contribution that they can be denoted by intensional extents.

For ease of discussion, we will use SNEE and SNEEql [9] as the query execution engine and language respectively. We chose SNEE due to its well-defined and modular query optimization stack, that extends the classical two-phase optimization approach from distributed query processing [19], as well as for the various optimization goals it supports. SNEEql is a declarative query language for sensor networks inspired by expressive classical stream query languages such as CQL [7]. Nevertheless, we stress that our findings apply in similar approaches where declarative languages are applicable.

3.1 Extending SNEEql

To support data analysis tasks at the declarative level, we manipulate them as any other *extensional* extent (i.e. relation, stream), leaving the query language syntax intact. As a distinction, We refer to them as *intensional* extents, i.e. sources of information for which it is not necessary that their tuples are acquired or stored.

Users create data analysis and mining tasks through `CREATE` statements, like creating a view in relational databases, which alters SNEE's metadata to accommodate the new extent. To support this functionality, we extend SNEEql's data definition language (DDL), utilizing a hierarchical decomposition of data analysis categories and their techniques. Figure 1 shows the updated DDL syntax. Tokens in bold are reserved terms, while the rest are replaced by the corresponding rule. Unmatched tokens refer to specific algorithms and their respective parameters, e.g. the value k for k-Means.

```
DDLIntExtent ::= createClause fromClause;
createClause ::= CREATE dattype [ datsubtype, datparams ]  identifier
fromClause   ::= FROM ( fromItem )
dattype      ::= CLASSIFIER | CLUSTER | SAMPLE |
                 ASSOCIATION_RULE | OUTLIER_DETECTION |
                 PROBFN | VIEW
datsubtype   ::= linearRegression |  knn | d3 | kmeans | ...
datparams    ::= paramListItem, dataparams | paramListItem
identifier   ::= Any valid identifier
fromItem     ::= Either an extent in the schema, or a sub-query
```

Fig. 1. Syntax for Defining an Intensional Extent.

```
Schema:
  AmazonForest:stream (id:int, time:ts, temperature:float)
  TropicalForestData:stream (id:int, time:ts, temperature:float, humidity:float)
```

Fig. 2. Example schema of two streams expressed in SNEEql.

3.2 Online correlation

Assume, for instance, the two extensional stream extents of Fig. 2, one for the amazon forest that reports temperature values, and a more general tropical forest stream that reports temperatures and humidity values.

Figure 3 shows the creation of a linear regression classifier over *TropicalForestData*, using tuples within a 20 minute window to construct it. We can then use that classifier in subsequent queries with *TropForestLRF* as the extent's name, as shown in Fig. 4. Here we wish to predict humidity values from the *AmazonForest* extent given its current temperature (this is what 'NOW' refers to). Incorporating intensional extents in such a way also has a natural interpretation in terms of query semantics: "Give me the humidity value of a tuple from (virtual) relation *TropForestLRF*, for which the temperature is equal to the current sensed temperature from *AmazonForest*". This makes our approach easy to understand for users who are familiar with SQL but not data analysis techniques.

Conceptually, when an intensional extent variable appears in an equality condition in the WHERE clause, what happens is akin to variable binding in logic languages, e.g. Datalog [3], after all necessary semantic checks have successfully completed. Our *query refactoring* approach makes extensive use of these value bindings.

Note that *TropForestLRF* is constantly updated, as it is an *intensional* extent, built over the *TropicalForestsData* stream. As data is acquired from that extent, the classifier is updated as well. More generally, intensional extents inherit the acquisitional properties of extensional ones, upon which they are built.

```
CREATE CLASSIFIER [linearRegression, humidity]
TropForestLRF FROM (
    SELECT RSTREAM temperature, humidity
    FROM TropicalForestData[FROM NOW-20 MIN TO NOW]
);
```

Fig. 3. Creating a Linear Regression Classifier.

```
SELECT RSTREAM AF.temperature, LRF.humidity
FROM   TropForestLRF LRF, AmazonForest[NOW] AF
WHERE  AF.temperature = LRF.temperature;
```

Fig. 4. Using the TropForestLRF intensional extent.

```
CREATE OUTLIER_DETECTION [D3, 5, 0.15] d3od
FROM (
    SELECT RSTREAM temperature
    FROM AmazonForest[FROM NOW-20 MIN TO NOW]
);
```

Fig. 5. Creating a D3 outlier detection extent.

```
SELECT RSTREAM AF.temperature
FROM   AmazonForest[NOW] AF, d3od od
WHERE  AF.temperature = od.temperature;
```

Fig. 6. Using the d3od intensional extent.

3.3 Outlier detection

Our approach for enabling in-network processing of data analysis tasks can also handle more complex constructs, such as the D3 outlier detection algorithm [28]. Detecting outliers is useful for several reasons, e.g. event indication, identification of faulty hardware, or as a first step to data cleaning.

D3 uses sampling and the Epanechnikov kernel density estimator, to approximate the distribution of sensed data. It reports data as outliers if they have low probability to have been drawn from the same underlying distribution that created their (multi-dimensional) neighboring data. D3 requires two parameters: a neighborhood *range* and a *probability* threshold. Figure 5 shows how to create a D3 intensional extent over the temperature values of *AmazonForest* from the last 20 minutes, where $range = 5$ and $probability = 15\%$. Figure 6 shows a query using that extent to check whether the most recent tuple is an outlier. As we can observe from it, intensional extents can also be part of a self-join query.

4 Query Refactoring

In this section, we present how an existing query execution infrastructure, i.e. SNEE, can be modified, so that it supports data analysis techniques. When a query is submitted, we check with SNEE's metadata whether it contains intensional extents or not. The occurrence of an intensional extent triggers its substitution by a templated subplan, which performs its algorithmic computations. We collectively refer to this process as *query refactoring* [31]. In essense, we reformulate an initially posed query into an equivalent one, that is also expressed in the same declarative language (SNEEql).

This approach has the added advantage that data analysis techniques are no longer black boxes but can leverage the engine's optimizer, without altering query semantics. Query refactoring only affects the parts of the query related to the intensional extent, leaving the rest intact. To better illustrate the needed modifications, Fig. 7a)-b) show SNEE's optimization stack with and without the query refactoring module, respectively.

The output of the query refactoring process depends on the intensional extent used. This process is transparent to the user, who will simply write the initial query. It follows that we need not perform any changes to the query language level to support data analysis in such a way. On the downside, our approach is limited by the expressive power of SNEEql. Provided that the algorithmic description of a data analysis technique can be expressed in the query language, it can then be incorporated in our approach easily. We now demonstrate how query refactoring is applied to the two examples from Section 3.

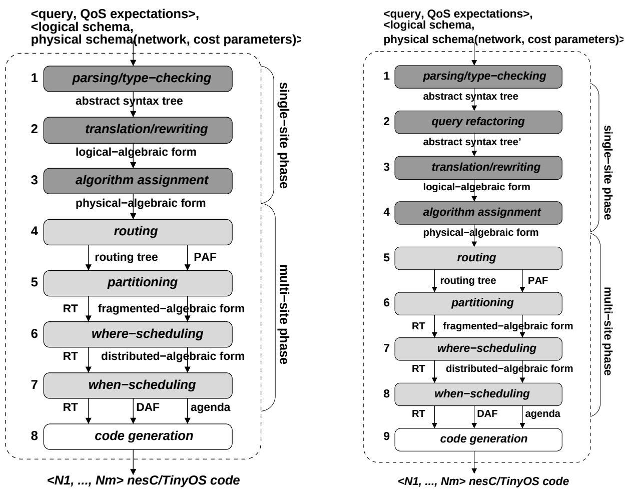


Fig. 7. SNEE optimization stack: (left) original stack, (right) query refactoring approach for data analysis techniques.

4.1 Linear Regression

We showcase the use of templated code in query refactoring through a general SNEEql query with a Linear Regression classifier. Assume an extent *LRFSource* with attributes *X* and *Y*, as shown in Fig. 8. Attribute *X* is the independent variable and *Y* is the dependent one. Values in bold are placeholders for actual extents and attribute names.

Training this classifier is equivalent to computing coefficients (a, b) based on *LRFSource*, i.e., the source over which the extent was created. Moreover, these need to be appropriately updated as new readings are acquired, as discussed in Section 3. Both of these goals can be achieved through the templated SNEEql subplan of Fig. 9.

Figure 10 shows a SNEEql query using LRF, where the last argument in the `SELECT` clause is the dependent variable of the classifier. The position of both vari-

```
CREATE CLASSIFIER [linearRegression, Y] LRF
FROM (
  SELECT RSTREAM X, Y
  FROM LRFSource
);
```

Fig. 8. Creation of a templated Linear Regression classifier.

```
SELECT RSTREAM (r.n*r.sxy - r.sx*r.sy) / (r.n*r.sxx - r.sx*r.sx) as a,
       (r.sy*r.sxx - r.sx*r.sxy) / (r.n*r.sxx - r.sx*r.sx) as b
FROM (
       SELECT RSTREAM COUNT(t.X) as n,
           SUM(t.X) as sx, SUM(t.Y) as sy,
           SUM(t.X*t.Y) as sxy, SUM(t.X*t.X) as sxx
       FROM (
           SELECT RSTREAM X, Y
           FROM LRFSource
       ) t
) r;
```

Fig. 9. Templated subquery for computing (a, b) values

```
SELECT RSTREAM $s_1$, $s_2$, ..., $s_n$, lri.Y
FROM $e_1$, $e_2$, ..., $e_m$, LRF lri
WHERE $w_1$ $OP_1$ $w_2$ $OP_2$ ... $OP_{l-1}$ $w_l$
```

Fig. 10. General form of a query using the *LRF* extent

```
SELECT RSTREAM $s_1$, $s_2$, ..., $s_n$, lri.a * Z + lri.b
FROM $e_1$, $e_2$, ..., $e_m$, (LRF_ab) lri
WHERE $w_1$ $OP_1$ $w_2$ $OP_2$ ... $OP_{l-2}$ $w_{l-1}$
```

Fig. 11. General form of a refactored query using Linear Regression

ables is insignificant; it is at the end to ease readability. Projected attributes from other extents may also appear in the `SELECT` clause. The OP_is in the `WHERE` clause are standard boolean operators, e.g., *AND*, *OR*, combining boolean expressions (the w_is).

Through SNEE's metadata, we see that *LRF* is an intensional extent, at which point query refactoring comes into play. Briefly explained, we need to do the following:

- Locate Z in the `WHERE` clause, such that ***lri.X***=Z or Z=***lri.X***. Let us assume that this is w_l in Fig. 10.
- Replace all occurrences of **lri.Y** with **lri**.$a * Z +$ **lri**.b.
- Remove w_l from the `WHERE` clause, as it will not be used anymore.
- *LRF* becomes a placeholder for the subplan of Fig. 9, and is substituted accordingly. We briefly refer to it as **LRF_ab**.

Upon completing these steps we obtain the templated form of the refactored query, shown in Fig. 11. Applying this process to the query in Fig. 4, where *TropForestLRF* is the classifier, we obtain the mappings in Table 1. Combined with the above templates, we obtain both the subplan for computing values (a, b) (Fig. 12), which substitutes **LRF_ab**, and the overall refactored SNEEql query (Fig. 13). Note that all of these operators are directly optimizable through the existing infrastructure.

4.2 D3 Outlier Detection

Figure 14 shows the refactored query of the one in Fig. 6. The **STDEV** operator computes the standard deviation of the temperature tuples in the window specified in the FROM clause, which was provided when creating the extent. Note that this is just a convenient way of writing the computation of standard deviation, which can be also expressed through additional subqueries. Furthermore, this type of notation allows us to use more efficient, approximate algorithms [8], if we see fit. Recall that, during creation, the range was set to 5 and the probability threshold to 15%. The range is used to compute the closed form of the Epanechnikov integral, whereas the probability filters

Table 1. Template variables mapping for the query in Fig. 4.

Template	Mapping

LRFSource	SELECT RSTREAM temperature, humidity
	FROM TropicalForestsData[FROM NOW-20 MIN TO NOW]

LRF	TropForestLRF
X	temperature
Y	humidity
Z	AF.temperature

```
SELECT RSTREAM AF.temperature, LRF.a * AF.temperature + LRF.b
FROM   AmazonForest[NOW] AF, (ab_COMP) LRF;
```

Fig. 12. Refactored Query of Fig. 4.

```
SELECT RSTREAM (r.n*r.sxy - r.sx*r.sy) / (r.n*r.sxx - r.sx*r.sx) as a,
       (r.sy*r.sxx - r.sx*r.sxy) / (r.n*r.sxx - r.sx*r.sx) as b
FROM (
   SELECT RSTREAM COUNT(t.temperature) as n,
          SUM(t.temperature) as sx, SUM(t.humidity) as sy,
          SUM(t.temperature*t.humidity) as sxy,
          SUM(t.temperature*t.temperature) as sxx
   FROM (
          SELECT RSTREAM temperature, humidity
          FROM   TropicalForestsData[FROM NOW-20 MIN TO NOW]
   ) t
) r;
```

Fig. 13. Subquery of (ab_COMP) in Fig. 12

points which are outliers, in the WHERE clause. We have marked the parameters with bold to distinguish them from the same values used as part of other expressions. We omit the query operator tree for the D3 refactored query due to space limitations.

4.3 Extensions

Query refactoring is a general technique, that can be applied to all settings with declarative query languages, e.g., relational, streaming, which is another advantage of our methodology. Nevertheless, expressing data analysis techniques as SNEEql queries is non-trivial in its own right. The fact that merging classical query processing with data mining has been an active reasearch topic for many years is indicative of its complexity. Additionally, given that intensional and extensional extents can now be interleaved, several query refactorings are possible, leaving room for additional optimizations.

Additional extensions include how to efficiently materialize such data mining models and reuse them. Clearly, this is not always possible. For example, materialization

```
SELECT RSTREAM od.temperature
FROM (
  SELECT x.temperature,
        ( 1/COUNT(y.temperature) ) * ( (1/4)^1 ) *
         SUM( (3 * 2 * 5 / q3.b1) -
         ( ( (x.temperature - y.temperature + 5) / q3.b1 )^3 -
           ( (x.temperature - y.temperature - 5) / q3.b1 )^3 ) ) as probability
  FROM (
    SELECT SQRT(5)*q1.sigma*(q2.rsize^(-1/5)) as b1
    FROM (
      SELECT STDEV(temperature) as sigma
      FROM AmazonForest[FROM NOW-20 MIN TO NOW SLIDE 20 MIN]
    ) q1,
    (
      SELECT COUNT(temperature) as rsize
      FROM AmazonForest[FROM NOW-20 MIN TO NOW SLIDE 20 MIN]
    ) q2
  ) q3,
  AmazonForest[now] x, AmazonForest[FROM NOW-20 MIN TO NOW SLIDE 20 MIN] y
  WHERE abs( (x.temperature - y.temperature) / q3.b1 ) < 1
  GROUP BY x.temperature
) od
WHERE od.probability < 0.15;
```

Fig. 14. Refactored query of D3 outlier detection algorithm.

is meaningful in a static environment like a relation database, but in a streaming environment, where the classifier is constantly updated as new data points arrive, such an alternative may be indifferent. What *is* interesting in both settings, however, is how to precompile and save the query operator tree of a data mining task, to subsequently integrate it in a new query, thus performing incremental optimizations. Such issues fall outside the scope of this paper but can serve as future research directions.

5 Experimental Evaluation

Efficiency in sensor networks is almost synomymous with energy consumption, which includes both CPU and radio energy consumption. To gain better insights, we also measured the number of transmitted messages and bytes. These two aspects are crucial in determining radio energy consumption. We evaluated all approaches using Avrora [30], a sensor network simulator, that provides accurate per-node statistics. All sources were written in nesC 2.0/TinyOS 2.x [14, 16] for MicaZ motes.

We will limit our experimental discussion to linear regression, due to lack of space, but also as a result of its widespread adoption as a data analysis method. Note, however, that we obtained similar results for the outlier detection technique (Fig. 22).

We experimented with various topologies and Table 2 summarizes some of their structural properties. The topologies are not directly comparable as their structural properties differ. For instance, an 8-node star-like network will behave differently from an 8-node chain. Because of this, extrapolating the results to other topologies should be performed with caution. Given a static topology during initialization, we construct a minimum-hop routing tree rooted at the sink, using one of several existing algorithms [5, 10, 33]. SNEE also uses the topology to find the best query routing tree, and does so during the routing stage of query optimization (Step 4 in Fig. 7(a)). As such, we have excluded the cost of building the routing tree from the graphs displayed below.

Given that we apply our method in a streaming setting, we experimented with various window, slide and acquisition intervals, and we will be using the following caption notation in the experimental figures for convenience: W:*w*, S:*s*, A:*a*, to signify them respectively. Varying the window and slide parameters gave similar results, so we omit these figures. Experiments were run for 300 seconds of simulated execution time. As the queries are periodical by nature, we can scale up the results for longer periods.

5.1 Handcrafted Algorithms

We implemented two baselines, both of which perform a depth-first traversal of the reverse routing tree. The sink is responsible for initiating a new tree traversal, close to the end of each acquisition interval, so that the result is reported in a timely manner.

Table 2. Structural properties of the topologies used in the experimental evaluation.

Size	Avg. Length	Max. Length	Leaf Nodes	Description
4	1.3	2	2	Tree
5	1	1	4	Star
8	2.14	3	3	Tree
9	1.75	3	4	Tree
11	1.7	2	7	Tree
12	2.18	4	6	Tree
20	1.79	4	9	Tree

In our first approach, NAÏVE, the sink probes nodes separately for data one after the other, receiving and aggregating the tuples. The second one, LC for "Local-Computation", traverses the tree and aggregates values in a postordered fashion, where each node is contacted by its parent only once within an epoch. On the contrary, SNEE optimizes queries based on a time-strict agenda. This allows nodes to contact each other at predefined times, using a push-based scheme. SNEE calculates these times by utilizing the routing tree and its optimization cost models. All approaches are graphically portrayed in Fig. 15 for a simple 4-node example. Labels denote the sequence in which node communication occurs, until we obtain the full result.

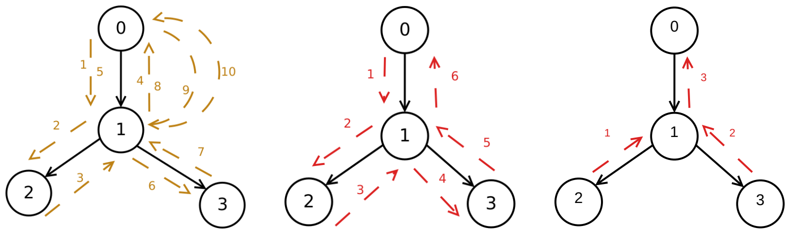
5.2 In-Network Performance

Radio Communication: Figure 16 shows the average, per-node, number of sent packets (y-axis) for the various acquisition intervals. On the x-axis, we plot the average network length, which is a better indicator of how the network is affected, than the network size. We clearly observe that in practically all cases SNEE-A's performance is superior to the handcrafted alternatives, due to its push-based scheme. All approaches perform similarly when average length is 1, due to the star-shaped topology, where each node sends directly to the sink. However, as the average network length grows higher, the difference among the techniques increases. The reason is that SNEE-A exhibits a steady performance across the topologies, which is expected as each node will send data only once during a single epoch. On the other hand, the custom techniques require the transmission of additional (control) messages to probe nodes for data.

The graphs in Fig. 17 show the average number of bytes sent in total by each node. SNEE-A and LC exhibit a steady behavior across all acquisition intervals, unlike Naïve, as a result locally aggregating the results. Even so, SNEE-A is still superior to the other two, due to the control messages that the handcrafted implementations rely on.

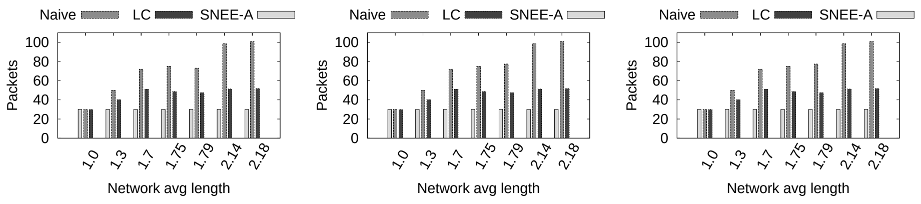
We can also see that the type of information to send depends on a combination of the sensing rate, the window size and the structural properties. For instance, for high sensing rates (Fig. 17(a)), it is preferrable to send aggregate information. However, as the sensing interval increases (Fig. 17(c)), fewer readings are taken during an epoch and sending the raw data becomes more efficient. Such optimizations are beneficial to a lot more queries when implemented within a query execution engine.

Radio Energy: Fig. 18 shows the average, minimum and maximum transmission energy consumption per node, for all techniques, clearly favoring SNEE-A. Given the push-based model and partial aggregations of SNEE-A, all three values are identical.



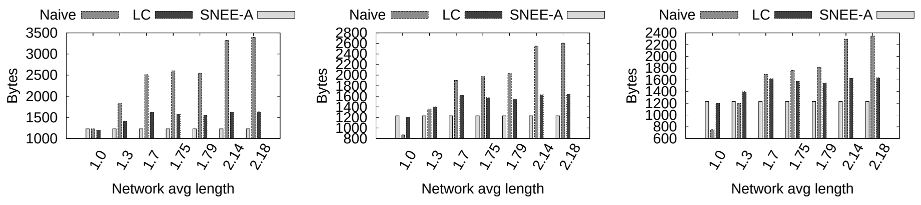
(a) Naïve approach (b) LC approach (c) SNEE-A approach

Fig. 15. Example of Linear Regression computation for Naïve, LC and SNEE-A approaches.



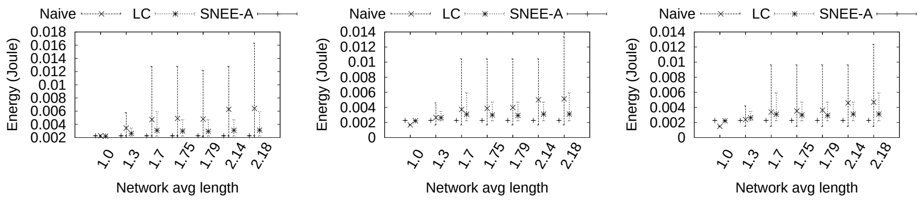
(a) W:10, S:10, A:2 (b) W:10, S:10, A:5 (c) W:10, S:10, A:10

Fig. 16. Average number of sent messages compared to the average network length for LR



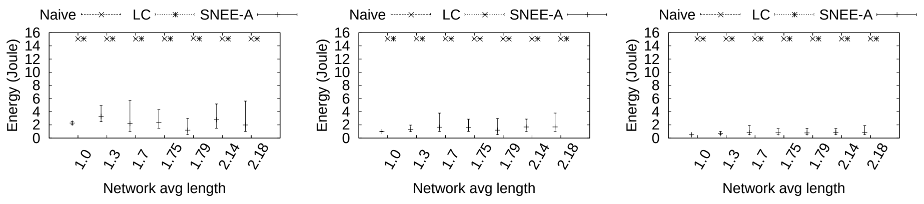
(a) W:10, S:10, A:2 (b) W:10, S:10, A:5 (c) W:10, S:10, A:10

Fig. 17. Average number of sent bytes compared to the average network length for LR



(a) W:10, S:10, A:2 (b) W:10, S:10, A:5 (c) W:10, S:10, A:10

Fig. 18. Transmission energy consumption compared to the average network length for LR

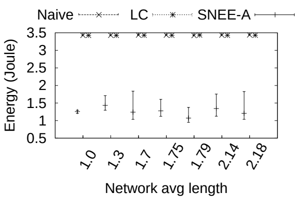


(a) W:10, S:10, A:2 (b) W:10, S:10, A:5 (c) W:10, S:10, A:10

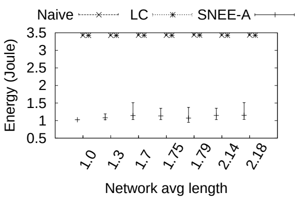
Fig. 19. Reception energy consumption compared to the average network length for LR

This is not true for the custom implementations which use control messages, the number of which is affected by the network's structural properties. Figure 18 also shows the load distribution among the nodes, with SNEE-A, distributing it almost evenly.

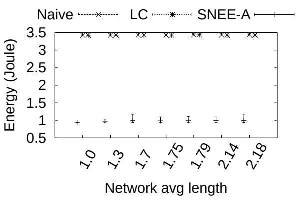
However, the factor that mostly affects radio energy consumption is the energy consumed while the radio is *on* waiting for messages, depicted in Fig. 19. The bespoke techniques have the radio switched *on* constantly, because it is impractical to manually compute when nodes will communicate. On the other hand, as a result of its agenda-



(a) W:10, S:10, A:2



(b) W:10, S:10, A:5



(c) W:10, S:10, A:10

Fig. 20. CPU energy consumption compared to the average network length for LR

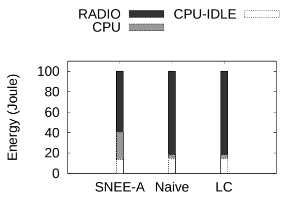
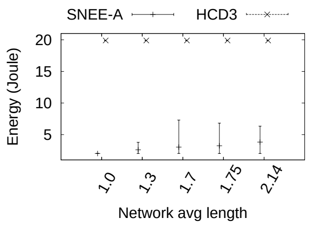
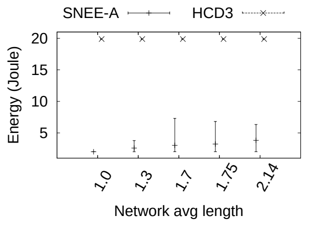


Fig. 21. Radio-CPU contribution(%) to power consumption.



(a) W:10, S:10, A:2



(b) W:10, S:10, A:10

Fig. 22. Average total energy consumption compared to the average network length for D3.

driven task execution, SNEE-A performs radio management, switching it *on* and *off* for each node independently of the others and achieves better performance.

Therefore, although we aimed for communication optimization as well, i.e., minimize transmission costs, that was not sufficient on its own. This validates our objective to utilize existing infrastructures and query engine optimizers.

CPU Usage: We finally turn our attention to the CPU consumption of the nodes. Figure 20 shows how the minimum, average and maximum CPU energy consumption is affected by the network's properties. Clearly, CPU follows a trend similar with radio reception, once again advantaging SNEE-A, for reasons we previously described.

These remarks are backed up by the graph in Fig. 21, showing the percentage with which the CPU and Radio components contribute to the total energy consumption, with an emphasis on the energy spent while the mote is idle. Even with SNEE-A's power management, the radio remains the dominant factor of consumption. However, the CPU is idle for proportionately less time compared to Naïve and LC. This implies that with SNEE-A, we make use of the resources of the node, when they are indeed required.

5.3 Ease of Programmability

One could argue that the handcrafted alternatives are inefficient because they do not use a push-based scheme. Firstly, as we already showed, most of the energy consumption is due to the radio being *idle*, which is a matter of *when* nodes communicate, rather than how they do so. Secondly, building manually such a push-based approach or even computing when nodes should communicate is impractical, as it involves accurate computation of processing and communication times for each node.

On the other hand, building a system or module that provides these accurate timings basically duplicates what the query engine already does. It is even less practical to rework this component when moving between settings (e.g., relational, streaming, distributed etc.). This brings us to another benefit of query refactoring: the time taken to write – and debug – the handcrafted code. For instance, SNEE-A has a clear advantage against the handcrafted alternatives, as i) the developer uses high-level (declarative) languages instead of low-level, and ii) the system autogenerates and deploys code for all nodes in the network, optimized for the requested goal. Finally, note that if we change the optimization goal, the bespoke techniques must be re-implemented, whereas for SNEE-A (and similar execution engines) it is a single parameter.

6 Conclusions and Future Work

In this paper we tackled the problem of integrating data analysis techniques with classic query processing through declarative languages. We proposed a query language extension, that incorporates major data mining categories, e.g. classification, outlier detection etc. We expressed data analysis tasks as intensional extents, and took advantage of existing query optimizers, with minimal modifications to the query execution stack. This also gives a natural interpretation in relational algebra terms. We implemented the above concepts in our prototype system SNEE-A, and compared its performance against handcrafted implementations.

We plan to incorporate additional techniques, to identify the limits of our approach and ways to overcome them using the framework we have presented herein. Interesting future directions include materialization of data mining models and support for incremental optimization.

Acknowledgements: This work has been funded by EU project SemSorGrid4Env (FP7-223913), INSIGHT and THALIS-DISFER, the UK EPSRC WINES Programme and a Heraclitus II Fellowship.

References

1. D. J. Abadi, Y. Ahmad, M. Balazinska, and U. Ç. et al. The design of the borealis stream processing engine. CIDR, pages 277–289, January 2005.
2. D. J. Abadi, D. Carney, U. Çetintemel, and M. e. a. Cherniack. Aurora: a new model and architecture for data stream management. *VLDB J.*, 12:120–139, August 2003.
3. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
4. C. Aggarwal, editor. *Data Streams – Models and Algorithms*. Springer, 2007.
5. P. Andreou, D. Zeinalipour-Yazti, P. K. Chrysanthis, and G. Samaras. Workload-aware query routing trees in wireless sensor networks. MDM, pages 189–196, 2008.
6. A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. Stream: the stanford stream data manager (demonstration description). SIGMOD, 2003.
7. A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *VLDB J.*, 15:121–142, June 2006.
8. B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. PODS, pages 234–243, 2003.
9. C. Y. Brenninkmeijer, I. Galpin, A. A. Fernandes, and N. W. Paton. A semantics for a query language over sensors, streams and relations. BNCOD, pages 87–99, 2008.
10. G. Chatzimilioudis, A. Cuzzocrea, and D. Gunopulos. Optimizing query routing trees in wireless sensor networks. In *ICTAI (2)*, pages 315–322, 2010.
11. S. Chaudhuri. Data mining and database systems: Where is the intersection? *Data Engineering Bulletin*, 21, 1998.

12. S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. *TODS*, 1999.
13. I. Galpin, C. Y. A. Brenninkmeijer, A. J. G. Gray, F. Jabeen, A. A. A. Fernandes, and N. W. Paton. Snee: a query processor for wireless sensor networks. *Distributed and Parallel Databases*, 29(1–2):31–85, 2011.
14. D. Gay, P. Levis, J. R. von Behren, M. Welsh, E. A. Brewer, and D. E. Culler. The nesc language: A holistic approach to networked embedded systems. PLDI, pages 1–11, 2003.
15. J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. Dmql: A data mining query language for relational databases. In *Proc. of the SIGMOD workshop on Research issues on Data Mining and knowledge discovery*, pages 27–33, 1996.
16. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. ASPLOS, pages 93–104, 2000.
17. T. Imieliński and A. Virmani. Msql: A query language for database mining. *Data Mining and Knowledge Discovery*, 3(4):373–408, 1999.
18. D. Klan, M. Karnstedt, K. Hose, L. Ribe-Baumann, and K.-U. Sattler. Stream engines meet wireless sensor networks: cost-based planning and processing of complex queries in anduin. *Distributed and Parallel Databases*, 29(1–2):151–183, 2011.
19. D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, 2000.
20. S. Krishnamurthy, S. Chandrasekaran, O. Cooper, and A. D. et al. Telegraphcq: An architectural status report. *IEEE Data Engineering Bulletin*, 26(1):11–18, March 2003.
21. C. Luo, H. Thakkar, H. Wang, and C. Zaniolo. A native extension of sql for mining data streams. SIGMOD, pages 873–875, 2005.
22. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: An acquisitional query processing system for sensor networks. *TODS*, 30(1):122–173, 2005.
23. G. Mitchell, S. B. Zdonik, and U. Dayal. Object-oriented query optimization: What's the problem? Technical report, Providence, RI, USA, 1991.
24. A. Netz, S. Chaudhuri, J. Bernhardt, and U. M. Fayyad. Integration of data mining with database technology. In *VLDB*, pages 719–722, 2000.
25. C. Ordonez. Integrating k-means clustering with a relational dbms using sql. *IEEE TKDE*, 18:188–201, February 2006.
26. S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. *Data Mining and Knowledge Discovery*, 4(2):89–125, 2000.
27. K.-U. Sattler and O. Dunemann. Sql database primitives for decision tree classifiers. CIKM, pages 379–386, 2001.
28. S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. VLDB, pages 187–198, 2006.
29. H. Thakkar, N. Laptev, H. Mousavi, B. Mozafari, V. Russo, and C. Zaniolo. Smm: A data stream management system for knowledge discovery. ICDE, pages 757–768, April 2011.
30. B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. IPSN, 2005.
31. G. Valkanas, D. Gunopulos, I. Galpin, A. J. G. Gray, and A. A. A. Fernandes. Extending query languages for in-network query processing. MobiDE, pages 34–41, 2011.
32. H. Wang and C. Zaniolo. Atlas: A native extension of sql for data mining. SDM, 2003.
33. Y. Yang, H.-H. Wu, and H.-H. Chen. SHORT: shortest hop routing tree for wireless sensor networks. *International Journal of Sensor Networks*, 2:368–374, July 2007.
34. Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9–18, 2002.

Mining (Soft-) Skypatterns using Constraint Programming

W. Ugarte[1], P. Boizumault[1], S. Loudni[1], B. Crémilleux[1], and A. Lepailleur[2]

[1] GREYC (CNRS UMR 6072) – University of Caen
Campus II Côte de Nacre, 14000 Caen - France
[2] CERMN (UPRES EA 4258 - FR CNRS 3038 INC3M) – University of Caen
Boulevard Becquerel, 14032 Caen Cedex - France

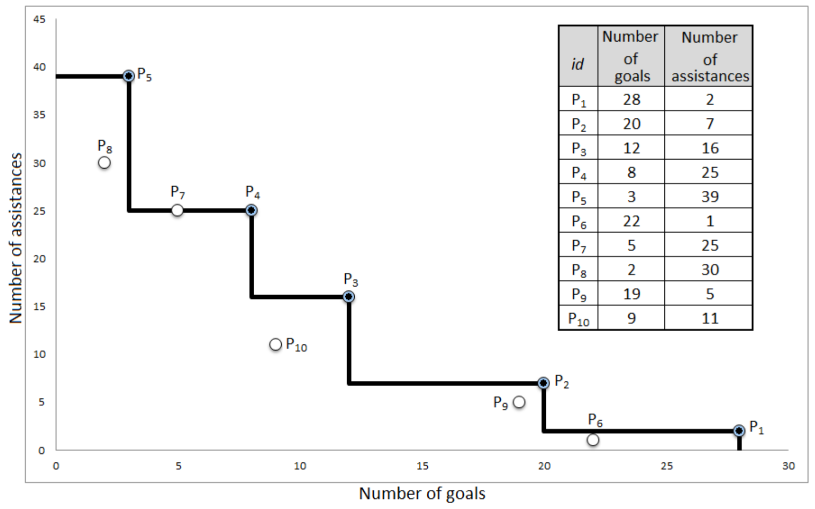
Abstract. Within the pattern mining area, skypatterns enable to express a user-preference point of view according to a dominance relation. In this paper, we deal with the introduction of softness in the skypattern mining problem. First, we show how softness can provide convenient patterns that would be missed otherwise. Then, thanks to Constraint Programming, we propose a generic and efficient method to mine skypatterns as well as soft ones. Finally, we show the relevance and the effectiveness of our approach through a case study in chemoinformatics.

1 Introduction

Discovering useful patterns from data is an important tool for data analysis and has been used in a wide range of applications. Many approaches have promoted the use of constraints to focus on the most promising knowledge according to a potential interest given by the final user. As the process usually produces a large number of patterns, a large effort is made to a better understanding of the fragmented information conveyed by the patterns and to produce *pattern sets* i.e. sets of patterns satisfying properties on the whole set of patterns [5].

Skyline queries [3] enable to express a user-preference point of view according to a *dominance* relation. In a multidimensional space where a preference is defined for each dimension, a point p_i *dominates* another point p_j if p_i is better (i.e., more preferred) than p_j in at least one dimension, and p_i is not worse than p_j on every other dimension. However, while this notion of skylines has been extensively developed and researched for database applications, it has remained unused until recently for data mining purposes. [17] proposes a technique to extract skyline graphs that maximize two measures (the number of vertices and the edge connectivity). The notion of skyline queries has been recently integrated into the constraint-based pattern discovery paradigm to mine skyline patterns (henceforth called *skypatterns*) [19]. As an example, a user may prefer a pattern with a high frequency, large length and a high confidence. In this case, we say that a pattern x_i dominates another pattern x_j if $freq(x_j) \geq freq(x_i)$, $size(x_j) \geq size(x_i)$, $confidence(x_j) \geq confidence(x_i)$ where at least one strict inequality holds. Given a set of patterns, the skypattern set contains the patterns that are not dominated by any other pattern. Skypatterns are interesting for a twofold reason: they do not require any threshold on the measures and the notion of dominance provides a global interest with semantics easily understood by the user.

Nevertheless, skypatterns queries, like other kinds of queries, suffer from the stringent aspect of the constraint-based framework. Indeed, a pattern satisfies or does not



id	Number of goals	Number of assistances
P_1	28	2
P_2	20	7
P_3	12	16
P_4	8	25
P_5	3	39
P_6	22	1
P_7	5	25
P_8	2	30
P_9	19	5
P_{10}	9	11

Fig. 1: A skyline example.

satisfy the constraints. But, what about patterns that slightly miss a constraint? The following example shows the interest of introducing softness. This example addresses *skylines in databases* because it is easier to illustrate the key points of introducing softness and to give rise the skypattern problem. Skypatterns and soft-skypatterns are formally introduced in the following sections. There are very few works such as [2, 21] which introduce softness into the mining process.

Consider a coach of a football team who looks for players for the next season (see Fig. 1). Every player is depicted according to the number of goals he scored and the number of assistances he performed during the last season. A point (here, a player) p_i *dominates* another point p_j if p_i is better than p_j in at least one dimension, and p_i is not worse than p_j on every other dimension. A skyline point is a point which is not dominated by any other point. The skyline set (or skyline for short) consists of players p_1, p_2, p_3, p_4 and p_5. Indeed, players p_6, p_7, p_8, p_9 and p_{10} are dominated by at least one other player, thus they cannot be part of the skyline. Nevertheless, the coach could be interested in non-skyline players if he looks for:

– players in a forward position: the coach gives the priority to the number of scored goals. The players p_1 (skyline), p_2 (skyline) are still interesting and p_6 (non-skyline) and p_9 (non-skyline) become interesting.
– players in an attacking midfielder position: the coach gives the priority to the number of performed assistances. The players p_4 (skyline) and p_5 (skyline) are still interesting and p_7 (non-skyline) and p_8 (non-skyline) become interesting.
– multipurpose players: the coach gives the priority to the trade-off between the number of scored goals and the number of performed assistances. The players p_3 (skyline) and p_4 (skyline) are still promising and p_{10} (non-skyline) becomes promising.

Moreover, skyline players are very sought and expensive: they might be signed by another team or their salaries could be out of budget. So, non-skyline players, that are close to skyline players, can be of great interest for the coach. Such promising players can be discovered by slightly relaxing the dominance relation.

Trans.	Items
t_1	B E F
t_2	B C D
t_3	A E F
t_4	A B C D E
t_5	B C D E
t_6	B C D E F
t_7	A B C D E F

Item	A	B	C	D	E	F
Price	30	40	10	40	70	55

Table 1: Transactional dataset $\mathscr{T}$.

The contributions of this paper are the following. First, we introduce the notion of soft skypattern. Second, we propose a flexible and efficient approach to mine skypatterns as well as soft ones thanks to the Dynamic CSP (Constraint Satisfaction Problems) framework [22]. Our proposition benefits from the recent progress on cross-fertilization between data mining and Constraint Programming (CP) [4, 9, 7]. The common point of all these methods is to model in a declarative way pattern mining as CSP, whose resolution provides the complete set of solutions satisfying all the constraints. We show how the (soft-)skypatterns mining problem can be modeled and solved using dynamic CSPs. A major advantage of the method is to improve the mining step during the process thanks to constraints dynamically posted and stemming from the current set of candidate skypatterns. Moreover, the declarative side of the CP framework leads to a unified framework handling softness in the skypattern problem. Finally, the relevance and the effectiveness of our approach is highlighted through a case study in chemoinformatics for discovering toxicophores.

This paper is organized as follows. Section 2 presents the context and defines skypatterns. Section 3 introduces soft skypatterns. Section 4 presents our flexible and efficient CP approach to mine skypatterns as well as soft ones. We review some related work in Section 5. Finally, Section 6 reports in depth a case study in chemoinformatics by performing both a performance and a qualitative analysis.

2 The skypattern mining problem

2.1 Context and definitions

Let $\mathscr{I}$ be a set of distinct literals called *items*. An itemset (or pattern) is a non-null subset of $\mathscr{I}$. The language of itemsets corresponds to $\mathscr{L}_{\mathscr{I}} = 2^{\mathscr{I}} \backslash \emptyset$. A transactional dataset $\mathscr{T}$ is a multiset of patterns of $\mathscr{L}_{\mathscr{I}}$. Each pattern (or transaction) is a database entry. Table 1 (left side) presents a transactional dataset $\mathscr{T}$ where each transaction t_i gathers articles described by items denoted $A,\ldots,F$. The traditional example is a supermarket database in which each transaction corresponds to a customer and every item in the transaction to a product bought by the customer. An attribute (*price*) is associated to each product (see Table 1, right side).

Constraint-based pattern mining aims at extracting all patterns x of $\mathscr{L}_{\mathscr{I}}$ satisfying a query $q(x)$ (conjunction of constraints) which is usually called *theory* [12]: $Th(q) = \{x \in \mathscr{L}_{\mathscr{I}} \mid q(x) \textit{ is true}\}$. A common example is the frequency measure leading to the minimal frequency constraint. The latter provides patterns x having a number of occurrences in the dataset exceeding a given minimal threshold min_{fr}: $freq(x) \geq min_{fr}$. There are other usual measures for a pattern x:

- $size(x)$ is the number of items that x contains.

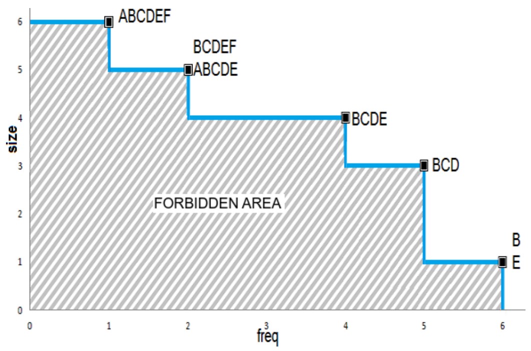


Fig. 2: Skypatterns extracted from the example in Table 1.

- $area(x) = freq(x) \times size(x)$.
- $min(x.val)$ is the smallest value of the item values of x for attribute val.
- $max(x.val)$ is the highest value of the item values of x for attribute val.
- $average(x.val)$ is the average value of the item values of x for attribute val.
- $mean(x) = (min(x.val) + max(x.val))/2$.

Considering the dataset described in Table 1, we have: $freq(BC)$=5, $size(BC)$=2 and $area(BC)$=10. Moreover, $average(BCD.price)$=30 and $mean(BCD.price)$=25.

In many applications, it is highly appropriated to look for contrasts between subsets of transactions, such as toxic and non toxic molecules in chemoinformatics (see Section 6). The growth rate is a well-used contrast measure [14].

Definition 1 (Growth rate). *Let $\mathscr{T}$ be a database partitioned into two subsets $\mathscr{D}_1$ and $\mathscr{D}_2$. The growth rate of a pattern x from $\mathscr{D}_2$ to $\mathscr{D}_1$ is:*

$$m_{gr}(x) = \frac{|\mathscr{D}_2| \times freq(x, \mathscr{D}_1)}{|\mathscr{D}_1| \times freq(x, \mathscr{D}_2)}$$

Moreover, the user is often interested in discovering richer patterns satisfying properties involving several local patterns. These patterns define pattern sets [5] or n-ary patterns [9]. The approach presented in this paper is able to deal with such patterns.

2.2 Skypatterns

Skypatterns have been recently introduced by [19]. Such patterns enable to express a user-preference point of view according to a dominance relation. Given a set of patterns, the skypattern set contains the patterns that are not dominated by any other pattern.

Given a set of measures M, if a pattern x_j is dominated by another pattern x_i according to all measures of M, x_j is considered as irrelevant. This idea is at the core of the notion of skypattern.

Definition 2 (Dominance). *Given a set of measures M, a pattern x_i dominates another pattern x_j with respect to M (denoted by $x_i \succ_M x_j$), iff $\forall m \in M, m(x_i) \geq m(x_j)$ and $\exists m \in M, m(x_i) > m(x_j)$.*

Consider the example in Table 1 with M=$\{freq, area\}$. Pattern BCD dominates pattern BC because $freq(BCD)$=$freq(BC)$=5 and $area(BCD)$>$area(BC)$. For M=$\{freq, size, average\}$, pattern BDE dominates pattern BCE because $freq(BDE)$=$freq(BCE)$=4, $size(BDE)$=$size(BCE)$=3 and $average(BDE.price)$>$average(BCE.price)$.

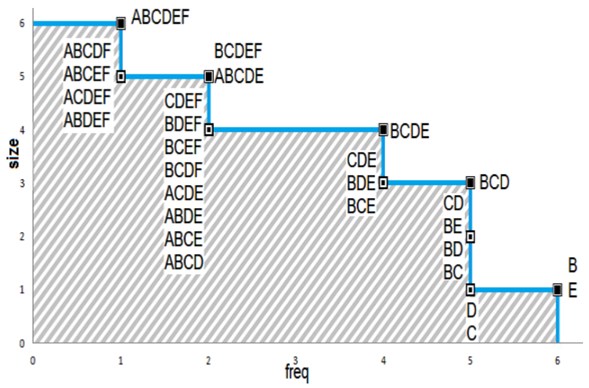


Fig. 3: Edge-skypatterns extracted from the example in Table 1.

Definition 3 (Skypattern operator). *Given a pattern set $P \subseteq \mathscr{L}_{\mathscr{I}}$ and a set of measures M, a skypattern of P with respect to M is a pattern not dominated in P with respect to M. The skypattern operator Sky(P,M) returns all the skypatterns of P with respect to M: Sky(P,M)* $= \{x_i \in P \mid \nexists x_j \in P, x_j \succ_M x_i\}$.

The skypattern mining problem is thus to evaluate the query $Sky(\mathscr{L}_{\mathscr{I}}, M)$. For instance, from the data set in Table 1 and with $M = \{freq, size\}$, $Sky(\mathscr{L}_{\mathscr{I}}, M) = \{ABCDEF, BCDEF, ABCDE, BCDE, BCD, B, E\}$ (see Figure 2). The shaded area is called the *forbidden area*, as it cannot contain any skypattern. The other part is called the *dominance area*. The edge of the dominance area (bold line) marks the boundary between these two zones.

The skypattern mining problem is challenging because of its NP-Completeness. There are $O(2^{|\mathscr{I}|})$ candidate patterns and a naive enumeration would lead to compute $O(2^{|\mathscr{I}|} \times |M|)$ measure values. [19] have proposed an efficient approach taking benefit of theoretical relationships between pattern condensed representations and skypatterns and making the process feasible when the pattern condensed representation can be extracted. Nevertheless, this method can only use a crisp dominance relation.

3 The soft skypattern mining problem

This section presents the introduction of softness in the skypattern mining problem. The skypatterns suffer from the stringent aspect of the constraint-based framework. In order to introduce softness in this context, we propose two kinds of soft skypatterns: the *edge-skypatterns* that belongs to the edge of the dominance area (see Section 3.1) and the δ-*skypatterns* that are close to this edge (see Section 3.2).

The key idea is to strengthen the dominance relation in order to soften the notion of non dominated patterns. The goal is to capture valuable skypatterns occurring in the forbidden area.

3.1 Edge-skypatterns

Similarly to skypatterns, edge-skypatterns are defined according to a dominance relation and a *Sky* operator. These two notions are reformulated as follows:

Definition 4 (Strict Dominance). *Given a set of measures M, a pattern x_i strictly dominates a pattern x_j with respect to M (denoted by $x_i \gg_M x_j$), iff $\forall m \in M$, $m(x_i) > m(x_j)$.*

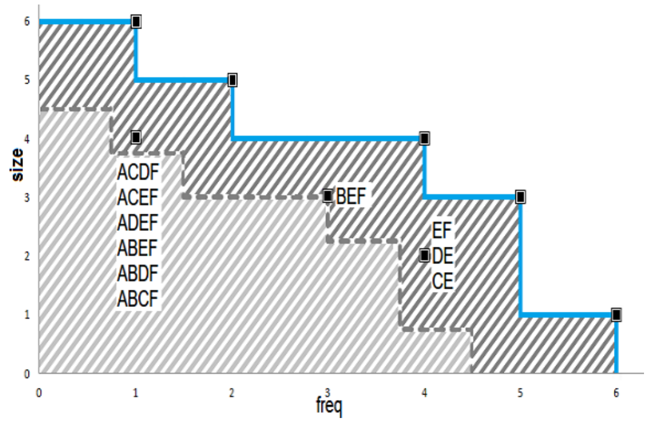


Fig. 4: δ-skypatterns (that are not edge ones) extracted from the example in Table 1.

Definition 5 (Edge-skypattern operator). *Given a pattern set $P \subseteq \mathscr{L}_{\mathscr{I}}$ and a set of measures M, an edge-skypattern of P, with respect to M, is a pattern not strictly dominated in P, with respect to M. The operator Edge-Sky(P,M) returns all the edge-skypatterns of P with respect to M: Edge-Sky$(P,M) = \{x_i \in P \mid \nexists x_j \in P, x_j \gg_M x_i\}$*

Given a set of measures M, the edge-skypattern mining problem is thus to evaluate the query *Edge-Sky*(P,M). Fig. 3 depicts the 28=7+(4+8+3+4+2) edge-skypatterns extracted from the example in Table 1 for M=$\{freq, size\}$. Obviously, all edge-skypatterns belong to the edge of the dominance area, and seven of them are skypatterns (see Fig. 2).

Proposition 1. *For two patterns x_i and x_j, $x_i \gg_M x_j \implies x_i \succ_M x_j$. So, for a pattern set P and a set of measures M, Sky$(P,M) \subseteq$ Edge-Sky(P,M).*

3.2 δ-skypatterns

In many cases the user may be interested in skypatterns expressing a trade-off between the measures. The δ-skypatterns address this issue where δ means a percentage of relaxation allowed by the user. Let $0 < \delta \leq 1$.

Definition 6 (δ-Dominance). *Given a set of measures M, a pattern x_i δ-dominates another pattern x_j w.r.t. M (denoted by $x_i \succ_M^{\delta} x_j$), iff $\forall m \in M$, $(1-\delta) \times m(x_i) > m(x_j)$.*

Definition 7 (δ-Skypattern operator). *Given a pattern set $P \subseteq \mathscr{L}_{\mathscr{I}}$ and a set of measures M, a δ-skypattern of P with respect to M is a pattern not δ-dominated in P with respect to M. The δ-skypattern operator δ-Sky(P,M) returns all the δ-skypatterns of P with respect to M: δ-Sky$(P,M) = \{x_i \in P \mid \nexists x_j \in P : x_j \succ_M^{\delta} x_i\}$.*

The δ-skypattern mining problem is thus to evaluate the query δ-*Sky*(P,M). There are 38 (28+10) δ-skypatterns extracted from the example in Table 1 for M=$\{freq, size\}$ and δ=0.25. Fig. 4 only depicts the 10 δ-skypatterns that are not edge-skypatterns. Intuitively, the δ-skypatterns are close to the edge of the dominance relation, the value of δ is the maximal relative distance between a skypattern and this border.

Proposition 2. *For two patterns x_i and x_j, $x_i \succ_M^{\delta} x_j \implies x_i \gg_M x_j$. So, for a pattern set P and a set of measures M, Edge-Sky$(P,M) \subseteq \delta$-Sky(P,M).*

To conclude, given a pattern set $P \subseteq \mathscr{L}_{\mathscr{I}}$ and a set of measures M, the following inclusions hold: *Sky*$(P,M) \subseteq$ *Edge-Sky*$(P,M) \subseteq \delta$-*Sky*(P,M).

4 Mining (soft-) skypatterns using CP

This section describes how the skypattern and the soft skypattern mining problems can be modeled and solved using Dynamic CSP [22]. A major advantage of this approach is to improve the mining step during the process thanks to constraints dynamically posted and stemming from the current set of the candidate skypatterns. Each time a solution is found, we dynamically post a new constraint leading to reduce the search space. This process stops when we cannot enlarge the forbidden area. Finally, the completeness of our approach is insured by the completeness of the CP solver. The implementation of our approach has been carried out in `Gecode`[1] extending the (CP based) pattern extractor developed by [9].

4.1 CSP and Dynamic CSP

A Constraint Satisfaction Problem (CSP) $P=(\mathscr{X},\mathscr{D},\mathscr{C})$ is defined by:

- a finite set of variables $\mathscr{X} = \{x_1, x_2, \ldots, x_k\}$,
- a domain $\mathscr{D}$, which maps every variable $x_i \in \mathscr{X}$ to a finite set of values $D(x_i)$,
- a finite set of constraints $\mathscr{C}$.

Algorithm 1 [7] shows how a CSP can be solved using a depth-first search. D and C denote respectively the current domains and the current set of constraints. In each node of the search tree, the algorithm branches by assigning values to a variable that is unfixed (line 7). It backtracks when a violation of constraints is found, i.e. at least one domain is empty (line 2). The search is further optimized by carefully choosing the variable that is fixed next (line 5); for instance, heuristics dom/deg selects the variable x_i having the smallest ratio between the size of its current domain and the number of constraints it occurs. The main concept used to speed-up the search is Filtering (constraint propagation) (line 1). Filtering reduces the domains of variables such that the domain remains locally consistent. A solution is obtained (line 9) when each domain $D(x_i)$ is reduced to a singleton and all constraints are satisfied.

```
Algorithm 1: Constraint-Search($D$,$C$)
1 $D \leftarrow propagate(D,C)$;
2 if there exists $x_i \in \mathscr{X}$ s.t. $D(x_i)$ is empty then
3     return failure;
4 if there exists $x_i \in \mathscr{X}$ s.t. $|D(x_i)| > 1$ then
5     Select $x_i \in \mathscr{X}$ s.t. $|D(x_i)| > 1$;
6     forall $v \in D(x_i)$ do
7         Constraint-Search($D \cup \{x_i -> \{v\}\}$);
8 else
9     output solution $D$;
```

A Dynamic CSP [22] is a sequence $P_1, P_2, ..., P_n$ of CSP, each one resulting from some changes in the definition of the previous one. These changes may affect every component in the problem definition: variables (addings or removals), domains (value addings or removals), constraints (addings or removals). For our approach, changes are only performed by adding new constraints.

[1]http://www.gecode.org/

Solving such dynamic CSP involves solving a single CSP with additional constraints posted during search. Each time a new solution is found, new constraints $\phi(\mathscr{X})$ are imposed. Such constraints will survive backtracking and state that next solutions should verify both the current set of constraints C and $\phi(\mathscr{X})$. So line 9 of Algorithm 1 becomes:

```
1 Output solution $D$;
2 $C \leftarrow C \cup \{\phi(\mathscr{X})\}$
```

Note that C is a variable global to all calls to procedure *Constraint-Search*(D,C).

4.2 Mining skypatterns using Dynamic CSP

This section describes our CP approach for mining both skypatterns and soft skypatterns. Constraints on the dominance relation are dynamically posted during the mining process and softness is easily introduced using such constraints.

Variable x will denote the (unknown) skypattern we are looking for. Changes are only performed by adding new constraints (see Section 4.1). So, we consider the sequence $P_1, P_2, ..., P_n$ of CSP where each $P_i = (\{x\}, \mathscr{L}, q_i(x))$ and:

- $q_1(x) = closed_M(x)$
- $q_{i+1}(x) = q_i(x) \wedge \phi_i(x)$ where s_i is the first solution to query $q_i(x)$

First, the constraint $closed_M(x)$ states that x must be a closed pattern w.r.t all the measures of M, it allows to reduce the number of redundant patterns[2]. Then, the constraint $\phi_i(x) \equiv \neg(s_i \succ_M x)$ states that the next solution (which is searched) will not be dominated by s_i. Using a short induction proof, we can easily argue that query $q_{i+1}(x)$ looks for a pattern x that will not be dominated by any of the patterns s_1, s_2, $\ldots$, s_i.

Each time the first solution s_i to query $q_i(x)$ is found, we dynamically post a new constraint $\phi_i(x)$ leading to reduce the search space. This process stops when we cannot enlarge the forbidden area (i.e. there exits n s.t. query $q_{n+1}(x)$ has no solution). For skypatterns, $\phi_i(x)$ states that $\neg(s_i \succ_M x)$ (see Definition 2):

$$\phi_i(x) \equiv (\bigvee_{m \in M} m(s_i) < m(x)) \vee (\bigwedge_{m \in M} m(s_i) = m(x))$$

But, the n extracted patterns s_1, s_2, $\ldots$, s_n are not necessarily all skypatterns. Some of them can only be "intermediate" patterns simply used to enlarge the forbidden area. A post processing step must be performed to filter all candidate patterns s_i that are not skypatterns, i.e. for which there exists s_j $(1 \leq i < j \leq n)$ s.t. s_j dominates s_i. So mining skypatterns is achieved in a two-steps approach:

1. Compute the set $S = \{s_1, s_2, \ldots, s_n\}$ of candidates using Dynamic CSP.
2. Filter all patterns $s_i \in S$ that are not skypatterns.

While the number of candidates (n) could be very large (the skypattern mining problem is NP-complete), it remains reasonably-sized in practice for the experiments we conducted (seeTable 2 for the case study in chemoinformatics.)

4.3 Mining soft skypatterns using Dynamic CSP

Soft skypatterns are processed exactly the same way. Each kind of soft skypatterns has its own constraint $\phi_i(x)$ according to its relation of dominance.

[2]The *closed* constraint is used to reduce pattern redundancy. Indeed, **closed skypatterns** make up an exact condensed representation of the whole set of skypatterns [19].

For edge-skypatterns, $\phi_i(x)$ states that $\neg(s_i \gg_M x)$ (see Definition 4):

$$\phi_i(x) \equiv \bigvee_{m \in M} m(s_i) \leq m(x)$$

For δ-skypatterns, $\phi_i(x)$ states that $\neg(s_i \succ_M^{\delta} x)$ (see Definition 6):

$$\phi_i(x) \equiv \bigvee_{m \in M} (1-\delta) \times m(s_i) < m(x)$$

As previously, the n extracted patterns s_1, s_2, ..., s_n are not necessarily all soft skypatterns. So, a post processing is required as for skypatterns (see Section 4.2). Mining soft skypatterns is also achieved in a two-steps approach:

1. Compute the set $S = \{s_1, s_2, \ldots, s_n\}$ of candidates using Dynamic CSP.
2. Filter all patterns $s_i \in S$ that are not soft skypatterns.

Once again, the number of candidates (n) remains reasonably-sized in practice for the experiments we conducted (see Table 3).

Pattern variables are set variables represented by their characteristic function with boolean variables. [4, 7] model an unknown pattern x and its associated dataset $\mathscr{T}$ by introducing two sets of boolean variables: $\{X_i \mid i \in \mathscr{I}\}$ where $(X_i = 1) \Leftrightarrow (i \in x)$, and $\{T_t \mid t \in \mathscr{T}\}$ where $(T_t = 1) \Leftrightarrow (x \subseteq t)$. Each set of boolean variables aims at representing the characteristic function of the unknown pattern. For a set of k unknown patterns [9], each pattern x_j is represented by its own set of boolean variables $\{X_{i,j} \mid i \in \mathscr{I}\}$ and $\{T_{t,j} \mid t \in \mathscr{T}\}$.

5 Related Work

Computing skylines is a derivation from the maximal vector problem in computational geometry [13], the Pareto frontier [10] and multi-objective optimization. Since its rediscovery within the database community by [3], several methods have been developed for answering skyline queries [15, 16, 20]. These methods assume that tuples are stored in efficient tree data structures. Alternative approaches have also been proposed to help the user in selecting most significant skylines. For example, [11] measures this significance by means of the number of points dominated by a skyline.

Introducing softness for skylines. [8] have proposed thick skylines to extend the concept of skyline. A thick skyline is either a skyline point p_i, or a point p_j dominated by a skyline point p_i and such that p_j is close to p_i. In this work, the idea of softness is limited to metric semi-balls of radius $\varepsilon>0$ centered at points p_i, where p_i are skylines.

Computing skypatterns is different from computing skylines. Skyline queries focus on the extraction of tuples of the dataset and assume that all the elements are in the dataset, while the skypattern mining task consists in extracting patterns which are elements of the frontier defined by the given measures. The skypattern problem is clearly harder because the search space for skypatterns is much larger than the search space for skylines: $O(2^{|\mathscr{I}|})$ instead of $O(|\mathscr{T}|)$ for skylines.

To the best of our knowledge, there are only two works dealing with skypatterns. [19] have proposed an approach taking benefit of theoretical relationships between pattern condensed representations and skypatterns and making the process feasible when the pattern condensed representation can be extracted. Nevertheless, this method can only use a crisp dominance relation. [17] deal with skypatterns from graphs but their technique only maximizes two measures (number of vertices and edge connectivity).

CP for computing the Pareto frontier. [6] has proposed an algorithm that provides the Pareto frontier in a CSP. This algorithm is based on the concept of nogoods[3] and uses spatial data structures (quadtrees) to arrange the set of nogoods. This approach only deals with non-dominated points. Moreover, it cannot be applied for mining skypatterns.

6 Case study: discovering toxicophores

A major issue in chemoinformatics is to establish relationships between chemicals and a given activity (e.g., CL50 is the lethal concentration of a substance required to kill half the members of a tested population after a specified test duration) in ecotoxicity. Chemical fragments[4] which cause toxicity are called *toxicophores* and their discovery is at the core of prediction models in (eco)toxicity [1, 18]. The aim of this present study, which is part of a larger research collaboration with the CERMN Lab, a laboratory of medicinal chemistry, is to investigate the use of softness for discovering toxicophores.

6.1 Experimental protocol

The dataset is collected from the ECB web site[5]. For each chemical, the chemists associate it with hazard statement codes (HSC) in 3 categories: H400 (very toxic, CL50 $\leq$ 1 mg/L), H401 (toxic, 1 mg/L $<$ CL50 $\leq$ 10 mg/L), and H402 (harmful, 10 mg/L $<$ CL50 $\leq$ 100 mg/L). We focus on the H400 and H402 classes. The dataset $\mathscr{T}$ consists of 567 chemicals, 372 from the H400 class and 195 from the H402 class. The chemicals are encoded using 1,450 frequent closed subgraphs previously extracted from $\mathscr{T}$[6] with a 1% relative frequency threshold.

In order to discover patterns as candidate toxicophores, we use both measures typically used in contrast mining [14] such as the growth rate since toxicophores are linked to a classification problem with respect to the HSC and measures expressing the background knowledge such as the aromaticity or rigidity because chemists consider that this information may yield promising candidate toxicophores. Our method offers a natural way to simultaneously combine in a same framework these measures coming from various origins. We briefly sketch these measures.

- Growth rate. When a pattern has a frequency which significantly increases from the H402 class to the H400 class, then it stands a potential structural alert related to the toxicity: if a chemical has, in its structure, fragments that are related to a toxic effect, then it is more likely to be toxic. Emerging patterns embody this natural idea by using the growth-rate measure (see Definition 1).

- Frequency. Real-world datasets are often noisy and patterns with low frequency may be artefacts. The minimal frequency constraint ensures that a pattern is representative enough (i.e., the higher the frequency, the better is).

- Aromaticity. Chemists know that the aromaticity is a chemical property that favors toxicity since their metabolites can lead to very reactive species which can interact with biomacromolecules in a harmful way. We compute the aromaticity of a pattern as the mean of the aromaticity of its chemical fragments.

[3] A nogood is a partial or complete assignment of the variables such that there will be no (new) solution containing it.

[4] A fragment denominates a connected part of a chemical structure containing at least one chemical bond.

[5] European Chemicals Bureau: http://echa.europa.eu/

[6] A chemical *Ch* contains an item *A* if *Ch* supports *A*, and *A* is a frequent subgraph of $\mathscr{T}$.

	Skypatterns				
		CP+SKY		MICMAC+SKY	
	# of Skypatterns	# of Candidates	CPU-Time	# of closed patterns	CPU-Time
M_1={*growth-rate*, *freq*}	**8**	613	**18m:34s**	41,887	**19m:20s**
M_2={*growth-rate*, *aromaticity*}	**5**	140	**15m:32s**	53,201	**21m:33s**
M_3={*freq*, *aromaticity*}	**2**	456	**16m:45s**	157,911	**21m:16s**
M_4={*growth-rate*, *freq*, *aromaticity*}	**21**	869	**17m:49s**	12,126	**21m:40s**

Table 2: Skypattern mining on ECB dataset.

Redundancy is reduced by using **closed skypatterns** which are an exact condensed representation of the whole set of skypatterns (see Footnote 2). We consider four sets of measures: M_1, M_2, M_3 and M_4 (see Table 2). For δ-skypatterns, we consider two values: δ=10% and δ=20%. The extracted skypatterns and soft skypatterns are made of molecular fragments. To evaluate the presence of toxicophores in their description, an expert analysis leads to the identification of well-known environmental toxicophores.

6.2 Performance analysis

This section compares our approach (noted CP+SKY) with MICMAC+SKY, which is the only other method able to mine skypatterns [19]. As our proposal, MICMAC+SKY proceeds in two steps. First, condensed representations of the whole set of patterns (i.e. closed patterns according to the considered set of measures) are extracted. Then, the sky operator is applied. Table 2 reports, for each set of measures:

– the number of skypatterns,
– for CP+SKY, the number of candidates (i.e. the number of intermediate patterns, see Section 4.2) and the associated CPU-time,
– for MICMAC+SKY, the number of closed patterns of the condensed representation and the associated CPU-time.

Table 3 reports, for each set of measures:

– the number of edge-skypatterns that are not (hard) skypatterns, the number of candidates and the required CPU-time,
– the number for δ-skypatterns that are not edge-skypatterns, the number of candidates and the required CPU-time.

CP+SKY outperforms MICMAC+SKY in terms of CPU-times (see Table 2). Moreover, the number of candidates generated by our approach remains small compared to the number of closed patterns computed by MICMAC+SKY. Thanks to dynamic constraints, our CP approach enables to drastically reduce the number of candidates. Moreover, increasing the number of measures leads to a larger number of (soft-)skypatterns, particularly for high values of δ. In fact, a pattern rarely dominates all other patterns on the whole set of measures. Nevertheless, in our experiments, the number of soft skypatterns remains reasonably small. For edge-skypatterns, there is a maximum of 144 patterns, while for δ-skypatterns, there is a maximum of $1,724$ patterns (for $\delta = 20\%$).

	Edge-Skypatterns			δ-Skypatterns					
				$\delta = 10\%$			$\delta = 20\%$		
	CP+Edge-SKY			CP+δ-SKY					
	# of Edge-skypatterns	# of Candidates	CPU-Time	# of δ-skypatterns	# of Candidates	CPU-Time	# of δ-skypatterns	# of Candidates	CPU-Time
M_1	**24**	1,746	**19m:02s**	**25**	4,204	**20m:48s**	**87**	6,253	**22m:36s**
M_2	**76**	688	**17m:51s**	**354**	1,678	**18m:14s**	**1,670**	2,816	**23m:44s**
M_3	**72**	1,726	**16m:50s**	**352**	4,070	**19m:43s**	**1,654**	6,699	**22m:25s**
M_4	**144**	3,021	**20m:27s**	**385**	6,048	**23m:36s**	**1,724**	8,986	**30m:14s**

Table 3: Soft skypattern mining on ECB dataset.

6.3 Qualitative analysis

In this section, we analyse qualitatively the (soft-)skypatterns by evaluating the presence of toxicophores in their description, according to well-known environmental toxicophores. For M_1=$\{$*growth-rate*, *freq*$\}$, soft skypatterns enable to efficiently detect well-known toxicophores emphasized by skypatterns, while for M_2=$\{$*growth-rate*, *aromaticity*$\}$ and M_4=$\{$*growth-rate*, *freq*, *aromaticity*$\}$, soft skypatterns enable to discover (new) interesting toxicophores that would not be detected by skypatterns.

- Growth rate and frequency measures (M_1). Only 8 skypatterns are found, and 3 well-known toxicophores are emphasized. Two of them are aromatic compounds, namely the chlorobenzene (p_1) and the phenol rings (p_2). The contamination of water and soil by organic aromatic chemicals is widespread as a result of industrial applications ranging from their use as pesticides, solvents to explosives and dyestuffs. Many of them may bioaccumulate in the food chain and have the potential to be harmful to living systems including humans, animals, and plants. The third one, the organophosphorus moiety (p_3) is a component occurring in numerous pesticides.

Soft skypatterns confirm the trends given by skypatterns. However, the chloro-substituted aromatic rings (e.g. p_4), and the organophosphorus moiety (e.g. p_5) are detected by the edge-skypatterns and by the δ-skypatterns. Indeed, several patterns containing these toxicophores are extracted.

- Growth rate and aromaticity measures (M_2). Figure 6 only reports the distribution of the (soft-)skypatterns for M_2. Soft skypatterns lead to the discovery of several different aromatic rings. In fact, the nature of these chemicals can vary in function of i) the presence/absence of heteroatoms (e.g. N, S), ii) the number of rings, and iii) the presence/absence of substituents.

Edge-skypatterns leads to the extraction of (i) *nitrogen aromatic compounds*: indole (p_1) and benzoimidazole (p_2), (ii) *S-containing aromatic compounds*: benzothiophene (p_3), (iii) *aromatic oxygen compounds*: benzofurane (p_4), and (iv) *polycyclic aromatic hydrocarbons*: naphthalene (p_5). δ-skypatterns complete the list of the aromatic rings which were not found during the extraction of the skypatterns, namely biphenyl (p_6).

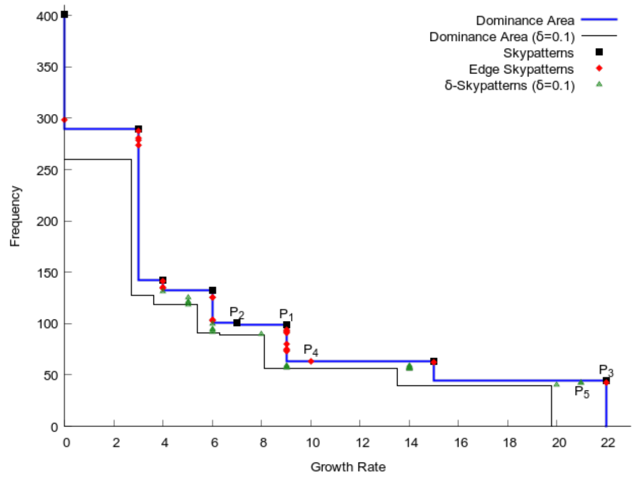


Fig. 5: Analysing the (soft-) skypatterns for M_1.

- Growth rate, frequency and aromaticity measures (M_4)**.** The most interesting results are provided using M_4 (see Figure 7). 21 skypatterns are mined, and several well-known toxicophores are emphasized: the phenol ring (see e_4), the chloro-substituted aromatic ring (see e_3), the alkyl-substituted benzene (see e_2), and the organophosphorus moiety (see P_1). Besides, information dealing with nitrogen aromatic compounds are also extracted (see e_1).

Soft skypatterns enable to mine several exotic aromatic rings (previously discussed), namely nitrogen and S-containing aromatic compounds, polycyclic aromatic hydrocarbons.

Moreover, edge-skypatterns enable to detect more precisely the chloro-substituted aromatic ring and the organophosphorus moiety which are located near P_1. For $\delta \in \{10\%, 20\%\}$, mining the δ-skypatterns leads to the extraction of new several interesting patterns, particularly substituted nitrogen aromatic rings and substituted anilines.

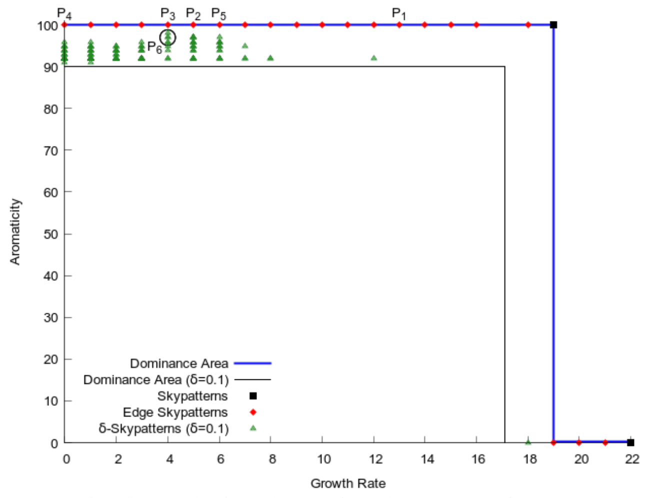


Fig. 6: Analysing the (soft-) skypatterns for M_2.

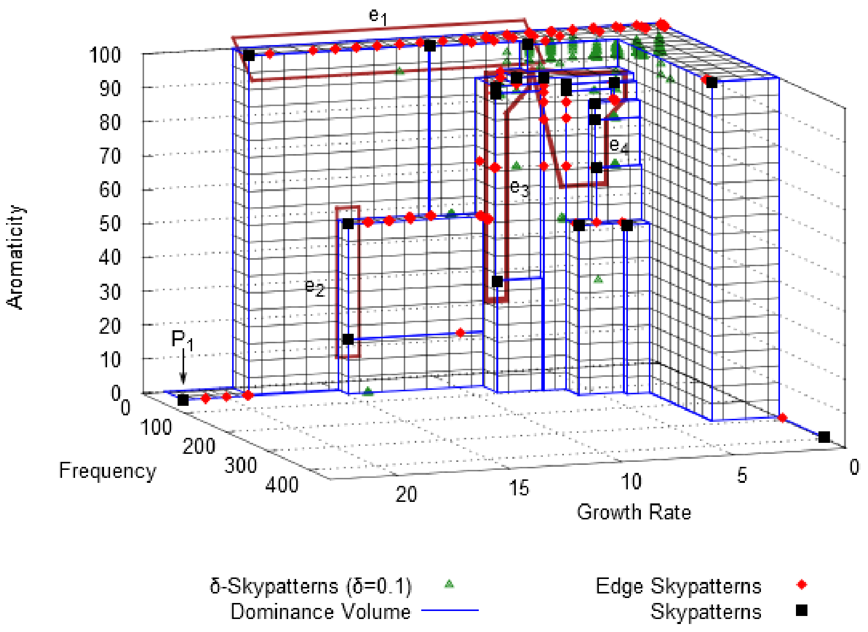


Fig. 7: Analysing the (soft-) skypatterns for M_4.

7 Conclusion

We have introduced the notion of soft skypattern and proposed a flexible and efficient approach to mine skypatterns as well as soft ones thanks to Dynamic CSP. Finally, the relevance and the effectiveness of our approach has been highlighted through a case study in chemoinformatics for discovering toxicophores.

In the future, we would like to study the introduction of softness on other tasks such as clustering, study the contribution of soft skypatterns for recommendation and extend our approach to skycubes. Another direction is to improve the solving stage by designing a one-step method: each time a new solution s_i is found, all candidates that are dominated by s_i can be removed (see Section 4.2). Another idea is to hybridize our CP approach with local search methods to improve the efficiency of the method.

Acknowledgments. We would like to thank Arnaud Soulet (University F. Rabelais of Tours, France), for providing the `MICMAC+SKY` program and his highly valuable comments. This work is partly supported by the ANR (French Research National Agency) funded project FiCoLoFo ANR-10-BLA-0214.

References

1. J. Bajorath and J. Auer. Emerging chemical patterns : A new methodology for molecular classification and compound selection. *J. of Chemical Information and Modeling*, 46:2502–2514, 2006.
2. S. Bistarelli and F. Bonchi. Soft constraint based pattern mining. *Data Knowl. Eng.*, 62(1):118–137, 2007.
3. S. Börzönyi, D. Kossmann, and K. Stocker. The skyline operator. In *17th Int. Conf. on Data Engineering*, pages 421–430. Springer, 2001.
4. L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In *KDD'08*, pages 204–212. ACM, 2008.
5. L. De Raedt and A. Zimmermann. Constraint-based pattern set mining. In *7th SIAM International Conference on Data Mining*. SIAM, 2007.
6. M. Gavanelli. An algorithm for multi-criteria optimization in csps. In Frank van Harmelen, editor, *ECAI*, pages 136–140. IOS Press, 2002.
7. T. Guns, S. Nijssen, and L. De Raedt. Itemset mining: A constraint programming perspective. *Artif. Intell.*, 175(12-13):1951–1983, 2011.
8. W. Jin, J. Han, and M. Ester. Mining thick skylines over large databases. In *PKDD'04*, pages 255–266, 2004.
9. M. Khiari, P. Boizumault, and B. Crémilleux. Constraint programming for mining n-ary patterns. In *CP'2010*, volume 6308 of *LNCS*, pages 552–567, 2010.
10. H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of ACM*, 22(4):469–476, October 1975.
11. X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE 2007*, pages 86–95, 2007.
12. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and K. Discovery*, 1(3):241–258, 1997.
13. J. Matousek. Computing dominances in E^n. *Inf. Process. Lett.*, 38(5):277–278, 1991.
14. P. Kralj Novak, N. Lavrac, and G. I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research*, 10:377–403, 2009.
15. D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.
16. D. Papadias, M. Yiu, N. Mamoulis, and Y. Tao. Nearest neighbor queries in network databases. In *Encyclopedia of GIS*, pages 772–776. 2008.
17. A. N. Papadopoulos, A. Lyritsis, and Y. Manolopoulos. Skygraph: an algorithm for important subgraph discovery in relational graphs. *Data Min. Knowl. Discov.*, 17(1):57–76, 2008.
18. G. Poezevara, B. Cuissart, and B. Crémilleux. Extracting and summarizing the frequent emerging graph patterns from a dataset of graphs. *J. Intell. Inf. Syst.*, 37(3):333–353, 2011.
19. A. Soulet, C. Raïssi, M. Plantevit, and B. Crémilleux. Mining dominant patterns in the sky. In *ICDM*, pages 655–664, 2011.
20. Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.
21. W. Ugarte, P. Boizumault, S. Loudni, and B. Crémilleux. Soft threshold constraints for pattern mining. In *Discovery Science*, pages 313–327, 2012.
22. G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10(3):253–281, 2005.

Query Rewriting for Rule Mining in Databases

Brice Chardin[1,2], Emmanuel Coquery[1,3], Benjamin Gouriou[4], Marie Pailloux[4], and Jean-Marc Petit[1,2]

[1] Université de Lyon, CNRS
LIRIS, UMR 5205, France
[2] INSA Lyon, France
[3] Université Claude Bernard Lyon 1
[4] Université Blaise Pascal, CNRS
LIMOS, UMR 6158, Clermont-Ferrand, France

Abstract. Promoting declarative approaches in data mining is a long standing theme. This paper goes into this direction by proposing a well-founded logical query language, $Safe\mathcal{RL}$, allowing the expression of a wide variety of "rules" to be discovered against the data. $Safe\mathcal{RL}$ extends and generalizes functional dependencies in databases to new and unexpected rules easily expressed with a SQL-like language. In this setting, every rule mining problem turns out to be seen as a query processing problem. We provide a query rewriting technique and a constructive proof of the main query equivalence theorem, leading to an efficient query processing technique. Based on a concrete SQL-like grammar for $Safe\mathcal{RL}$, we show how a tight integration can be performed on top of any DBMS. The approach has been implemented and experimented on sensor network data. This contribution is an attempt to bridge the gap between pattern mining and databases and facilitates the use of data mining techniques by SQL-aware analysts.

Keywords: Pattern mining, databases, query languages, query processing

1 Introduction

The relational database management systems (DBMS) market is already huge and continues to grow since it is expected to nearly double by 2016 [1]. As a trivial consequence for the data mining community, it makes sense – more than ever – to query the data where they are by using state of the art database technologies.
While a lot of techniques have been proposed over the last 20 years for pattern mining, only a few of them are tightly coupled with a DBMS. Most of the time, some pre-processing has to be performed before the use of pattern mining techniques and the data have to be formatted and exchanged between different systems, turning round-trip engineering into a nightmare.

In this paper, we provide a logical view for a certain class of pattern mining problems. More precisely, we propose a well-founded logical query language,

EMP	Empno	Lastname	Work dept	Job	Educ level	Sex	Sal	Bonus	Comm	Mgrno
	10	SPEN	C01	FINANCE	18	F	52750	500	4220	20
	20	THOMP	B01	MANAGER	18	M	41250	800	3300	
	30	KWAN	C01	FINANCE	20	F	38250	500	3060	10
	50	GEYER	B01	MANAGER	16	M	40175	800	3214	20
	60	STERN	D21	SALE	14	M	32250	500	2580	30
	70	PULASKI	D21	SALE	16	F	36170	700	2893	100
	90	HENDER	D21	SALE	17	F	29750	500	2380	10

DEPT	Deptno	Deptname	Mgrno	Admrdept	Loc
	A00	SPIFFY CS DIV.	-	A00	-
	B01	PLANNING	20	A00	501
	C01	INF. CENTER	30	A00	403
	D01	DEV. CENTER	60	A00	-
	D11	MANUFACTURING SYSTEMS	-	D01	-
	D21	ADMIN. SYSTEMS	70	D01	501

Fig. 1. Running example

$SafeRL$, based on tuple relational calculus (TRC), allowing the expression of a wide variety of "rules" to be discovered against the data. $SafeRL$ extends and generalizes functional dependencies (FDs) in databases to new and unexpected rules easily expressed with a practical SQL-like language derived from $SafeRL$, called $\mathcal{RQL}$. To start with, let us consider the running example given in Figure 1 with two relations *Emp* and *Dept*. *Educlevel* represents the number of years of formal education, *Sal* the yearly salary, *Bonus* the yearly bonus and *Comm* the yearly commission. This example will be used throughout the paper.

Intuitively, a $\mathcal{RQL}$ query is defined by the FINDRULES clause and generates rules of the form $X \rightarrow Y$ with X and Y disjoint attribute sets taken from the OVER clause. The SCOPE clause defines tuple-variables over some relations obtained by classical SQL queries and the CONDITION clause defines the predicate to be evaluated on each attribute occurring in $X \cup Y$.

Example 1. To make things concrete, we give some examples of $\mathcal{RQL}$ queries.

Q_1:
```
FINDRULES
OVER Empno,Lastname,Workdept,Job,Sex,Bonus
SCOPE t1,t2 Emp
CONDITION ON A IS t1.A = t2.A;
```
Q'_1:
```
FINDRULES
OVER Empno,Lastname,Workdept,Job,Sex,Bonus
SCOPE t1,t2 (SELECT * FROM Emp WHERE Educlevel>16)
CONDITION ON A IS t1.A = t2.A;
```
Q''_1:
```
FINDRULES
OVER Educlevel,Sal,Bonus,Comm
SCOPE t1,t2 Emp
CONDITION ON A IS 2*ABS(t1.A-t2.A)/(t1.A+t2.A)<0.1;
```

Q_1 discovers FDs from *Emp* over a subset of attributes. Recall that a FD $X \to Y$ holds in a relation r if for all tuples $t1, t2 \in r$, and for all $A \in X$ such that $t1[A] = t2[A]$ then for all $A \in Y$, $t1[A] = t2[A]$. As shown in Q_1, $\mathcal{RQL}$ can express the discovery of FDs with a natural syntax. For example, $Empno \to Lastname$, $Workdept \to Job$ hold in *Emp*.

We can easily restrict FDs to some subset of tuples as shown with Q_1' which discovers rules comparable to conditional functional dependencies [2] by considering only employees with a level of qualification above 16. For instance, $Sex \to Bonus$ holds, meaning that above a certain level of qualification (16), the gender determines the bonus. This rule was not elicited by Q_1 because of employees 60 and 70.

Q_1'' is an approximation of FD for numeric values, where strict equality is discarded to take into account variations under 10%. For instance, salaries 41250 and 38250 are considered close (7.5% difference), but not salaries 41250 and 36170 (13.1% difference). $Sal \to Comm$ then holds, meaning that employees earning similar salaries receive similar commissions.

Nevertheless, $\mathcal{RQL}$ can do much more and is not restricted to FD at all.

Example 2. null values in *Dept*.

Q_2:
```
FINDRULES
OVER Deptname,Mgrno,Admrdept,Loc
SCOPE t1 Dept
CONDITION ON A IS t1.A IS NULL;
```

This query discovers rules between null values of the relation *Dept*. In dirty databases, null values can be a nightmare and knowing relationships between attributes with respect to null values could definitely be useful. For instance, $Mgrno \to Loc$ holds in *Dept*.

In this setting, every rule mining problem can be easily specified and turns out to be seen as a query processing problem. We provide a query rewriting technique and a constructive proof of the main query equivalence theorem, leading to an efficient query processing technique. Based on a concrete SQL-like grammar for $Safe\mathcal{RL}$, we have shown how a tight integration can be performed on top of any DBMS. The approach has been implemented and experimented on sensor network data, showing that our approach allows the discovery of meaningful rules and scales well. This contribution is an attempt to bridge the gap between pattern mining and databases to facilitate the use of data mining techniques by SQL-aware analysts. The ultimate goal of this work is to integrate pattern mining techniques into core DBMS technologies.

Related works Defining specific languages for pattern mining is a long standing goal and poses several challenges to first specify the data of interest and second, to pose pattern mining queries against these data. A survey of data mining query languages has been done in [3]. Recently, constraint programming appears to be convenient and useful since it allows addressing both issues with a unique declarative language [4, 5].

Nevertheless, we argue that pattern mining languages should benefit from direct extensions of SQL languages as done in [6], since data are most of the time stored in DBMSs. Practical approaches, as close as possible of DBMSs, have been proposed for example in [7–9] to interact more directly with DBMSs query engines. The $SafeRL$ language proposed in this paper goes into this direction by providing a formal semantic based on the TRC language, its practical counterpart $\mathcal{RQL}$ turning out to be easily implemented on top of every DBMS. FDs, association rules with 100% confidence, ad-hoc language proposed in [10] are special cases of our $Safe\mathcal{RL}$ language but none of them has a logical query language foundation.

With respect to [11], we consider a database instead of a single relation in $Safe\mathcal{RL}$, we provide a SQL-like syntax $\mathcal{RQL}$ and we focus on query processing techniques that can be reused to efficiently execute $\mathcal{RQL}$ queries. Theoretical results on decidability problems on variant of $Safe\mathcal{RL}$ languages are given in [11]. From a technical point of view, this paper is a generalization of the approach taking into account FD only [12] to compute agree sets from database relations using SQL. FDs, CFDs in databases and implications in formal concept analysis have been studied in e.g. [12–15].

Paper organization Section 2 introduces some notations and recalls important notions on relational calculus and closure systems. Section 3 presents the syntax and semantics of the $Safe\mathcal{RL}$ language, while section 4 presents some results used for computing the answer to $Safe\mathcal{RL}$ queries. Section 5 presents experimental results and section 6 concludes. Due to space constraints, proofs are omitted.

2 Preliminaries

This section introduces main definitions and notations used throughout the paper for the relational model, safe TRC, rules and closure systems.

2.1 Relational model

We use the named perspective of the relational model in which tuples are functions.

Fix a finite universe $\mathbb{U}$ of *attributes* (denoted by $\mathtt{A}, \mathtt{B}, \ldots$), a countably infinite domain $\mathbb{D}$ of *constants* (denoted by $\mathtt{c}, \mathtt{c}', \ldots$) and a finite set $\mathbb{R}$ of *relation symbols* (denoted by $\mathtt{R}, \mathtt{S}, \ldots$). $\mathbb{U}, \mathbb{D}, \mathbb{R}$ are pairwise disjoints. Each relation symbol $\mathtt{R}$ has a schema, a subset of $\mathbb{U}$, denoted by the symbol itself, i.e. $\mathtt{R} \subseteq \mathbb{U}$. Conveniently, we will sometimes omit to refer to the relation symbol when dealing with a subset of attributes, i.e. a schema. A *tuple* $\mathtt{t}$ over $\mathtt{R}$ is a total function $\mathtt{t} : \mathtt{R} \rightarrow \mathbb{D}$. A *relation* $\mathtt{r}$ over $\mathtt{R}$ is a finite set of tuples over $\mathtt{R}$. A *database schema* $\mathbf{R}$ is a set of relation symbols, e.g. $\mathbf{R} = \{\mathtt{R_1}, \ldots, \mathtt{R_n}\}$. A *database instance* (or simply a database) is a function d from $\mathbf{R}$ to the set of possible relations such that $d(\mathtt{R_i}) = \mathtt{r_i}, \mathtt{r_i}$ a relation over $\mathtt{R_i}$, for $i = 1..n$.

2.2 Variables and assignments

$SafeR\mathcal{L}$ has different formal variables for attributes, tuples and schemata: a set $\mathbb{A}$ of attribute-variables $(A, B, \ldots)$, a set $\mathbb{T}$ of tuple-variables $(s, t, \ldots)$ and a set $\mathbb{S}$ of schema-variables $(X, Y, \ldots)$. $\mathbb{A}, \mathbb{T}, \mathbb{S}, \mathbb{U}, \mathbb{D}, \mathbb{R}$ are pairwise disjoints.

An attribute-assignment ρ (resp. a schema-assignment Σ) is a function that maps an attribute-variable A (resp. a schema-variable X) to an attribute $\rho(A) \in \mathbb{U}$ (resp. a subset of attributes $\Sigma(X) \subseteq \mathbb{U}$). A tuple-assignment σ is also a function from a tuple-variable t to a tuple $\mathtt{t}$ defined over some schema. Conveniently, a tuple-variable t can be explicitly defined over some schema $\mathtt{X}$, noted by $t : \mathtt{X}$ and we will use the notation $sch(t) = \mathtt{X}$.

For an attribute-assignment ρ (as well as for tuple-assignments and schema-assignments) we denote by $\rho_{A \mapsto \mathtt{A}}$ the assignment defined by:

$$\rho_{A \mapsto \mathtt{A}}(B) = \begin{cases} \mathtt{A} & \text{if } B = A \\ \rho(B) & \text{if } B \neq A \end{cases}$$

2.3 Safe TRC

For the sake of clarity, we recall here the syntax and semantics of the TRC in its simplest form. TRC formulas noted $\psi, \psi_1, \psi_2, \ldots$ are defined inductively as usual, where $\mathtt{A}, \mathtt{B} \in \mathbb{U}$, $\mathtt{X} \subseteq \mathbb{U}$, $\mathtt{c} \in \mathbb{D}$, $\mathtt{R} \in \mathbb{R}$, $t, t_1, t_2 \in \mathbb{T}$:

$$\mathtt{R}(t) \mid t_1.\mathtt{A} = t_2.\mathtt{B} \mid t.\mathtt{A} = \mathtt{c} \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \exists t : \mathtt{X}\ (\psi)$$

Given a database d over $\mathbf{R}$ and a tuple assignment σ, the satisfaction of a TRC formula ψ is inductively defined as follows:

- $\langle d, \sigma \rangle \models \mathtt{R}(t)$ if $\sigma(t) \in d(\mathtt{R}), \mathtt{R} \in \mathbf{R}$
- $\langle d, \sigma \rangle \models t_1.\mathtt{A} = t_2.\mathtt{B}$ if $\sigma(t_1)(\mathtt{A}) = \sigma(t_2)(\mathtt{B})$
- $\langle d, \sigma \rangle \models t.\mathtt{A} = \mathtt{c}$ if $\sigma(t)(\mathtt{A}) = \mathtt{c}$
- $\langle d, \sigma \rangle \models \neg\psi$ if $\langle d, \sigma \rangle \not\models \psi$
- $\langle d, \sigma \rangle \models \psi_1 \wedge \psi_2$ if $\langle d, \sigma \rangle \models \psi_1$ and $\langle d, \sigma \rangle \models \psi_2$
- $\langle d, \sigma \rangle \models \exists t : \mathtt{X}\ (\psi)$ if there exists a tuple $\mathtt{t}$ over $\mathtt{X}$ such that $\langle d, \sigma_{t \mapsto \mathtt{t}} \rangle \models \psi$

A TRC query is an expression of the form

$$q = \{t : \mathtt{X} \mid \psi\}$$

where ψ is a TRC formula with exactly one free variable t. The set of answers $ans(q, d)$ of q w.r.t. a database d is

$$ans(q, d) = \{\sigma(t) \mid \langle d, \sigma \rangle \models \psi\}$$

In the sequel, we will consider a restriction of the TRC, equivalent to the relational algebra in order to be compatible with SQL, known as *safe TRC* [16].

2.4 Rules and closure systems

Rules or implications, closure systems and closure operators have been widely studied in many branches of applied mathematics and computer sciences, with many applications in databases with functional dependencies [17] and in formal concept analysis with implications [18]. The interested reader should refer to [19] for a comprehensive survey. We summarize the main results that are useful for the rest of the paper.

Let $U \subseteq \mathbb{U}$. A closure system C on U is such that $U \in C$ and for all $X, Y \in C$, $X \cap Y \in C$ [18]. Let F be a set of rules on U. A closure system can be defined for F, noted $CL(F) = \{X \subseteq U | X = X_F^+\}$ where X_F^+ is the closure of X with respect to F. Let $IRR(F)$ be the set of meet-irreducible elements of $CL(F)$. The notion of *base* of a closure system is defined as follows:

Definition 1. *Let $CL(F)$ be a closure system. A base $\mathcal{B}$ of $CL(F)$ is such that $IRR(F) \subseteq \mathcal{B} \subseteq CL(F)$*

A base is called a context in FCA terminology [18]. From a functional dependency inference point of view, we quote the following approach [20, 12] to discover the so-called canonical basis from a relation r:

1. Compute a base of the closure system associated to FDs from r (known as *agree sets*),
2. From the base, compute the unique canonical cover for exact FDs and Gottlob and Libkin cover for approximate FDs [21, 20, 22].

The rest of this paper proposes a generalization of this approach. Indeed, each $Safe\mathcal{RL}$ query defines a closure system and therefore, in order to reuse previous results, the problem turns out to be on the computation of a base with respect to a query from the database [12]. Due to space constraints, we will focus on this first stage, the second stage will be omitted.

3 A Query Language for Rule Mining

In the introduction, we have illustrated $\mathcal{RQL}$ – a SQL-like friendly language – through examples. This section formally defines the syntax and semantics of $Safe\mathcal{RL}$ from which $\mathcal{RQL}$ is derived. We have introduced safe TRC for expressing SQL like queries. Before defining $Safe\mathcal{RL}$, it remains to precisely define (cf. previous examples of $\mathcal{RQL}$ queries) the `CONDITION` clause, through the notion of *mining formulas*:

```
CONDITION ON A IS delta_cond(A, t1, ..., tn);
```

3.1 Mining Formulas

Mining formulas, denoted by $\delta, \delta_1, \delta_2, \ldots$, are defined over tuple-variables $\mathbb{T}$, attribute-variables $\mathbb{A}$ and constants $\mathbb{D}$ only. Their syntax and their semantics are defined as follows.

Definition 2. *Let $t, t_1, t_2 \in \mathbb{T}$, $A, B \in \mathbb{A}$ and $\mathtt{c} \in \mathbb{D}$. A mining formula is of the form: $t_1.A = t_2.B \mid t.A = \mathtt{c} \mid \neg\delta \mid \delta_1 \wedge \delta_2$*

The satisfaction *of a mining formula δ w.r.t. a tuple-assignment σ and an attribute-assignment ρ, denoted by $\langle\sigma, \rho\rangle \models \delta$, is inductively defined as follows:*

- $\langle\sigma, \rho\rangle \models t_1.A = t_2.B$ ***iff*** $\sigma(t_1)(\rho(A)) = \sigma(t_2)(\rho(B))$
- $\langle\sigma, \rho\rangle \models t.A = \mathtt{c}$ ***iff*** $\sigma(t)(\rho(A)) = \mathtt{c}$
- $\langle\sigma, \rho\rangle \models \neg\delta$ ***iff*** $\langle\sigma, \rho\rangle \not\models \delta$
- $\langle\sigma, \rho\rangle \models \delta_1 \wedge \delta_2$ ***iff*** $\langle\sigma, \rho\rangle \models \delta_1$ *and* $\langle\sigma, \rho\rangle \models \delta_2$

Such formulas are very simple and restrictive: given one attribute and several tuples, it allows to tell whether or not a mining formula holds.

3.2 $\boldsymbol{Safe\mathcal{RL}}$ queries

The $Safe\mathcal{RL}$ query language can now be defined. A $Safe\mathcal{RL}$ query over a database schema $\mathbf{R}$ is an expression of the form:

$$Q = \{X \to Y \mid \forall t_1 \ldots \forall t_n(\psi(t_1, \ldots, t_n) \wedge (\forall A \in X(\delta(A, t_1, \ldots, t_n)) \to \forall A \in Y(\delta(A, t_1, \ldots, t_n))))\}, \text{ where:}$$

- X, Y are schema-variables
- ψ is a TRC-formula over $\mathbf{R}$ with n free tuple-variables $t_1, \ldots, t_n$
- δ is a mining formula with $t_1, \ldots, t_n$ free tuple-variables and A a free attribute-variable

When clear from context, a $Safe\mathcal{RL}$ query Q may also be simply denoted by $Q = \langle\psi(t_1, \ldots, t_n), \delta(A, t_1, \ldots, t_n)\rangle$ or even $Q = \langle\psi, \delta\rangle$.

Example 3. The mining of FDs over `EMP` is expressed as the query $Q = \langle \mathtt{EMP}(t_1) \wedge \mathtt{EMP}(t_2), (t_1.A = t_2.A)\rangle$.

The attributes of ψ are equal to $\bigcup_{i=1}^{n} sch(t_i)$ whereas the schema of Q, denoted by $sch(Q)$, is defined by: $sch(Q) = \bigcap_{i=1}^{n} sch(t_i)$: only common attributes of tuple-variables are meaningful to discover rules.

To specify the result of the evaluation of a $Safe\mathcal{RL}$ query against a database, we define the notion of satisfaction.

A $Safe\mathcal{RL}$ query $\langle\psi, \delta\rangle$ is satisfied in a database d and w.r.t. a schema-assignment Σ, denoted by $\langle d, \Sigma\rangle \models \langle\psi, \delta\rangle$ if the following holds:

$$\text{For all tuple-assignment } \sigma \text{ such that } \langle d, \sigma\rangle \models \psi: \quad (1)$$
$$\text{if for all } \mathtt{A} \in \Sigma(X),\ \langle\sigma, \rho_{A\mapsto\mathtt{A}}\rangle \models \delta \quad (2)$$
$$\text{then for all } \mathtt{A} \in \Sigma(Y),\ \langle\sigma, \rho_{A\mapsto\mathtt{A}}\rangle \models \delta \quad (3)$$

Intuitively, this definition generalizes the definition of FD satisfaction in a relation: instead of only 2 tuples, we may have n tuples from many relations and verifying a certain condition (1); and instead of the condition "for all $A \in R, t_1[A] = t_2[A]$", we may have any mining formula (2), (3).

The answer of a $Safe\mathcal{RL}$ query $Q = \langle\psi, \delta\rangle$ in a database d over $\mathbf{R}$, denoted by $ans(Q, d)$, is defined as: $ans(Q, d) = \{\Sigma(X) \to \Sigma(Y) \mid \langle d, \Sigma\rangle \models \langle\psi, \delta\rangle, \Sigma(X) \cup \Sigma(Y) \subseteq sch(Q)\}$.

3.3 $\mathcal{RQL}$: A practical language for $SafeR\mathcal{L}$

$\mathcal{RQL}$ is a practical SQL-like declarative language to express $Safe\mathcal{RL}$ queries. Let us consider a $Safe\mathcal{RL}$ query $Q = \langle \psi(t_1, \ldots, t_n), \delta(A, t_1, \ldots, t_n) \rangle$ and its associated $\mathcal{RQL}$ query:

```
FINDRULES
OVER A1, ..., An
SCOPE t1 (SQL1), ..., tn (SQLn)
WHERE condition(t1, , ..., tn)
CONDITION ON A IS delta_cond(A, t1, ..., tn);
```

The clause FINDRULES identifies $\mathcal{RQL}$ queries. The clause SCOPE specifies every tuple to be used to discover the rules and corresponds to the tuple-variables of ψ, a safe TRC formula. The clause CONDITION gives the condition to be observed against the data, the so-called mining formula δ. The clause OVER allows restrictions of rules to a particular set of attributes, typically the attributes of $sch(Q)$. The WHERE clause is defined as usual. We have already given many examples of $\mathcal{RQL}$, other details are omitted. Note that $\mathcal{RQL}$ allows much more flexibility than $Safe\mathcal{RL}$ since syntactic sugars available in SQL can be used for free, as in query Q_1'' of Example 1.

4 Theoretical results

In [11], a slightly different language for rule mining has been proposed. One of the main results was to point out that every query is "Armstrong-compliant", meaning basically that Armstrong axioms are sound and that each query defines a closure system. The same result holds for $Safe\mathcal{RL}$ queries.

Given a database d and a $Safe\mathcal{RL}$ query Q, the basic idea is to compute a base of the closure system associated with Q from d. Let us start by introducing the closure system associated with Q.

An appendix provides the proofs of all propositions, lemmas and theorems.

4.1 Closure system and bases for $Safe\mathcal{RL}$ queries

Given a query Q against a database d, the definitions of a base and a closure system given in Section 2.4 are extended to $ans(Q, d)$.

We say that $\mathtt{Z} \subseteq \mathbb{U}$ satisfies $ans(Q, d)$ if for all $\mathtt{X} \rightarrow \mathtt{Y} \in ans(Q, d)$, $\mathtt{X} \not\subseteq \mathtt{Z}$ or $\mathtt{Y} \subseteq \mathtt{Z}$. The *closure system* of Q in d, denoted by $CL_Q(d)$, is defined by: $CL_Q(d) = \{\mathtt{Z} \subseteq sch(Q) \mid \mathtt{Z} \text{ satisfies } ans(Q, d)\}$.

In our setting, the definition of the base is:

Definition 3. *Let $Q = \langle \psi, \delta \rangle$ be a $Safe\mathcal{RL}$ query over $\mathbf{R}$ and d a database over $\mathbf{R}$. The* base *of Q in d, denoted by $B_Q(d)$, is defined by:*

$$B_Q(d) = \bigcup_{\substack{\sigma \ s.t. \\ \langle d, \sigma \rangle \models \psi}} \Big\{ \{\mathtt{A} \in sch(Q) \mid \langle \sigma, \rho_{A \mapsto \mathtt{A}} \rangle \models \delta \} \Big\}$$

That is, $B_Q(d)$ is the set of all $\mathtt{Z} \subseteq sch(Q)$ for which there exists σ such that $\langle d, \sigma\rangle \models \psi$ and $\langle \sigma, \rho_{A\mapsto \mathtt{A}}\rangle \models \delta$ for all $\mathtt{A} \in \mathtt{Z}$. Note that since A is the only free attribute variable in δ, using $\rho_{A\mapsto \mathtt{A}}$ fully determines the attribute assignment in the evaluation of δ.

Proposition 1. *$B_Q(d)$ is a base of the closure system $CL_Q(d)$.*

4.2 Computing the base of a $\mathcal{S}afe\mathcal{RL}$ query using query rewriting

The naive approach consists in executing n SQL queries against the database, to cache all intermediary results, to keep only the right combination of tuples with respect to the WHERE clause and then to compute the base of the closure system. We can do much better: the basic idea is to decompose the query in order to push as much as possible the processing into the SQL query engine.

For every $\mathcal{RQL}$ query $Q = \langle \psi, \delta\rangle$ involving n tuple-variables, there exists another query $Q' = \langle \psi', \delta'\rangle$ with a *unique tuple-variable* . The practical consequence of this remark is that the computation of the base can be done in a unique SQL query, i.e. the base of $\langle \psi', \delta'\rangle$ can be delegated to the SQL query engine which was not the case in the former formulation. By means of a rewriting function rw, we transform $Q = \langle \psi, \delta\rangle$ into $Q' = \langle \psi', \delta'\rangle$. The idea of rw is that the unique tuple-variable t appearing in Q' takes its values in the schema built from the disjoint union of $sch(t_i), i = 1..n$ and is essentially the concatenation of the initial t_i's.

Let $\mathtt{R}$ be the new schema built from the disjoint union of the t_i's, i.e. $\mathtt{R} = \bigcup_{i\in 1..n}\{\langle \mathtt{A}, i\rangle \mid \mathtt{A} \in sch(t_i)\}$.

The function rw is defined on mining formulae inductively, with t a fresh tuple-variable. Each fresh attribute-variable A_i replaces A in δ', noted $\mathsf{rw}(\delta)$ in the sequel, for the corresponding t_i part of t:

$$\mathsf{rw}(\delta) = \begin{cases} \neg\mathsf{rw}(\delta') & \text{if } \delta \text{ is } \neg\delta' \\ \mathsf{rw}(\delta_1) \wedge \mathsf{rw}(\delta_2) & \text{if } \delta \text{ is } \delta_1 \wedge \delta_2 \\ t.A_i = t.A_j & \text{if } \delta \text{ is } (t_i.A = t_j.A) \\ t.A_i = \mathsf{c} & \text{if } \delta \text{ is } (t_i.A = \mathsf{c}) \end{cases}$$

We also overload rw to transform tuple-assignment σ and attribute-assignment ρ into respective assignments. $\mathsf{rw}(\sigma) = \sigma'$ is defined by $\sigma'(t) = \mathtt{t}$ and $\mathtt{t}(\langle \mathtt{A}, i\rangle) = \sigma(t_i)(\mathtt{A})$. $\mathsf{rw}(\rho) = \rho'$ is defined by $\rho'(A_i) = \langle \rho(A), i\rangle$.

Given a mining formula and the previous rewriting $\mathsf{rw}()$, we have the following lemma.

Lemma 1. *$\langle \sigma, \rho\rangle \models \delta$ iff $\langle \mathsf{rw}(\sigma), \mathsf{rw}(\rho)\rangle \models \mathsf{rw}(\delta)$*

Finally, it remains to rewrite TRC formula ψ with $\mathsf{rw}(\psi)$ by forcing t to have each t_i as one of its components:

$$\mathsf{rw}(\psi) = \exists t_1 : sch(t_1), \ldots, t_n : sch(t_n)(\psi \wedge \bigwedge_{\mathtt{A}\in sch(Q)} \bigwedge_{i=1}^{n} t.\mathtt{A}_i = t_i.\mathtt{A})$$

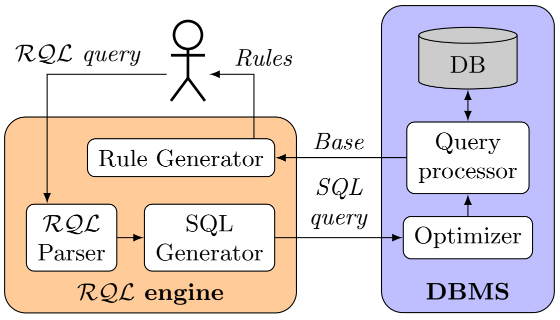


Fig. 2. Query processing overview

Given a safe TRC formula and the previous rewriting $\mathsf{rw}()$, we have the following result.

Lemma 2. $\langle d, \sigma\rangle \models \psi$ *iff* $\langle d, \mathsf{rw}(\sigma)\rangle \models \mathsf{rw}(\psi)$

Therefore $B_Q(d)$ is computable by running only one SQL query, corresponding to the safe TRC query $\{t \mid \mathsf{rw}(\psi)\}$, except for the SELECT part that consists in evaluating, for each $\mathtt{A} \in sch(Q)$, the satisfaction of $\mathsf{rw}(\delta)$.

Theorem 1.

$$B_Q(d) = \bigcup_{\substack{\mathsf{rw}(\sigma) \text{ s.t.} \\ \langle d,\mathsf{rw}(\sigma)\rangle \models \mathsf{rw}(\psi)}} \Big\{ \{\mathtt{A} \in sch(Q) \mid \exists \rho : \rho(A) = \mathtt{A} \wedge \langle \mathsf{rw}(\sigma), \mathsf{rw}(\rho)\rangle \models \mathsf{rw}(\delta)\} \Big\}$$

The previous theorem allows pushing query processing as much as possible into the DBMS to compute the base of the closure system associated to every $Safe\mathcal{RL}$ query.

5 Implementation and experiments

Given a $\mathcal{RQL}$ query Q, the query processing engine consists of a Java/JavaCC application to:

1. Compute the base of the closure system of Q using the generated SQL query provided by Theorem 1.
2. Compute the canonical cover for exact rules and a Gottlob and Libkin cover for approximate rules [21]. Details are out of the scope of this paper, note just that we have reused the code of T. Uno [23] for the most expensive part of the rule generation process.

Figure 2 gives an overview of $\mathcal{RQL}$ query processing.

samples	
id	DECIMAL(20,0)
timestamp	TIMESTAMP
type	DECIMAL(3,0)
value	DECIMAL(10,0)

descriptions	
id	DECIMAL(20,0)
type	VARCHAR(12)
location	VARCHAR(18)
description	VARCHAR(78)

Fig. 3. PlaceLab database schema

time	bathroom_light	kitchen_humidity_0	...	bedroom_temperature_5
2006-08-22 00:00:00	0.4971428	4344	...	21.43
2006-08-22 00:01:00	0.6685879	4344	...	21.43
2006-08-22 00:02:00	0.4985673	4344	...	21.465
		...		
2006-09-18 23:58:00	1567.7822	5324	...	22.53
2006-09-18 23:59:00	1563.5891	5276	...	22.50

Fig. 4. Sensors data

5.1 Sensor Data

We experimented our $\mathcal{RQL}$ processing engine using the PlaceLab dataset provided by the MIT House_n Consortium and Microsoft Research [24].

The PlaceLab is a 93 m^2 apartment instrumented with several hundred sensors. During the experiment, interior conditions (temperature, humidity, light, pressure, current, gas and water flow) were captured by 106 sensors, along with 92 reed switches installed on doors, drawers and windows to detect open-closed events. 277 wireless motion sensors were placed on nearly all objects that might be manipulated. Two participants used the PlaceLab as a temporary home for 10 weeks.

The available data is a subset of about a month from the original 2.5 months experiment, from August 22, 2006 to September 18. Raw data is extracted from binary files, where each reading contains a sensor id, a timestamp and a value (the measurement). Sensors meta-data include, for each sensor id, type, location and a short textual description, along with other meta-data of little interest for our experiments, such as installation date, etc. This data is stored in an Oracle database whose schema is depicted in figure 3.

For data mining queries, we focused on variations of the physical properties of the environment, such as temperature, light, etc., which amount to more than 100 million tuples. A view, easily expressed with SQL, has been created to synchronize and resample every sensor with a common sampling interval (one minute). This view, illustrated in figure 4, has 165 attributes and 32543 tuples. Except for the time, each attribute is associated either with one of the 106 physical properties sensors or one of the 58 selected switches.

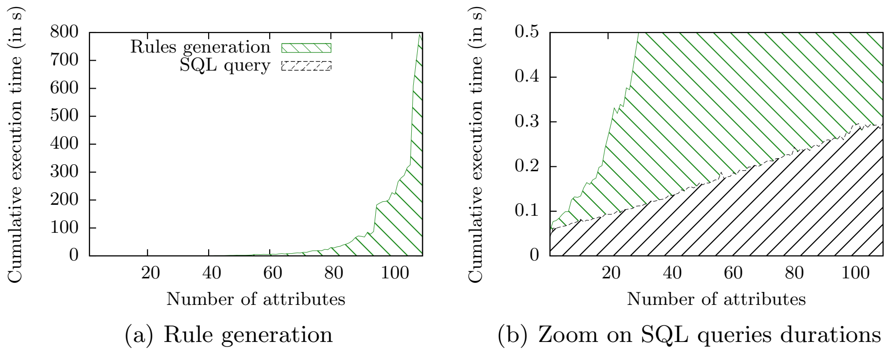


Fig. 5. Execution time for null rules

5.2 Experimental Results

The server used for these experiments is a virtual machine running on VMware 4.1 with 2× Intel Xeon X5650 and 7.2K disks in RAID 5. This virtual machine, running Debian 6.0, disposes of 16 GB of RAM and 4 CPU cores. The DBMS is Oracle Database 11g Release 2.

In these experiments, we consider three families of $\mathcal{RQL}$ queries.

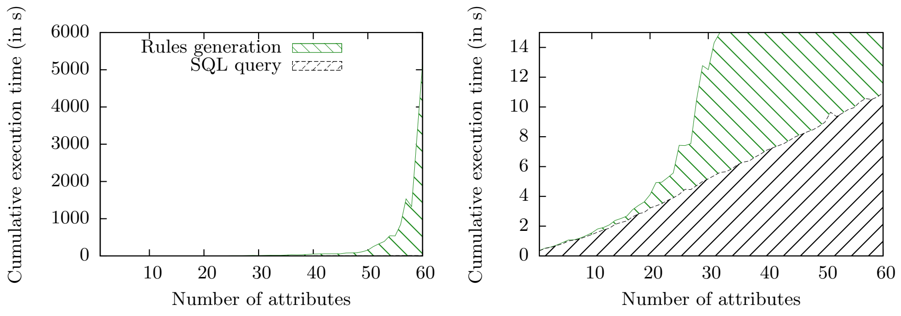
Q1: rules for null values The first set of queries mine rules between sensors for null values. Such queries can be used to identify groups of sensors which are unavailable simultaneously, due, for example, to a shared acquisition system.

```
FINDRULES
OVER <list of attributes>
SCOPE t1 sensors
CONDITION ON A IS t1.A IS NULL;
```

For example, if we consider all temperature sensors as the list of attributes, we can group sensors dining_room_temperature_1 (A), dining_room_temperature_2 (B), dining_room_temperature_3 (C), dining_room_temperature_4 (D), hall_temperature_0 (E) and hall_temperature_3 (F) according to rules ($F \to C$, $A \to CF$, $AD \to E$, $BE \to D$, $EF \to A$, $AB \to DE$, $CE \to AF$, $DF \to AE$, $BF \to ADE$, $CD \to AEF$, $BC \to ADEF$). Similarly, we can group four sensors from the living room, two sensors from the office, two sensors from the hall with three sensors from the kitchen, etc.

Figure 5 gives the cumulative execution time for various number of attributes in the *OVER* clause of Q1. As expected, rule generation is by far the most expensive part when the query runs over a large set of attributes. The SQL query however lasts less than a second and increases linearly: computation of the base by the DBMS is efficient.

Q2: Functional dependencies The second set of queries finds functional dependencies within a time window. This time window is specified using SQL conditions on the timestamp.



(a) Rule generation (b) Zoom on SQL queries durations

Fig. 6. Execution time for functional dependencies

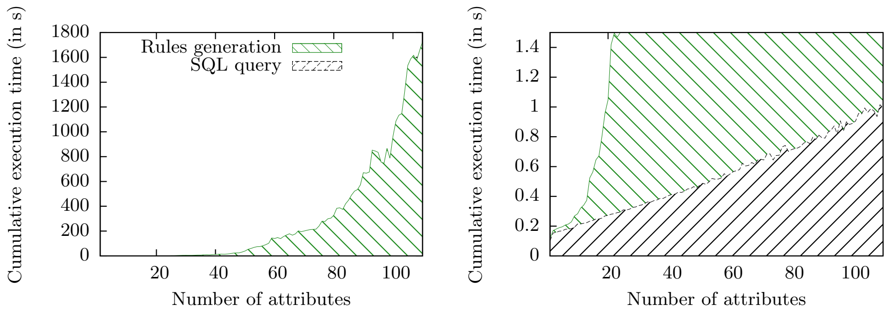
```
FINDRULES
OVER <list of attributes>
SCOPE t1,t2 (
  SELECT * FROM sensors
  WHERE time BETWEEN '2006-08-31 00:00:00'
                 AND '2006-08-31 23:59:59')
WHERE t1.rowid < t2.rowid
CONDITION ON A IS t1.A = t2.A;
```

To generate the base, the corresponding SQL query performs a Cartesian product (more precisely, a theta self-join on t1.rowid < t2.rowid, which gives half as many tuples). Consequently, the SQL part is significantly more costly compared with the previous family of $\mathcal{RQL}$ queries. Figure 6 gives the cumulative execution time for various number of attributes in the *OVER* clause of Q2.

Q3: rules for local maximums $\mathcal{RQL}$ queries can express a wide range of conditions for rules. For example, the following query finds rules for local maximums of measurements (i.e. $X \rightarrow A$ is interpreted as: if all sensors in X have a local maximum at time T, then sensor A has a local maximum at time T).

```
FINDRULES
OVER <list of attributes>
SCOPE t1,t2,t3 sensors
WHERE t2.time = t1.time+interval '1' minute
  AND t3.time = t2.time+interval '1' minute
CONDITION ON A IS t1.A < t2.A AND t2.A > t3.A;
```

Even though this query has three tuple-variables, all three are bound by a 1:1 relationship. Consequently, this query is computed efficiently. Figure 7 shows performances similar to Q1.



(a) Rule generation (b) Zoom on SQL queries durations

Fig. 7. Execution time for local maximums rules

6 Conclusion

In this paper, we have introduced $SafeR\mathcal{L}$, a logical query language based on TRC and devoted to rule discovery in databases. The rule mining problem is seen as a query processing problem, for which we have proposed a query rewriting technique allowing the delegation of as much processing as possible to the underlying DBMS engine. $\mathcal{RQL}$, the concrete language of $Safe\mathcal{RL}$, is a SQL-like pattern mining language which allows SQL developers to extract precise information without specific knowledge in data mining. A system has been developed and tested against a real-life database provided by the MIT House_n Consortium and Microsoft Research [24]. These experiments show both the feasibility and the efficiency of the proposed language. A web prototype for $\mathcal{RQL}$ has been released and is available for teaching and research: `http://rql.insa-lyon.fr`

References

1. MarketResearch.com: Worldwide relational database management systems 2012-2016 forecast (Aug 2012)
2. Bohannon, P., Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for data cleaning. In: Proceedings of the 23rd International Conference on Data Engineering. ICDE '07 (2007) 746–755
3. Blockeel, H., Calders, T., Fromont, E., Goethals, B., Prado, A., , Robardet, C.: A practical comparative study of data mining query languages. In Springer, ed.: Inductive Databases and Constraint-Based Data Mining. Sao Deroski, Bart Goethals, Pane Panov (2010) 59–77
4. Guns, T., Nijssen, S., Raedt, L.D.: Itemset mining: A constraint programming perspective. Artif. Intell. **175**(12-13) (2011) 1951–1983
5. Métivier, J.P., Boizumault, P., Crémilleux, B., Khiari, M., Loudni, S.: A constraint-based language for declarative pattern discovery. In: Declarative Pattern Mining Workshop, ICDM 2011. (2011) 1112–1119
6. Calders, T., Wijsen, J.: On monotone data mining languages. In: Proceedings of the 8th International Workshop on Database Programming Languages. DBPL '01 (2001) 119–132

7. Fang, L., LeFevre, K.: Splash: ad-hoc querying of data and statistical models. In: Proceedings of the 13th International Conference on Extending Database Technology. EDBT '10 (2010) 275–286
8. Ordonez, C., Pitchaimalai, S.K.: One-pass data mining algorithms in a DBMS with UDFs. In: SIGMOD Conference. (2011) 1217–1220
9. Blockeel, H., Calders, T., Fromont, É., Goethals, B., Prado, A., Robardet, C.: An inductive database system based on virtual mining views. Data Min. Knowl. Discov. **24**(1) (2012) 247–287
10. Agier, M., Petit, J.M., Suzuki, E.: Unifying framework for rule semantics: Application to gene expression data. Fundam. Inform. **78**(4) (2007) 543–559
11. Agier, M., Froidevaux, C., Petit, J.M., Renaud, Y., Wijsen, J.: On Armstrong-compliant logical query languages. In: Proceedings of the 4th International Workshop on Logic in Databases. LID '11 (2011) 33–40
12. Lopes, S., Petit, J.M., Lakhal, L.: Functional and approximate dependency mining: database and FCA points of view. J. Exp. Theor. Artif. Intell. **14**(2-3) (2002) 93–114
13. Medina, R., Nourine, L.: A unified hierarchy for functional dependencies, conditional functional dependencies and association rules. In: ICFCA. Volume 5548 of LNCS., Springer (2009) 98–113
14. Medina, R., Nourine, L.: Conditional functional dependencies: An FCA point of view. In: ICFCA. (2010) 161–176
15. Babin, M.A., Kuznetsov, S.O.: Computing premises of a minimal cover of functional dependencies is intractable. Discrete Applied Mathematics **161**(6) (2013) 742–749
16. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
17. Armstrong, W.W.: Dependency structures of data base relationships. In: Proceedings of the IFIP Congress. (1974) 580–583
18. Ganter, B., Wille, R.: Formal Concept Analysis. Springer (1999)
19. Caspard, N., Monjardet, B.: The lattices of closure systems, closure operators, and implicational systems on a finite set: A survey. Discrete Applied Mathematics **127**(2) (2003) 241–269
20. Mannila, H., Räihä, K.J.: Algorithms for inferring functional dependencies from relations. Data and Knowldege Engineering **12**(1) (1994) 83–99
21. Gottlob, G., Libkin, L.: Investigations on Armstrong relations, dependency inference, and excluded functional dependencies. Acta Cybernetica **9**(4) (1990) 385–402
22. Demetrovics, J., Thi, V.D.: Some remarks on generating Armstrong and inferring functional dependencies relation. Acta Cybernetica **12**(2) (1995) 167–180
23. Murakami, K., Uno, T.: Efficient algorithms for dualizing large-scale hypergraphs. CoRR **1102.3813** (2011)
24. Intille, S.S., Larson, K., Tapia, E.M., Beaudin, J.S., Kaushik, P., Nawyn, J., Rockinson, R.: Using a live-in laboratory for ubiquitous computing research. In: PERVASIVE '06. (2006) 349–365

A Constraint Programming Approach for Mining Sequential Patterns in a Sequence Database

Jean-Philippe Métivier[1], Samir Loudni[1], and Thierry Charnois[2]

[1] GREYC (CNRS UMR 6072) – University of Caen
Campus II Côte de Nacre, 14000 Caen - France
[2] LIPN CNRS (UMR 7030) – University PARIS 13
99, avenue Jean-Baptiste Clment 93430 Villetaneuse - France

Abstract. Constraint-based pattern discovery is at the core of numerous data mining tasks. Patterns are extracted with respect to a given set of constraints (frequency, closedness, size, etc). In the context of sequential pattern mining, a large number of devoted techniques have been developed for solving particular classes of constraints. The aim of this paper is to investigate the use of Constraint Programming (CP) to model and mine sequential patterns in a sequence database. Our CP approach offers a natural way to simultaneously combine in a same framework a large set of constraints coming from various origins. Experiments show the feasibility and the interest of our approach.

1 Introduction

Sequential pattern mining is a well-known data mining technique introduced in [1] to find regularities in a database of sequences. This problem is central in many application domains, such as web usage mining [7], bioinformatics and text mining [3]. For effectiveness and efficiency considerations, many authors [10, 24] have promoted the use of constraint to focus on the most promising knowledge by reducing the number of extracted patterns to those of a potential interest given by the final user. The most popular example is the minimal frequency constraint: it addresses all sequences having a number of occurrences in the database exceeding a given minimal threshold.

There are already in the literature many algorithms to extract *sequential patterns* (e.g. `GSP` [19], `SPADE` [25], `PrefixSpan` [15]), *closed* sequential patterns (e.g. `CloSpan` [23], `BIDE` [22]) or sequential patterns satisfying *regular expression* (e.g. `SPIRIT` [11]). All the above methods, though efficient, suffer from two major problems. First, they tackle particular classes of constraints (i.e. *monotonic* and *anti-monotonic* ones) by using devoted techniques. However, several practical constraints required in data mining tasks, such as regular expression, aggregates, do not fit into these classes. Second, they lack of generic methods to push various constraints into the sequential pattern mining process. Indeed, adding and handling simultaneously several types of constraints in a nice and elegant way beyond the few classes of constraints studied is not still trivial. The lack of generic approaches restrains the discovery of useful patterns because the user has to develop a new method each time he wants to extract patterns satisfying a new type of constraints. In this paper, we address this open issue by proposing a generic approach for modelling and mining sequential patterns under various constraints using Constraint Programming (CP).

Our proposition benefits from the recent progress on cross-fertilization between data mining and CP for itemset mining [12, 14, 17]. The common point of all these methods is to model in a declarative way pattern mining as Constraint Satisfaction Problems (CSP), whose resolution provides the complete set of solutions satisfying all the constraints. The great advantage of this modelling is its flexibility, it enables to define and to push new constraints without having to develop new algorithms from scratch.

The key contribution of this paper is to propose a CP modelling for the problem of mining sequential patterns in a sequence database [1]. Our approach addresses in a unified framework a large set of constraints. This includes constraints such as frequency, closedness and size, and other constraints such as regular expressions, gap and constraints on items. Moreover, our approach enables to combine simultaneously different types of constraints. This leads to the first CP-based model for discovering sequential patterns in a sequence database under various constraints. Experiments on a case study on biomedical literature for discovering gene-RD relations from PubMed articles show the feasibility and the interest of our approach.

This paper is organized as follows. Section 2 gives the necessary definitions and presents the problem formulation. Section 3 introduces the main principles of constraint programming. Section 4 describes our CP model for mining sequential patterns in a sequence database. We review some related work in Section 5 and Section 6 reports in depth a case study from the biomedical literature domain for discovering gene-RD relations from PubMed articles. Finally, we conclude and draw some perspectives.

2 Sequential Pattern Mining

In this section, we introduce sequential patterns defined by Agrawal et al. [1].

2.1 Sequential Patterns

Sequential pattern mining [1] is a data mining technique that aims at discovering correlations between events through their order of appearance. Sequential pattern mining is an important field of data mining with broad applications (e.g., biology, marketing, security) and there are many algorithms to extract frequent sequences [19, 25, 23].

In the context of sequential patterns extraction, a *sequence* is an ordered list of literals called *items*. A sequence s is denoted by $\langle i_1, i_2 \ldots i_n \rangle$ where i_k, $1 \leq k \leq n$, is an item. Let $s_1 = \langle i_1, i_2 \ldots i_n \rangle$ and $s_2 = \langle i'_1, i'_2 \ldots i'_m \rangle$ be two sequences. s_1 is *included* in s_2 if there exist integers $1 \leq j_1 < j_2 < \ldots < j_n \leq m$ such that $i_1 = i'_{j_1}$, $i_2 = i'_{j_2}$, ..., $i_n = i'_{j_n}$. s_1 is called a *subsequence* of s_2. s_2

Table 1. SDB_1: a sequence database

Sequence identifier	Sequence
1	$\langle a\ b\ c\ d\ a \rangle$
2	$\langle d\ a\ e \rangle$
3	$\langle a\ b\ d\ c\ \rangle$
4	$\langle c\ a \rangle$

is called a *super-sequence* of s_1, denoted by $s_1 \preceq s_2$. For example the sequence $\langle a\ b\ d\ c \rangle$ is a super-sequence of $\langle b\ c \rangle$: $\langle b\ c \rangle \preceq \langle a\ b\ d\ c \rangle$. A *sequence database SDB* is a set of tuples (sid, s), where *sid* is a sequence identifier and s a sequence. For instance, Table 1 represents a sequence database of four sequences. A tuple (sid, s) *contains* a sequence s_1, if $s_1 \preceq s$. The *support* of a sequence s_1 in a sequence database *SDB*, denoted $sup(s_1)$, is the number of tuples containing s_1 in the database[1]. For example, in Table 1, $sup(\langle c\ a \rangle) = 2$.

A *frequent sequential pattern* is a sequence such that its support is greater or equal to a given threshold: *minsup*. The *sequential pattern mining* problem is to find the *complete* set of frequent sequential patterns with respect to a given sequence database SDB and a support threshold *minsup*.

2.2 Sequential Pattern Mining under Constraints

In order to drive the mining process towards the user objectives and to eliminate irrelevant patterns, one can define constraints [10]. The most commonly used constraint is the frequency constraint (that assigns a value to *minsup*). We review some of the most important constraints for the sequential mining problem [10].

Closedness Constraint The *closed sequential patterns* [23] are a condensed representation of the whole set of sequential patterns. This condensed representation eliminates redundancies according to the frequency constraint. A frequent sequential pattern s is closed if there exists no other frequent sequential pattern s' such that $s \preceq s'$ and $sup(s) = sup(s')$. For instance, with $minsup = 2$, the sequential pattern $\langle b\ c \rangle$ from Table 1 is not closed whereas the pattern $\langle a\ b\ c \rangle$ is closed.

Item constraint. An item constraint specifies subset of items that should or should not be present in the sequential patterns. For instance, if we impose the constraint $C_{item} \equiv sup(p) \geq 2 \wedge (a \in p) \wedge (b \in p)$, three sequential patterns are mined from Table 1: $p_1 = \langle a\ b \rangle$, $p_2 = \langle a\ b\ c \rangle$ and $p_3 = \langle a\ b\ d \rangle$.

Size constraint. The aim of this constraint is to limit the length of the patterns, the length being the number of occurrences of items. The length of a pattern

[1] The relative support is also used:
$$sup_{SDB}(T) = \frac{|\{(sid, s)\ s.t.\ (sid, s) \in SDB \wedge (T \preceq s)\}|}{|SDB|}$$

p will be denoted by $len(p)$. For example, if $len(p) \geq 3 \wedge sup(p) \geq 2$, only two sequential patterns are extracted (p_2 and p_3).

Gap constraint. Another widespread constraint is the gap constraint. A sequential pattern with a gap constraint $C_{gap} \equiv [M, N]$, denoted by $p_{[M,N]}$, is a pattern such as at least M items and at most N items are allowed between every two neighbor items, in the original sequences. For instance, let $p_{[0,2]} = \langle c\ a \rangle$ and $p_{[1,2]} = \langle c\ a \rangle$ be two patterns with two different gap constraints and let us consider the sequences of Table 1. Sequences 1 and 4 support pattern $p_{[0,2]}$ (sequence 1 contains one item between (c) and (a) whereas sequence 4 contains no item between (c) and (a)). But only Sequence 1 supports $p_{[1,2]}$ (only sequences with one or two items between (c) and (a) support this pattern).

Regular expression constraint. A regular expression constraint C_{RE} is a constraint specified as a regular expression over the set of items. A sequential pattern satisfies C_{RE} if and only if the pattern is accepted by its equivalent deterministic finite automata [11]. For instance, the two sequential patterns $\langle a\ b\ c \rangle$ and $\langle a\ d\ c \rangle$ from Table 1 satisfy the regular expression constraint $C_{RE} = a * \{bb|bc|dc\}$.

3 Constraint Programming

In this section, we first introduce basic constraint programming concepts and then present two constraints of interest: `Among` and `Regular`.

3.1 Preliminaries

Constraint programming (CP) is a generic framework for solving combinatorial problems modelled as Constraint Satisfaction Problems (CSP). The key power of CP lies in its declarative approach towards problem solving: in CP, the user specifies the set of constraints which has to be satisfied, and the CP solver generates the correct and complete set of solutions. In this way, the specification of the problem is separated from the search strategy.

A *Constraint Satisfaction Problem* (CSP) consists of a finite set of variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ with finite domains $\mathcal{D} = \{D_1, \ldots, D_n\}$ such that each D_i is the set of values that can be assigned to X_i, together with a finite set of constraints $\mathcal{C}$, each on a subset of $\mathcal{X}$. A constraint $C \in \mathcal{C}$ is a subset of the cartesian product of the domains of the variables that are in C. The goal is to find an assignment $(X_i = d_i)$ with $d_i \in D_i$ for $i = 1, \ldots, n$, such that all constraints are satisfied. This assignment is called a solution to the CSP. For a given assignment t, $t[X_i]$ denotes the value assigned to X_i in t.

In Constraint Programming (see [2]), the solution process consists of iteratively interleaving search phases and propagation phases. The search phase essentially consists of enumerating all possible variable-value combinations, until we find a solution or prove that none exists. It is generally performed on a tree-like structure. In order to avoid the systematic generation of all the combinations and reduce the search space, the propagation phase shrinks the search space: each constraint propagation algorithm removes values that a priori cannot be part of a solution w.r.t. the partial assignment built so far. The removal of inconsistent domain values is called *filtering*.

An important modelling technique from CP are the *global constraints* that provide shorthands to often-used combinatorial substructures. Global constraints embed specialized filtering techniques that exploit the underlying structure of the constraint to establish stronger levels of consistency much more efficiently. Nowadays, global constraints are considered to be one of the most important components of CP solvers in practice.

3.2 The Among and Regular Global Constraints

Among Constraint. This constraint restricts the number of occurrences of some given values in a sequence of n variables [5]:

Definition 1 (`Among` constraint, [5]). *Let $X = X_1, \ldots, X_n$ be a sequence of n variables, S a set of values, D^X the cartesian product of the variable domains in X. Let l and u be two integers s.t. $0 \leq l \leq u \leq n$.*

$$\texttt{Among}(X, S, l, u) = \{t \in D^X \mid l \leq |\ \{i, t[X_i] \in S\}\ | \leq u\}$$

The `Among` constraint can be encoded by channelling into 0/1 variables using the sum constraint [6]: $\forall i \in \{1, \ldots, n\}\ B_i = 1 \ \leftrightarrow\ t[X_i] \in S \ \wedge\ l \leq \sum_{i=1}^{n} B_i \leq u$.

Regular Constraint. Given a deterministic finite automaton M describing a regular language, constraint $\texttt{Regular}(X, M)$ ensures that every sequence of values taken by the variables of X have to be a member of the regular language recognised by M:

Definition 2 (`Regular` constraint, [16]). *Let M be a Deterministic Finite Automaton (DFA), $\mathcal{L}(M)$ the language defined by M, X a sequence of n variables. $\texttt{Regular}(X, M)$ has a solution iff $\exists\, t \in D^X$ s.t. $t \in \mathcal{L}(M)$.*

In [16], `Regular` constraint over a sequence of n variables is modelled by a layered directed graph $\mathcal{G} = (V, U)$, and a solution to $\texttt{Regular}(X,M)$ corresponds to an s-t path in graph $\mathcal{G}$, where s is the "source" node and t the "sink" node.

4 Modeling Sequential Patterns using CP

This section presents our CP modelling for the sequential pattern mining problem. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n items, EOS a symbol not belonging to I ($EOS \notin I$) denoting the end of a sequence, SDB a set of m sequences and ℓ the maximal length of the sequence in SDB.

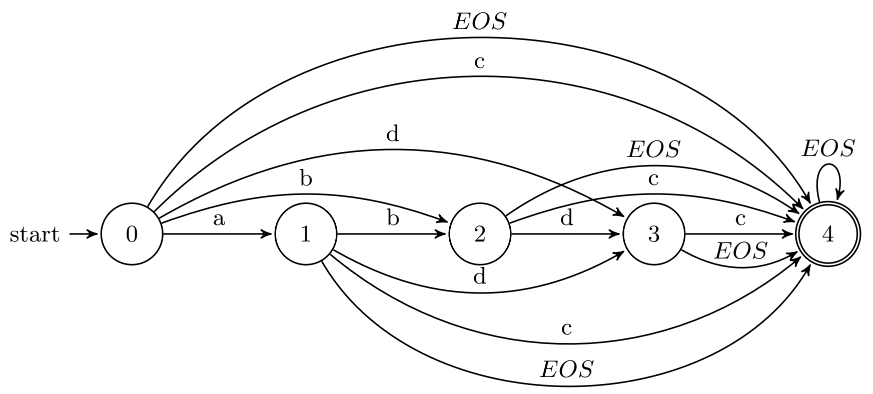


Fig. 1. The automaton modelling all subsequences of the sequence $\langle a\ b\ d\ c \rangle$.

4.1 Modelling an Unknown Sequential Pattern

Let p be the unknown sequential pattern we are looking for. First, ℓ variables $\{P_1, P_2, \ldots, P_\ell\}$ having $D_i = I \cup \{EOS\}$ for domain are introduced to represent p. Second, m boolean variables S_s (having $\{0, 1\}$ for domain) are used such that $(S_s = 1)$ iff sequence s contains the unknown sequential pattern p:

$$(S_s = 1) \Leftrightarrow (p \preceq s) \qquad (1)$$

In equation (1), $(S_s = 1)$ if pattern p is a *subsequence* of s; 0 otherwise. So, $sup(p) = \Sigma_{s \in SDB}\, S_s$.

4.2 Modelling Sequential Pattern Mining

Let SDB be a sequence database and let a support threshold $minsup$. To encode that "$p \preceq s$", we first have to generate an automaton A_s capturing all subsequences that can be found inside a given sequence s. Then, we have to impose a `Regular` constraint stating that the unknown pattern p must be recognized by the automaton A_s.

To reduce the number of states of the automaton, for each sequence, we consider only its frequent items in the SDB w.r.t. $minsup$. Indeed, any super-pattern of an infrequent item cannot be frequent. Figure 1 shows an example of automaton generated for the third sequence of Table 1. Algorithm 1 depicts the pseudo-code for generating the automaton A_s.

To enumerate the complete set of frequent sequential patterns with respect to a given sequence database SDB and a support threshold $minsup$, we need to express that the unknown sequential pattern p occurs at least $minsup$ times. This problem is modelled by the following constraints:

Algorithm 1: Pseudo-code for generating A_s.

```
function generateAutomaton(Sequence $s$)
Automaton $A_s$;
$A_s$.nbState ← length($s$) + 1;
$A_s$.addInitialState(0);
$A_s$.addAcceptingState(length($s$));
foreach ($state \in [0,$length$(s)]$) do
    foreach ($position \in [state+1,$length$(s)]$) do
        $item$ ← getItem($s$,$position$);
        $A_s$.addTransition($state$,$item$,$position$);
    $A_s$.addTransition($state$,$EOS$,length($s$));
return $A_s$;
```

Theorem 1 (Frequent Sequential Pattern Mining). *Sequential pattern mining is expressed by the following constraints:*

$$\forall s \in SDB : S_s = 1 \leftrightarrow \texttt{Regular}(p, A_s) \tag{2}$$

$$sup(p) = \sum_{s \in SDB} S_s \geq minsup \tag{3}$$

Proof. The reified constraint (2) models the support constraint. By construction, the automaton A_s encodes all sequential patterns that are subsequences of the sequence s. If the `Regular` constraint is satisfied (resp. not satisfied), then S_s must be equal to 1 (resp. must be equal to 0). The propagation is also performed, in a same way, from the equality constraint toward the `Regular` constraint.

The frequency constraint (3) enforces that at least *minsup* variables from S must take value 1. Together with constraint (2), it enforces that at least *minsup* sequences must support the sequential pattern described by p. □

4.3 Modelling other Constraints

This section shows how our CP approach enables us to express in a straightforward way constraints presented in Section 2.2.

Closedness Constraint. By definition a *closed* sequential pattern is the largest pattern that is contained in all selected sequences. Intuitively, in our encoding this corresponds to the sequential pattern having the less number of variables P_i instantiated to EOS. Thus, the satisfaction problem is turned into an optimization one: *minimize* the numbers of variables P_i instantiated to EOS. To express this minimization problem, we first have to add for each variable P_i an unary constraint c_i stating that if ($P_i = EOS$) we have to pay a cost 1; 0 otherwise.

Then, we have to minimize the cost function $c(p) = \sum_{P_i \in p} c_i$ to obtain a closed sequential pattern:

$$\begin{array}{ll} \texttt{minimize}_p & c(p) \\ sup(p) & \geq minsup \end{array} \tag{4}$$

To enumerate the complete set of closed sequential patterns with respect to a given sequence database SDB and a support threshold $minsup$, we need to avoid future patterns to be equal to the previously found closed patterns. So, each time a frequent sequential pattern p is proven closed, we dynamically add a new constraint to forbid it.

Item Constraint. In order to specify that a subset of items should or should not be present in the sequential patterns, we have to add the following constraint:

$$\texttt{Among}(p, V, [l, u]) \tag{5}$$

where V is a subset of items, l and u are two integers s.t. $0 \leq l \leq u \leq \ell$. The `Among` constraint enforces that the number of variables of p that take a value from V is at least l and at most u. Since p represents the sequential patterns we are looking for, the previous constraint ensures that items of V should be present at least l times in the mined patterns.

To express the fact that the sequential patterns should not contain any item of V, we just have to set l and u to 0.

Size Constraint. In order to consider the frequent sequential patterns of size k, we just have to add the following constraints:

$$\forall i \in [1 \ldots k] : P_i \neq EOS \tag{6}$$

$$\forall i \in [k+1 \ldots \ell] : P_i = EOS \tag{7}$$

The previous constraints enforce that the k first variables of p must be different from EOS, while the $(\ell - k)$ remaining variables of p must be equal to EOS.

The minimum size constraint (i.e. $len(p) \geq k$) can be formulated by the following constraint:

$$\forall i \in [1 \ldots k] : P_i \neq EOS \tag{8}$$

For the maximum size constraint (i.e. $len(p) \leq k$), this can be modelled as follows:

$$\forall i \in [k+1 \ldots \ell] : P_i = EOS \tag{9}$$

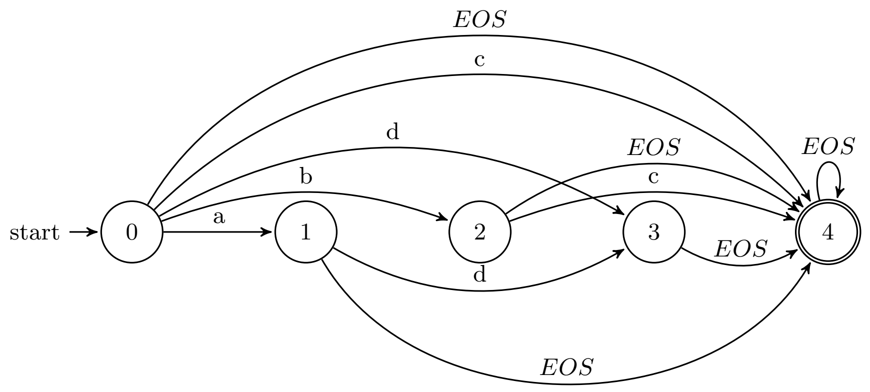


Fig. 2. The new automaton modelling all subsequences of the sequence $\langle a\ b\ d\ c \rangle$ satisfying the gap constraint [1,1].

Gap Constraint. Let $p_{[M,N]}$ be the sequential pattern satisfying the gap constraint $[M, N]$. To encode this constraint we have to modify the construction of the automaton A_s (cf. Theorem 1) in a such way that only transitions respecting the gap constraint will be kept. Let $A_s^{[M,N]}$ be the new resulting automaton representing all sequential patterns that are subsequence of the sequence s and satisfying the gap constraint. Finally, the reified constraint (2) is rewrited as follows:

$$\forall s \in SDB : S_s = 1 \leftrightarrow \texttt{Regular}(p, A_s^{[M,N]}) \tag{10}$$

Theorem 2 (Frequent Sequential Pattern Mining with Gap). *Sequential pattern mining with a gap constraint is expressed by constraints (3) and (10).*

Figure 2 gives the new automaton obtained from the automaton of Figure 1 with the gap constraint $[1, 1]$. For instance, the sequential pattern $\langle a\ b \rangle$ does not satisfy the gap constraint; it is not recognized by the new automaton.

To generate the automaton $A_s^{[M,N]}$ for a sequence s, we need to modify Algorithm 1 in a such way that only valid transitions satisfying the gap constraint $[M, N]$ are considered . This is done by adding the following condition inside the second loop foreach: ($state == 0 \mid\mid M \leq position{-}state \leq N$) (see Algorithm 1).

Regular Expression Constraint. Let A_{re} be an automaton encoding a regular expression over the set of items. Then, the regular expression constraint can be formulated as follows:

$$\texttt{Regular}(p, A_{re}) \tag{11}$$

As presented in [8], a regular expression can be translated into a deterministic finite automaton. Thus, the `Regular` constraint over p ensures that every sequence of values taken by the variables of p have to be a member of the regular language recognised by A_{re}, therefore recognized by the regular expression associated to A_{re}.

5 Related Work

Computing Sequential Patterns. In the context of constraint-based sequential pattern mining, several algorithms have been proposed [19, 25, 15, 23, 22, 11]. All these algorithms exploit properties of the constraints (i.e., monotonicity, anti-monotonicity or succinctness) to perform effective pruning. For constraints that do not fit in these categories, they are handled by relaxing them to achieve some nice property (like anti-monotonicity) facilitating the pruning. For instance, Garofalakis et al. [11] proposed regular expressions as constraints and developed a family of `SPIRIT` algorithms each achieving a different kind of relaxation of the regular expression constraint. Such a method, though interesting, makes tricky the integration of such constraints in a nice and elegant way. So, unlike these algorithms, our approach enables to address in a unified framework a broader set of constraints, and more importantly, to combine them simultaneously.

CP for Pattern Mining. In the context of local patterns, an approach using CP for itemset mining has been proposed in [17]. This approach addresses in a unified framework a large set of local patterns and constraints such as frequency, closedness, maximality, constraints that are monotonic or anti-monotonic. To deal with richer patterns satisfying properties involving several local patterns, different extensions have been proposed, such as pattern sets [13], n-ary patterns [14], top-k patterns [20] or skypatterns [21]. Our approach also benefits from the recent progress on cross-fertilization between data mining and CP for itemset mining, but it addresses a different problem with a different modelling.

CP for Sequence Mining. More recently, Coquery et al. [9] have proposed a SAT-Based approach for Discovering frequent, closed and maximal sequential patterns with wildcards in only a single sequence of items. However, unlike [9], our approach considers a database of sequences of items. Moreover, in [9], the sequential patterns with non-contiguous items are modelled using empty items as wildcards. But the gap between the items have to be fixed. Then, for instance the two sequential patterns $\langle a\ o\ b\rangle$ and $\langle a\ o\ o\ b\rangle$ are considered different. On the contrary, our modelling enables us to define any (minimal or maximal) value for the gap.

6 Experimentations

Experiments are conducted on texts from biological and medical texts. The goal is to discover relations between genes and rare diseases. The details of this application is given in [4]. In this section, we focus on the extraction of sequential patterns using our CP approach, and we give quantitative results showing the relevant of the approach.

6.1 Case Study

Settings. We created a corpus from the PubMed database using HUGO[2] dictionary and Orphanet dictionary to query the database to get sentences having at least one rare disease and one gene. $17,527$ sentences have been extracted in this way and we labelled the gene and rare disease (RD) names thanks to the two dictionaries. For instance, the sentence "*<**disease**>**Muir-Torre syndrome**<\**disease**> is usually inherited in an autosomal dominant fashion and associated with mutations in the mismatch repair genes, predominantly in <**gene**>**MLH1**<\gene> and <**gene**>**MSH2**<\gene> genes.*" contains one recognized RD, and two recognized genes. These $17,527$ sentences are the training corpus from which we experiment the sequential pattern extraction.

Sequential Pattern Extraction. Sequences of the SDB are the sentences of the training corpus: an item corresponds to a word of the sentence. We carry out a POS tagging of the sentences thanks to the TreeTagger tool [18]. In the sentences, each word is replaced by its lemma, except for gene names (respectively disease names) which are replaced by the generic item $GENE$ (respectively $DISEASE$). Note that unlike machine learning based methods, our approch does not require to annotate the relations: they are discovered.

In order to discover sequential patterns, we use usual constraints such as *the minimal frequency* and *the minimal length* constraints and other useful constraints expressing some *linguistic knowledge* (e.g. *membership* and *association* constraints). The goal is to retain sequential patterns which convey linguistic regularities (e.g., gene-rare disease relationships). Our method offers a natural way to simultaneously combine in a same framework these constraints coming from various origins. We briefly sketch these constraints.

- *The minimal frequency constraint.* Three values of minimal frequency have been experimented: 2%, 5%, and 10%.
- *The minimal length constraint.* The aim of this constraint is to remove sequential patterns that are too small w.r.t. the number of items (number of words) to be relevant linguistic patterns. We tested this constraint with a value set to 3.
- *The membership constraint.* This constraint enables to filter out sequential patterns that do not contain some selected items. For example, we express that the extracted patterns must contain at least three items (expressing the linguistic relation): $GENE$, $DISEASE$ and noun or verb[3]. We used the item constraint to enforce this constraint.
- *The association constraint.* This constraint expresses that all sequential patterns that contain the verb item must contain its lemma and its grammatical category. We used the item constraint to enforce this constraint.
- *The closedness constraint.* In order to exclude redundancy between patterns, we used *closed* sequential patterns.

[2] www.genenames.org

[3] For each word (i.e. item), its grammatical category is stored in a base.

#sentences	50		100		150		200		250	
	#sol.	time	#sol.	time	#sol.	time	#sol.	time	#sol.	time
freq > 2%	129	1,105	329	12,761	441	35,164	(89)	–	(34)	–
freq > 5%	47	285	67	1,571	81	2,091	94	4,119	119	8,516
freq > 10%	4	53	21	251	26	577	29	1,423	28	2,764

#sentences	300		350		400		450		500	
	#sol.	time	#sol.	time	#sol.	time	#sol.	time	#sol.	time
freq > 2%	(129)	–	(45)	–	(10)	–	(1)	–	(0)	–
freq > 5%	101	9,620	93	16057	83	21,764	84	35,962	(26)	–
freq > 10%	30	5147	24	4,493	23	7,026	20	13,744	21	17,708

Table 2. Results obtained on different subsets of the PubMed dataset.

Experimental Protocol. We carried out experiments on several subsets of the PubMed dataset with different sizes ranging from 50 to 500 sentences. A timeout of 10 hours has been used. For each subset, we report the number of extracted closed sequential patterns and the CPU-times to extract them (in seconds). When the timeout is reached, the number of extracted patterns (until the timeout) is given in parenthesis.

All experiments were conducted on AMD Opteron 2.1 GHz and a RAM of 256 GB. We implemented our proposal in C++ using the library toulbar2[4] for solving constrained optimization problems modelled as *Cost Function Network* (CFN).

6.2 Results

Table 2 reports the results we obtained on different subsets of the PubMed dataset with values of *minsup* ranging from 2% to 10%. From these results, we can draw the following remarks.

i) Soundness and Flexibility. As the resolution performed by the CP solver is sound and complete, our approach is able to mine the correct and complete set of sequential patterns satisfying the different constraints. We compared the sequential patterns extracted by our approach with those found by [4], and the two approaches return the same set of patterns. Table 2 depicts the number of closed sequential patterns according to *minsup*. As expected, the lower *minsup* is, the larger the number of extracted sequential patterns.

ii) Highlighting useful sequential patterns. Our approach allowed to extract several relevant *linguistic patterns*. Such patterns can be used to explain RD-gene relationships from PubMed articles. For instance, three sequential patterns of great interest were highlighted by the expert:

1. $\langle(DISEASE)\ (be)\ (cause)\ (by)\ (mutation)\ (in)\ (the)\ (GENE)\rangle$
2. $\langle(GENE)\ (occur)\ (in)\ (DISEASE)\rangle$

[4] `https://mulcyber.toulouse.inra.fr/projects/toulbar2.`

3. $\langle (DISEASE)\ (be)\ (an)\ (mutation)\ (in)\ (GENE) \rangle$

From a biomedical point of vue, these sequential patterns are interesting since they convey a notion of *causality* (i.e. gene *cause* rare disease).

iii) Computational Efficiency. This experiment quantify runtimes and the scalability of our approach. In practice, runtimes vary according to the size of the datasets. For datasets with size up to 200, the set of all solutions is computed. We observe that runtimes vary from few seconds for high frequency thresholds to about few hours for low frequency thresholds. However, for large size datasets (≥ 200) and low frequency thresholds (i.e. $minsup = 2\%$), the CP approach does not succeed to complete the extraction of all closed sequential patterns within a timeout of 10 hours. Indeed, with the increase of the size of the dataset, the search space and the runtime increase drastically, and the solver spends much more time to find the first solution.

Finally, note that comparing runtimes with those obtained by ad hoc approaches would be rather difficult. In fact, these approaches use devoted techniques and do not offer the same level of genericity and expressivity as in our CP approach. Moreover, they cannot push in depth simultaneously different categories of constraints. Above all, there does not exist any algorithm, neither tool, for extracting sequential patterns under all the contraints proposed in our work.

7 Conclusion

We have proposed a flexible approach to mine sequential patterns of items in a sequence database. The declarative side of the CP framework easily enables us to address a large set of constraints and leads to a unified framework for extracting sequential patterns under constraints. Finally, the feasibility and the interest of our approach has been highlighted through experiments on a case study in biomedical literature for discovering gene-RD relations from PubMed articles.

We are currently investigating a new direction to enhance the efficiency of our approach: instead of constructing an automaton for every sequence, it would be more efficient to build some variant of a Prefix Tree Automata on the original dataset to avoid some redundancies. Furthermore, we intend to extend our approach to discover sequential patterns of itemsets in a sequence database. Discovering pattern sets is an attractive road to propose actionable patterns and the CP modelling is a proper paradigm to tackle this challenge [13, 14]. Further work is to address this issue.

Acknowledgements. This work is partly supported by the ANR (French Research National Agency) funded project FiCOLOFO ANR-10-BLA-0214. We would like to thank Bruno Crémilleux for his valuable comments.

References

1. R. Agrawal and R. Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *ICDE*, pages 3–14. IEEE Computer Society, 1995.

2. K. R. Apt. *Principles of constraint programming.* Cambridge University Press, 2003.
3. N. Béchet, P. Cellier, T. Charnois, and B. Crémilleux. Discovering linguistic patterns using sequence mining. In *CICLing (1)*, pages 154–165, 2012.
4. N. Béchet, P. Cellier, T. Charnois, and B. Crémilleux. Sequential pattern mining to discover relations between genes and rare diseases. In *IEEE Int. Symp. on Computer-Based Medical Systems (CBMS)*, pages 1–6, 2012.
5. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
6. C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Among, common and disjoint constraints. In *CSCLP*, pages 29–43, 2005.
7. I. V. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model-based clustering. In Raghu Ramakrishnan, Salvatore J. Stolfo, Roberto J. Bayardo, and Ismail Parsa, editors, *KDD*, pages 280–284. ACM, 2000.
8. Chia-Hsiang Chang and Robert Paige. From regular expressions to dfa's using compressed nfa's. *Theor. Comput. Sci.*, 178(1-2):1–36, 1997.
9. E. Coquery, S. Jabbour, L. Saïs, and Yakoub Salhi. A sat-based approach for discovering frequent, closed and maximal patterns in a sequence. In Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, editors, *ECAI*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 258–263. IOS Press, 2012.
10. G. Dong and J. Pei. *Sequence Data Mining*, volume 33 of *Advances in Database Systems.* Kluwer, 2007.
11. Minos N. Garofalakis, R. Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. *IEEE Trans. Knowl. Data Eng.*, 14(3):530–552, 2002.
12. T. Guns, S. Nijssen, and L. De Raedt. Itemset mining: A constraint programming perspective. *Artif. Intell.*, 175(12-13):1951–1983, 2011.
13. T. Guns, S. Nijssen, and L. De Raedt. k-pattern set mining under constraints. *IEEE Trans. Knowl. Data Eng.*, 25(2):402–418, 2013.
14. M. Khiari, P. Boizumault, and B. Crémilleux. Constraint programming for mining n-ary patterns. In David Cohen, editor, *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 552–567. Springer, 2010.
15. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In D. Georgakopoulos and A. Buchmann, editors, *ICDE*, pages 215–224. IEEE Computer Society, 2001.
16. G. Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *CP'04*, volume 2239 of *LNCS*, pages 482–495. Springer, 2004.
17. L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In Ying Li, Bing Liu, and Sunita Sarawagi, editors, *KDD'08*, pages 204–212. ACM, 2008.
18. H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of International Conference on New Methods in Language Processing*, September 1994.
19. R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *EDBT*, volume 1057 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 1996.

20. W. Ugarte, P. Boizumault, S. Loudni, and B. Crémilleux. Soft threshold constraints for pattern mining. In Jean-Gabriel Ganascia, Philippe Lenca, and Jean-Marc Petit, editors, *Discovery Science*, volume 7569 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2012.
21. W. Ugarte, P. Boizumault, S. Loudni, B. Crémilleux, and A. Lepailleur. Découverte des soft-skypatterns avec une approche PPC. In Christel Vrain, André Péninou, and Florence Sèdes, editors, *EGC*, volume RNTI-E-24 of *Revue des Nouvelles Technologies de l'Information*, pages 217–228. Hermann-Éditions, 2013.
22. J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In Z. Meral Özsoyoglu and Stanley B. Zdonik, editors, *ICDE*, pages 79–90. IEEE Computer Society, 2004.
23. X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large databases. In Daniel Barbará and Chandrika Kamath, editors, *SDM*. SIAM, 2003.
24. M. J. Zaki. Sequence mining in categorical domains: Incorporating constraints. In *CIKM*, pages 422–429. ACM, 2000.
25. M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.

The representation of sequential patterns and their projections within Formal Concept Analysis

Aleksey Buzmakov[1,2], Elias Egho[1], Nicolas Jay[1], Sergei O. Kuznetsov[2], Amedeo Napoli[1], and Chedy Raïssi[1]

[1] LORIA (CNRS – Inria Nancy Grand Est – Université de Lorraine) Campus Scientifique, B.P. 70239, Vandœuvre-lès-Nancy, France
[2] Higher School of Economics – National Research University Pokrovskiy Bd. 11 – 109028 Moscow – Russia
{aleksey.buzmakov, elias.egho, nicolas.jay, napoli, chedy.raissi}@inria.fr, skuznetsov@hse.ru

Abstract. Nowadays data sets are available in very complex and heterogeneous ways. The mining of such data collections is essential to support many real-world applications ranging from healthcare to marketing. In this work, we focus on the analysis of *"complex"* sequential data by means of interesting sequential patterns. We approach the problem using an elegant mathematical framework: Formal Concept Analysis (FCA) and its extension based on *"pattern structures"*. Pattern structures are used for mining complex data (such as sequences or graphs) and are based on a subsumption operation, which in our case is defined with respect to the partial order on sequences. We show how pattern structures along with projections (i.e., a data reduction of sequential structures), are able to enumerate more meaningful patterns and increase the computing efficiency of the approach. Finally, we show the applicability of the presented method for discovering and analyzing interesting patients' patterns from a French healthcare data set of cancer patients. The quantitative and qualitative results are reported in this use case which is the main motivation for this work.

Keywords: formal concept analysis, pattern structures, sequential pattern structures, sequences

Introduction

Sequence data is largely present and used in many applications. Consequently, mining sequential patterns from sequence data has become an important and crucial data mining task. In the last two decades, the main emphasis has been on developing efficient mining algorithms and effective pattern representations [1–5]. However, the problem with traditional sequential pattern mining algorithms (and generally with all pattern enumeration algorithms) is that they generate a large number of frequent sequences while few of them are truly relevant. To echo this challenge, some recent studies try to enumerate patterns using some alternative interestingness measures or by sampling representative patterns. A general idea, which is a framework of finding *statistically significant patterns*, is to extract patterns whose characteristic on a given measure, such as frequency, strongly deviates from its expected value under a null model. In this work, we focus on complementing the statistical approaches with a sound and adequate algebraic approach. That is, *can we develop a framework for enumerating only patterns of required types based solely on data lattices and its associated measures?*

The above question can be answered by addressing the problem of analyzing sequential data with the formal concept analysis framework (FCA), an elegant mathematical approach to data analysis [6], and pattern structures, an extension of FCA that handles complex data [7]. To analyze a dataset of "complex" sequences while avoiding the classical efficiency bottlenecks, we introduce and explain the usage of projections which are mathematical functions that respect certain algebraic properties. This novel usage of projections for sequences allows one to reduce the computational costs and the volume of enumerated patterns, avoiding thus the infamous "pattern flooding". In addition, we provide and discuss several measures to rank patterns with respect to their "interestingness", giving the order in which the patterns may be efficiently analyzed.

In this paper, we develop a novel, rigorous and efficient approach for working with sequential pattern structures in formal concept analysis. The main contributions of this work can be summarized as follows:

Pattern structure specification and analysis. We propose a novel way of dealing with sequences based on complex alphabets by mapping them to pattern structures. The genericity power provided by the pattern structures allows our approach to be directly instantiated with state-of-the-art FCA algorithms, making the final implementation flexible, accurate and scalable.

Projections of Sequential Pattern Structures. We introduce and discuss the notion of "projections" for sequential pattern structures. These mathematical objects significantly decrease (i.e., filter) the number of patterns, while preserving the most interesting ones for an expert. Projections are easily built to answer questions that an expert may have. Moreover, combinations of projections and concept stability index provide an efficient tool for the analysis of complex sequential datasets. The second advantage of projections is its ability to significantly decrease the complexity of a problem, saving thus computational time.

Experimental evaluations. We evaluate our approach on real sequence dataset of a regional healthcare system. The data set contains ordered sets of hospitalizations for cancer patients with information about the hospitals they visited, causes for the hospitalizations and medical procedures. These ordered sets are considered as sequences. The experiments reveal interesting (from a medical point of view) and useful patterns, and show the feasibility and the efficiency of our approach.

The paper is organized as follows. Section 1 introduces formal concept analysis and pattern structures. The specification of pattern structures for the case of sequences is presented in Section 2. Section 3 describes projections of sequential pattern structures followed in Section 4.1 by the evaluation and experimations. Finally, related works are discussed before concluding the paper.

1 FCA and Pattern Structures

1.1 Formal Concept Analysis

FCA [6] is a formalism for data analysis. FCA starts with a formal context and builds a set of formal concepts organized within a concept lattice. A formal context is a triple (G, M, I), where G is a set of objects, M is a set of attributes and I is a relation between G and M, $I \subseteq G \times M$. In Table 1, a formal context is shown. A Galois connection between G and M is defined as follows:

$$A' = \{m \in M \mid \forall g \in A, (g, m) \in I\}, \qquad A \subseteq G$$
$$B' = \{g \in A \mid \forall m \in M, (g, m) \in I\}, \qquad B \subseteq M$$

The Galois connection maps a set of objects to the maximal set of attributes shared by all objects and reciprocally. For example, $\{g_1, g_2\}' = \{m_4\}$, while $\{m_4\}' = \{g_1, g_2, g_4\}$.

	m_1	m_2	m_3	m_4
g_1	x			x
g_2			x	x
g_3		x		
g_4			x	x

Table 1: A toy FCA context.

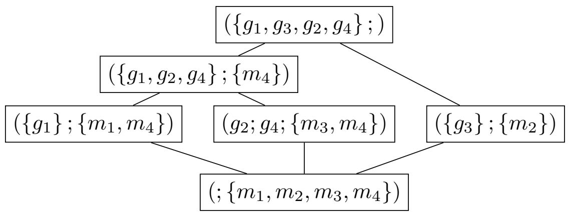

$(\{g_1, g_3, g_2, g_4\}\,;)$
$(\{g_1, g_2, g_4\}\,;\{m_4\})$
$(\{g_1\}\,;\{m_1, m_4\})$
$(g_2; g_4; \{m_3, m_4\})$
$(\{g_3\}\,;\{m_2\})$
$(;\{m_1, m_2, m_3, m_4\})$

Fig. 1: Concept Lattice for the toy context

A formal concept is a pair (A, B), where A is a subset of objects, B is a subset of attributes, such that $A' = B$ and $A = B'$, where A is called the extent of the concept, and B is called the intent of the concept. A formal concept corresponds to a pair of maximal sets of objects and attributes, i.e. it is not possible to add an object or an attribute to the concept without violating the maximality property. For example a pair $(\{g_1, g_2, g_4\}, \{m_4\})$ is a formal concept.

Formal concepts can be partially ordered w.r.t. the extent inclusion (dually, intent inclusion). For example, $(\{g_1\}\,;\{m_1, m_4\}) \leq (\{g_1, g_2, g_4\}, \{m_4\})$. This partial order of concepts is shown in Figure 1.

1.2 Stability Index of a Concept

The number of concepts in a lattice for real-world tasks can be considerable. To find the most interesting subset of concepts, different measures can be used such as the stability of the concept [8] or the concept probability and separation [9]. These measures helps extracting the most interesting concepts and were shown to be reliable in noisy data [9].

Definition 1. *Given a concept c, the concept stability $Stab(c)$ is the number of subsets of the concept extent (denoted $Ext(c)$), whose description is equal to the concept intent (denoted $Int(c)$). Hereafter $\wp(P)$ is a powerset of P.*

$$Stab(c) := \frac{|\{s \in \wp(Ext(c)) \mid s' = Int(c)\}|}{|\wp(Ext(c))|} \tag{1}$$

Stability measures how much the concept depends on the initial dataset. The bigger the stability the more objects can be deleted from the context without affecting the intent of the concept, i.e. the intent of the most stable concepts are likely to be a characteristic pattern of the studied data set.

To the best of our knowledge the fastest algorithm [10] processes a concept lattice L, in the worse case, in $O(|L|^2)$ where $|L|$ is the size of the concept lattice. For a big lattice, the stability calculation time can be high, and an estimation of the stability is useful. It should be noted that in a lattice the extent of any parent of a concept c is a superset of the extent of c, while the extent of any child is a subset. Given a concept c and its child, $\forall s \subseteq Ext(child), s'' \subseteq Ext(child) \subset Ext(c)$, i.e. $s' \neq Int(c)$. Thus, any subset of any child of the concept c should be excluded from the numerator in Equation 1.

$$Stab(c) \leq 1 - \max_{ch \in Children}(2^{-Diff(c,ch)}), \tag{2}$$

where $Diff(c, ch)$ is the extent difference between concept c and its child ch, $Diff(c, child) = |c.Ext \setminus child.Ext|$. Thus, if we would like to find stable concepts, with stability more than 0.97, we should select among concepts with

$$\max_{ch \in Children}(Diff(c,ch)) \geq -\log(1 - 0.97) = 5.06. \tag{3}$$

1.3 Pattern Structures

Although FCA applies to binary context, more complex data such as sequences or graphs can be directly processed as well. For that, pattern structures were introduced in [7].

Definition 2. *A pattern structure is a triple $(G, (D, \sqcap), \delta)$, where G is a set of objects, $(D, \sqcap)$ is a complete meet-semilattice of descriptions and $\delta : G \to D$ maps an object to a description.*

The lattice operation in the semilattice ($\sqcap$) corresponds to the similarity between two descriptions. Standard FCA can be presented in terms of a pattern structure. In this case, G is the set of objects, the semilattice of descriptions is $(\wp(M), \sqcap)$ and a description is a set of attributes, with the $\sqcap$ operation corresponding to the set intersection. If $x = \{a, b, c\}$ and $y = \{a, c, d\}$ then $x \sqcap y = x \cap y = \{a, c\}$. The mapping $\delta : G \to \wp(M)$ is given by, $\delta(g) = \{m \in M \mid (g, m) \in I\}$, and returns the description for a given object as a set of attributes.

The Galois connection between $\wp(G)$ and D is defined as follows:

$$A^{\diamond} := \bigsqcap_{g \in A} \delta(g), \qquad \text{for } A \subseteq G$$

$$d^{\diamond} := \{g \in G \mid d \sqsubseteq \delta(g)\}, \qquad \text{for } d \in D$$

The Galois connection makes a correspondence between sets of objects and descriptions. Given a set of objects A, $A^{\diamond}$ returns the description which is common to all objects in A. And given a description d, $d^{\diamond}$ is the set of all objects whose description subsumes d. More precisely, the partial order (or the subsumption order) on D ($\sqsubseteq$) is defined w.r.t. the similarity operation $\sqcap$: $c \sqsubseteq d \Leftrightarrow c \sqcap d = c$, and c is subsumed by d.

Definition 3. *A pattern concept of a pattern structure $(G, (D, \sqcap), \delta)$ is a pair (A, d) where $A \subseteq G$ and $d \in D$ such that $A^{\diamond} = d$ and $d^{\diamond} = A$, A is called the concept extent and d is called the concept intent.*

A pattern concept corresponds to the maximal set of objects A whose description subsumes the description d, where d is the maximal common description for objects in A. The set of all concepts can be partially ordered w.r.t. partial order on extents (dually, intent patterns, i.e $\sqsubseteq$), within a concept lattice. The stability of a pattern concept can be defined or estimated by the same procedure as for a formal concept, since the stability only depends on extents.

An example of pattern structures is given in Table 2, while the corresponding lattice is depicted in Figure 2.

2 Sequential Pattern Structures

2.1 An Example of Sequential Data

Patient	Trajectory
p^1	$\langle [H_1, \{a\}]; [H_1, \{c, d\}]; [H_1, \{a, b\}]; [H_1, \{d\}] \rangle$
p^2	$\langle [H_2, \{c, d\}]; [H_3, \{b, d\}]; [H_3, \{a, d\}] \rangle$
p^3	$\langle [H_4, \{c, d\}]; [H_4, \{b\}]; [H_4, \{a\}]; [H_4, \{a, d\}] \rangle$

Table 2: Toy sequential data on patient medical trajectories.

Imagine that we have medical trajectories of patients, i.e. sequences of hospitalizations, where every hospitalization is described by a hospital name and a set of procedures. An example of sequential data on medical trajectories with three patients is given in Table 2. There are a set of procedures $P = \{a, b, c, d\}$ a set of hospital names $T_H = \{H_1, H_2, H_3, H_4, CL, CH, *\}$, where hospital names are hierarchically organized (by level of generality), H_1 and H_2 are central hospitals (CH) and H_3 and H_4 are clinics (CL), and $*$ denotes the root of this hierarchy. The least common ancestor in this hierarchy is denoted as $h_1 \sqcap h_2$, for any $h_1, h_2 \in T_H$, i.e. $H_1 \sqcap H_2 = CH$. Every hospitalization is described with one hospital name and may contain several procedures. The procedure order in each hospitalization is not important. For example, the first hospitalization $[H_2, \{c, d\}]$ for the second patient (p^2) was in hospital H_2 and during this hospitalization patient underwent procedures c and d. An important task is to find the "characteristic" sequences of procedures and associated hospitals in order to improve hospitalization planning, optimize clinical processes or detect anomalies.

The search for characteristic sequences can be performed by finding the most stable concepts in the lattice corresponding to a sequential pattern structure. For the simplification of calculations, subsequences are considered without "gaps", i.e the order of non consequent elements is not taken into account. It is reasonable in this task, because a hospitalization is a rather rare situation in the life of a patient, and, thus, in the most cases a hospitalization has a strong relation to the previous one. Next subsections define partial order on sequences and the corresponding pattern structures.

2.2 Partial Order on Complex Sequences

A sequence is constituted of elements from an alphabet. The classical subsequence matching task requires no special properties of the alphabet. Several generalization of the classical case were made by introducing subsequence relation based on itemset alphabet [11] or on multidimensional and multilevel alphabet [12]. Here, we generalize the previous cases, requiring for an alphabet to form a semilattice $(E, \sqcap_E)$[3]. This generalization allows one to process in a unified way all types of complex sequential data.

Definition 4. *Given an alphabet lattice* $(E, \sqcap_E)$*,*

1. $\langle\rangle$ *is a sequence;*
2. *for any sequence* $s = \langle e_1; ...; e_n\rangle$ *and any element* $e \in E$*,* $s \circ e = \langle e_1; ...; e_n; e\rangle$ *is a sequence.*

Definition 5. *A sequence* $t = \langle t_1; ...; t_k\rangle$ *is a subsequence of a sequence* $s = \langle s_1; ...; s_n\rangle$*, denoted* $t \leq s$*, iff* $k \leq n$ *and there exists* $j_1, ..j_k$ *such that* $1 \leq j_1 < j_2 < ... < j_k \leq n$ *and for all* $i \in \{1, 2, ..., k\}$*,* $t_i \sqsubseteq_E s_{j_i}$*.*

With complex sequences and such kind of subsequences the computation can be hard. Thus, for the sake of simplification, only "restricted" subsequences are considered, where only the order of consequent elements is taken into account, i.e. given j_1 in Definition 5, $j_i = j_{i-1} + 1$ for all $i \in \{2, 3, ..., k\}$. Below the word "subsequence" refers to "restricted" subsequence if not specified otherwise.

In the running example (Section 2.1), the alphabet is $E = T_H \times \wp(P)$ with the similarity operation $(h_1, P_1) \sqcap (h_2, P_2) = (h_1 \sqcap h_2, P_1 \cap P_2)$, where $h_1, h_2 \in T_H$ are hospitals and $P_1, P_2 \in \wp(P)$ are sets of procedures. Thus, the sequence ss^1 in

[3] It should be noted that in this paper we consider two semilattices, the first one is on the characters of the alphabet, $(E, \sqcap_E)$, and the second one is introduced by pattern structures, $(D, \sqcap)$.

Table 3 is a subsequence of p^1 in Table 2 because if we set $j_i = i+1$ (Definition 5) then $ss_1^1 \sqsubseteq p_{j_1}^1$ ('CH' is more general than H_1 and $\{c,d\} \subseteq \{c,d\}$), $ss_2^1 \sqsubseteq p_{j_2}^1$ (the same hospital and $\{b\} \subseteq \{b,a\}$) and $ss_3^1 \sqsubseteq p_{j_3}^1$ ('*' is more general than H_1 and $\{d\} \subseteq \{d\}$).

2.3 Sequential Meet-semilattice

Now, we can precisely define the sequential pattern structure that is used for representing and managing sequences. For that, we make an analogy with the pattern structures for graphs [13] where the meet-semilattice operation $\sqcap$ respects subgraph isomorphism. Thus, we introduce a sequential meet-semilattice respecting subsequence relation. Given an alphabet lattice $(E, \sqcap_E)$, $\mathfrak{S}$ is the set of all sequences based on $(E, \sqcap_E)$. $\mathfrak{S}$ is partially ordered w.r.t. Definition 5. $(D, \sqcap)$ is a semilattice on sequences $\mathfrak{S}$, where $D \subseteq \wp(\mathfrak{S})$ such that if $d \in D$ contains a sequence s then all subsequences of s should be included into d, $\forall s \in d, \nexists \tilde{s} \leq s : \tilde{s} \notin d$, and similarity operation is the set intersection for two set of sequences. Given two patterns $d_1, d_2 \in D$, the set intersection operation ensures that if a sequence s belongs to $d_1 \sqcap d_2$ then any subsequence of s belongs to $d_1 \sqcap d_2$ and thus $(d_1 \sqcap d_2) \in D$. As the set intersection operation is idempotent, commutative and associative, $(D, \sqcap)$ is a valid semilattice.

However, the set of all possible subsequence for a given sequence can be rather large. Thus, it is more efficient and representable to keep a pattern $d \in D$ as a set of all maximal sequences $\tilde{d}$, $\tilde{d} = \{s \in d \mid \nexists s^* \in d : s^* \geq s\}$. Furthermore, every pattern will be given only by the set of all maximal sequences. For example, $\{p^2\} \sqcap \{p^3\} = \{ss^6, ss^7, ss^8\}$ (see Tables 2 and 3), i.e. $\{ss^6, ss^7, ss^8\}$ is the set of all maximal sequences specifying the intersection result of two sets of sequences specified by sequences p^2 and p^3, correspondingly $\{ss^6, ss^7, ss^8\} \sqcap \{p^1\} = \{ss^4, ss^5\}$. Note that representing a pattern by the set of all maximal sequences allows for an efficient implementation of the intersection "$\sqcap$" of two patterns (see Section 4.1 for more details).

Example 1. The sequential pattern structure for our example (Subsection 2.1) is $(G, (D, \sqcap), \delta)$, where $G = \{p^1, p^2, p^3\}$, $(D, \sqcap)$ is the semilattice of sequential descriptions, and δ is the mapping associating an object in G to a description in D shown in Table 2. Figure 2 shows the resulting lattice of sequential pattern concepts for this particular pattern structure $(G, (D, \sqcap), \delta)$.

3 Projections of Sequential Pattern Structures

Pattern structures can be hard to process due to the usually large number of concepts in the concept lattice, the complexity of the involved descriptions and the similarity operation. Moreover, a given pattern structure can produce a lattice with a lot of patterns which are not interesting for an expert. *Can we save computational time by avoiding to compute useless patterns?* Projections of pattern structures "simplify" to some degree the computation and allow one to work

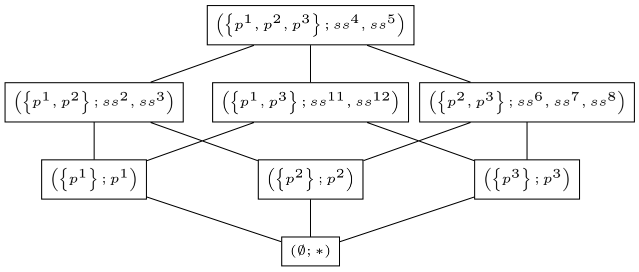

$(\{p^1,p^2,p^3\};ss^4,ss^5)$
$(\{p^1,p^2\};ss^2,ss^3)$
$(\{p^1,p^3\};ss^{11},ss^{12})$
$(\{p^2,p^3\};ss^6,ss^7,ss^8)$
$(\{p^1\};p^1)$
$(\{p^2\};p^2)$
$(\{p^3\};p^3)$
$(\emptyset;*)$

Fig. 2: The concept lattice for the pattern structure given by Table 2. Concept intents reference to sequences in Tables 2 and 3.

	Subsequences		Subsequences
ss^1	$\langle[CH,\{c,d\}];[H_1,\{b\}];[*,\{d\}]\rangle$	ss^2	$\langle[CH,\{c,d\}];[*,\{b\}];[*,\{d\}]\rangle$
ss^3	$\langle[CH,\{\}];[*,\{d\}];[*,\{a\}]\rangle$	ss^4	$\langle[*,\{c,d\}];[*,\{b\}]\rangle$
ss^5	$\langle[*,\{a\}]\rangle$	ss^6	$\langle[*,\{c,d\}];[CL,\{b\}];[CL,\{a\}]\rangle$
ss^7	$\langle[CL,\{d\}];[CL,\{\}]\rangle$	ss^8	$\langle[CL,\{\}];[CL,\{a,d\}]\rangle$
ss^9	$\langle[CH,\{c,d\}]\rangle$	ss^{10}	$\langle[CL,\{b\}];[CL,\{a\}]\rangle$
ss^{11}	$\langle[*,\{c,d\}];[*,\{b\}]\rangle$	ss^{12}	$\langle[*,\{a\}];[*,\{d\}]\rangle$

Table 3: Subsequences of patient sequences in Table 2.

with a reduced description. In fact, projections can be considered as filters on patterns respecting mathematical properties. These properties ensure that the projection of a semilattice is a semilattice and that projected concepts have a correspondence to original ones. Moreover, the stability measure of projected concepts never decreases w.r.t the original concepts [7].

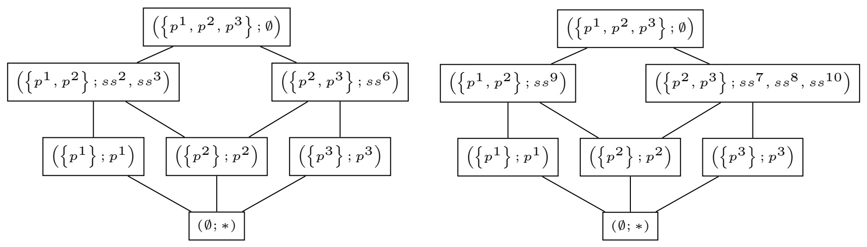
A possible projection for sequential pattern structures comes from the following observation. In many cases it may be more interesting to analyze long subsequences. We call these projections *Minimal Length Projection* (MLP) and they depend on the minimal allowed length l for the sequences in a pattern. To project a pattern structure w.r.t. MLP, a pattern should be substituted with the pattern where any sequence of length less then l is removed.

Example 2. If we set the minimal length threshold to 3, then there is only one maximal common subsequence ss^6 in Table 3 between p^2 and p^3 in Table 2, while ss^7 and ss^8 are too short to be considered. Figure 3a shows the corresponding projected lattice for the pattern structure in Table 2.

Another important type of projections is connected to a variation of the lattice alphabet $(E, \sqcap_E)$. The simplest variation is to ignore of certain fields in the elements. For example, if a hospitalization is described by a hospital name and a set of procedures, then procedures can be ignored in similarity computation. For that, in any element a set of procedures can be substituted by $*$ which is the most general element of the taxonomy of hospitals.

Another variation of the alphabet, is to require that some field(s) should not be empty. For example, we want to find patterns with non-empty set of procedures, or we want to have information about hospital (the element $*$ of hospital taxonomy is not allowed in an element of a sequence). Such variations are easy to realize within our approach. For this, computing the similarity operation between elements of the alphabet, one should check if the result contains empty fields and, if yes, should substitute the result by $\perp$. This variation is useful, as shown in the experimental section, but this variation is rather difficult to define within classical frequent sequence mining approaches.

Example 3. An expert is interested in finding sequential patterns describing how a patient changes hospitals, without interest in procedures. Thus, any element of the alphabet lattice containing a non-empty set of procedures is projected to the corresponding element with the same hospital and an empty set of procedures. Moreover, an expert is interested in finding sequential patterns containing information about the hospital in every hospitalization, i.e. hospital field in the patterns cannot be $*$, e.g. ss^5 is an invalid pattern, while ss^6 is a valid pattern in Table 3. Figure 3b shows the lattice corresponding to the projected pattern structure (Table 2) by changing the alphabet semilattice.



(a) MLP projection, $l = 3$ (b) Projection removing '*' hospitals

Fig. 3: The projected concept lattices for the pattern structure given by Table 2. Concept intents refer to the sequences in Tables 2 and 3.

4 Sequential Pattern Structure Evaluation

4.1 Implementation

Nearly any state-of-the-art FCA algorithm can be adapted to process pattern structures instead of standard FCA contexts. We adapted `AddIntent` algorithm [14], as the lattice structure is important for us to calculate stability (see the algorithm for calculating stability in [10]). To adapt the algorithm to our needs, every set intersection operation on attributes should be substituted with semilattice operation $\sqcap$ on corresponding patterns, while every subset checking operation should be substituted with semilattice order checking $\sqsubseteq$, in particular all $(\cdot)'$ should be substituted with $(\cdot)^\diamond$.

The next question is how the semilattice operations ($\sqcap$, $\sqsubseteq$) can be implemented. Given two sets of sequences $S = \{s^1, ...s^n\}$ and $T = \{t^1, ..., t^m\}$, the similarity between these sets, $S \sqcap T$, is calculated according to Section 2.3, i.e. maximal sequences among all common subsequences for any pair of s^i and t^j.

To find all common subsequences of two sequences, the following observations can be useful. If $ss = \langle ss_1; ...; ss_l \rangle$ is a subsequence of $s = \langle s_1; ...; s_n \rangle$ with

$j_i^s = k^s + i$ (Definition 5: k^s is the index difference from which ss is a subsequence of s) and a subsequence of $t = \langle t_1; ...; t_m \rangle$ with $j_i^t = k^t + i$ (likewise), then for any index $i \in \{1, 2, ..., l\}$, $ss_i \sqsubseteq_E (s_{j_i^s} \sqcap t_{j_i^t})$. Thus, to find all maximal common subsequences between s and t, we first align s and t in all possible ways. For each alignment of s and t we compute the resulting intersection. Finally, we keep only the maximal intersected subsequences.

Let us consider two possible alignments of s^1 and s^2:

$$\begin{array}{lllllll|lllllll}
s^1 = & \langle \{a\}; & \{c,d\}; & \{b,a\}; & \{d\} & & \rangle & s^1 = & \langle \{a\}; & \{c,d\}; & \{b,a\}; & \{d\} & \rangle \\
s^2 = & & & \langle \{c,d\}; & \{b,d\}; & \{a,d\} & \rangle & s^2 = & & \langle \{c,d\}; & \{b,d\}; & \{a,d\} & \rangle \\
ss^l = & & & \langle \ \emptyset \ ; & \{d\} & & \rangle & ss^r = & & \langle \{c,d\}; & \{b\}; & \{d\} & \rangle
\end{array}$$

The left intersection ss^l is not retained, as it is not maximal, while the right intersection ss^r is kept.

4.2 Experiments and Discussion

The experiments are carried out on an "Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz" computer with 8Gb of memory under the Ubuntu 12.04 operating system. The algorithms are not parallelized and are coded in C++.

First, the public available database from UCI repository on anonymous web data is used as a benchmark data set for scalability tests. This database contains around 10^6 transactions, and each transaction is a sequence based on "simple" alphabet, i.e. with no order on the elements. The overall time changes from 37279 seconds for the sequences of length $MLP \geq 5$ upto 97042 seconds for the sequences of length $MLP \geq 3$. For more details see the web-page.[4]

Our use-case data set comes from PMSI[5], a French healthcare system [15]. Each elements of a sequence has a "complex" nature. The dataset contains 2400 patients suffering from *cancer*. Every patient is described as a sequence of hospitalizations without any timestamps. The hospitalization is a tuple with three elements: (i) healthcare institution (e.g. university hospital of Paris (CHU_{Paris})), (ii) reason of the hospitalization (e.g. a cancer disease), and (iii) set of medical procedures that the patient underwent. An example of a medical trajectory of a patient is provided below:

$$\langle [\text{CHU}_{Paris}, \text{Cancer}, \{P_1, P_2\}]; [\text{CH}_{Lyon}, \text{Chemo}, \{\}]; [\text{CH}_{Lyon}, \text{Chemo}, \{\}] \rangle \, .$$

.This sequence represents a patient trajectory with three hospitalizations. It expresses that one patient was first admitted to the university hospital of Paris (CHU_{Paris}) for a cancer problem as reason, and underwent procedures P_1 and P_2. Then he had two consequent hospitalizations in Central hospital of Lyon (CH_{Lyon}) for doing chemotherapy with no additional procedures. We substituted the same consequent hospitalizations by the number of repetitions. With this substitution, we have shorter and more understandable trajectory. For example, the above pattern should be transformed into two hospitalizations where the first hospitalization repeats once and the second twice:

$$\langle [\text{CHU}_{Paris}, \text{Cancer}, \{P_1, P_2\}][1]; [\text{CH}_{Lyon}, \text{Chemo}, \{\}][2] \rangle \, .$$

[4] `http://www.loria.fr/~abuzmako/PKDD2013/experiment-uci.html`

[5] Programme de Médicalisation des Sytèmes d'Information.

The healthcare institution was associated with a geographical taxonomy of 4 levels of granularity (i.e. Root, Region, Department and Healthcare institution). This taxonomy has 304 node. Where hospitalization reasons and medical procedures are simple sets without any associated subsumption relation. The set of hospitalisation reasons has 1939 items and the set of medical procedures has 723 items. The distribution of sequence lengths' is shown in Figure 4.

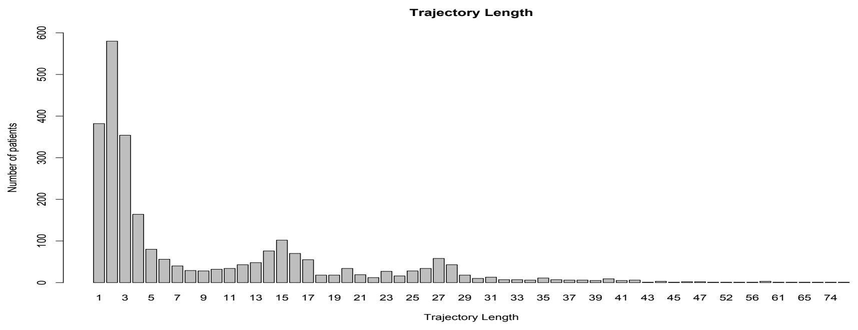


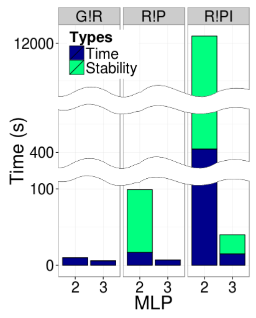
Fig. 4: The length distribution of sequences in the dataset

For this dataset the computation of the whole lattice is infeasible. However our medical expert is not interested in all possible patterns, but rather in patterns which answer his analysis question(s). First of all, an expert may know the minimal size of sequences he is interested in, i.e. setting the MLP projection. If an expert is interested in sequential patterns, the patterns of length 1 are unlikely to be of interest for him (knowing that people go to hospitals when they are sick is not a valuable new knowledge). Thus, we use the MLP projection of length 2 and 3 and take into account the small average length of the sequences.

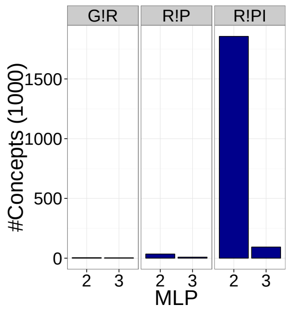
Figure 5 shows computational time, the number of concepts in the lattice, and the number of stable concepts for different projections. For example, computation of the lattice for projection with name "`R!PI`" takes 400 seconds and calculation of stability for every concept in the lattice takes 12000 seconds (Figure 5a), the size of the lattice is $1.8 \cdot 10^6$ concepts (Figure 5b) where around 1000 concepts have stability index more than 0.97 while an approximated solution to find stable concepts (Formula 3) return only few unstable ones (Figure 5c).

Table 4 shows some interesting concept intents with the corresponding support and ranking w.r.t. to concept stability. For example the concept #1 is obtained under the projection `R!P` for $MLP \geq 2$, with the intent containing a *Cancer* hospitalization followed by a *Chemotherapy.* This concept is the most stable concept in the lattice for the given projection, and the cardinality of the concept extent is 452 patients.

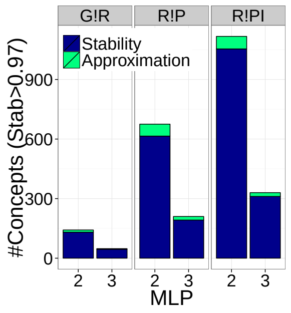
The first question that the analyst would like to address here is *"What are the sequential relations between hospitalization reasons and corresponding procedures?"*. To answer this question, we are not interested about healthcare institutions. Thus, any alphabet element should be projected by substituting healthcare institution fields by the '*' hospital. As hospitalization reason is important in



(a) Calculation time



(b) Concepts



(c) Stable concepts

Fig. 5: Parameters of the result for different projections.

#	Projection	Intent	Stab. Rank	Support
1	`R!P2`	$\langle[Cancer, \{\}]; [Chemo, \{\}]\rangle$	1	452
2	`R!P2`	$\langle[Cancer, \{App.\}]; [Ch.Prep, \{\}]; [Chemo, \{\}]\rangle$	4	293
3	`R!P3`	$\langle[Cancer, \{App.\}]; [Ch.Prep, \{\}]; [Chemo, \{\}]\rangle$	2	293
4	`R!PI3`	$\langle[Cancer, \{\}] * 1; [Ch.Prep, \{\}] * 1; [Chemo, \{\}] * [8, 24]\rangle$	4	193
5	`G!R!3`	$\langle[Bourgogne, Cancer]; [Bourgogne, Ch.Prep]; [A\ clinic\ in\ Dijon, Chemo]\rangle$	5	29

Table 4: Interesting concepts, for different projections. `Chemo` is chemotherapy, `Ch.Prep` is preparation for chemotherapy, `App.` is an operation for appendicitis.

each hospitalization so any alphabet element without the hospitalization reason is of no use and should be projected to the bottom element $\perp$ of the alphabet lattice. This is a projection of the hospitalization alphabet and, thus, gives us the projection of the pattern structure. Such projections are called `R!P2` or `R!P3`, meaning that we consider the fields "Reason" and "Procedures", while the reason should not be empty and the MLP parameter is 2 or 3. *Patterns #1 and #2* are obtained under the `R!P2` projection. *Pattern #1* trivially states that in the Bourgogne region, *"When a patient has a cancer, he undergoes chemotherapy"* which is one of the standard procedure followed by french physicians. This pattern gives a general viewpoint about the cancer treatment.

The next accurate question is *"How do the doctors detect colon cancer?"*. *Pattern #2 and #3* answer our question, they show that cancer is detected during an appendicitis surgical intervention which is followed by preparation for chemotherapy and chemotherapy itself. These two patterns highlight a recently discovered fact that acute appendicitis has been shown to occur antecedent to cancer [16] within three years because of a carcinoma in colon or rectum. Therefore, any patient over the age of 40 presenting with acute appendicitis is carefully checked for carcinoma in the colon. We can also note that *patterns #2 and #3* have the same form, but pattern #3 was obtained under `R!P3` projection, and has higher stability rank (2) than pattern #2 (4). *Pattern #4* can help healthcare managers and doctors quantify on average the number of usually required chemotherapies for a patient. It shows that *"After detecting cancer, the patients require chemotherapy between 8 and 24 times in many cases"*. This pattern has been extracted by the projection `R!PI3` (i.e. involving interval information). Figure 5a and 5b shows that this task is time and memory expensive.

"Where do patients prefer staying (i.e. hospital location) during their treatment, and why ?". To answer this expert question, we consider only healthcare institutions and reason fields, requiring both to "have" some information, i.e. projections `G!R!2` and `G!R!3`. Nearly all patterns show that patients usually prefer to be treated in the same region, without any preferences about the exact hospital. However, *pattern #5* obtained under `G!R!3` projection shows us that a good proportion of patients prefer to undergo Chemotherapy in *a precise private clinic in Dijon*[6], while cancer detection and preparation is usually done everywhere in the Bourgogne region, depending on the patient preferences.

Figure 5 shows that with the increase of the minimal length of a pattern (from 2 to 3), the memory and the time consumption is reduced, in some cases significantly. Figure 5a shows that the precise stability calculation can take more time than the calculation of the lattice, correspondingly the lattice computation for projection `R!PI2` takes 400 seconds, while the stability calculation procedure takes 30 times more (12000). However, the approximation of concept stability that is presented in the beginning of the paper (Formula 3) is fast and does filter only few unstable concepts (less then 5%), while finding all stable (Figure 5c).

5 Related Work

The most widely used approach for analyzing sequences is, probably, mining frequent subsequences [2–4, 12]. The most general type of sequences among them is described in [12], where every element of the sequence is multidimensional and multilevel, i.e. every element can be characterized by several components, and for every component a kind of hierarchy can be applied. Then, every element e in a sequences is substituted by all the most specific elements, which are more general than e and, thus, the task is reduced to sequences of itemsets. In our approach, the elements of sequences are considered to be even more general, for example, beside multidimensional and multilevel sequences, sequences of graphs fall under the definition. Moreover, frequent subsequences mining gives birth to a lot of subsequences which can be hardly analyzed by an expert.

Formal Concept Analysis (FCA) [6] allows one to measure several indexes, related to the importance of a pattern. One of the FCA approaches is [17], where authors process sequential dataset based on "simple" alphabet without involving any partial order on it, in this approach maximal common subsequences (with no gaps) were mined and analyzed with FCA. In the work [11] only sequences of itemsets were considered. All closed subsequences were, first, mined and then regrouped by specialized algorithm in order to obtain a lattice similar to the FCA lattice. Comparing with both approaches, our approach suggests a more general definition of sequences and, thanks to pattern structures, there is no 'premining' step to find frequent (or maximal) subsequences. This allows us to apply different "projections" specializing the request of an expert and simplifying

[6] the name of the clinic is anonymized.

the calculation. In addition, in our approach nearly all state-of-the-art FCA algorithms can be used in order to efficiently process a dataset.

Another type of the FCA generalization is based on well-known LCM algorithm [18]. Authors of [19] process multirelational databases by extending LCM. Although this approach perfectly works for special kinds of multirelational databases, it cannot process sequential datasets for the same reason why it cannot process graph datasets in the settings of frequent graph mining.

Projections is an essential part of our approach and can be considered as a special kind of constraints. Many constraints that do not change subsequence relation have a corresponding projection. Authors of survey [20] (Section 5) enumerate 8 types of constraints, two of them, i.e. "item constraint" and "length constraint", correspond the introduced projections.

Conclusion

In this paper, we present a novel approach for analyzing complex sequential data. This kind of data is a generalization of data considered in previous approaches. The approach is based on the formalism of sequential pattern structures and projections. Our work complements the general orientations towards *statistically significant patterns* by presenting strong formal results on the notion of interestingness from a concept lattice point of view. Using pattern structures leads to the construction of a pattern concept lattice, which does not require the setting of a support threshold, as usually needed in classical sequential pattern mining. Moreover, the use of projections gives a lot of flexibility especially for mining and interpreting special kinds of patterns.

Our framework was tested on a large-scale benchmark dataset and on a real-world dataset with patient hospitalization trajectories. Interesting patterns answering to the questions of an expert are extracted and interpreted, showing the feasibility and usefulness of the approach and the importance of the stability as a pattern-selection procedure.

For future work, we are planning to more deeply investigate projections, their potentialities w.r.t. the types of patterns. Finally, another research direction is mining of association rules or building a Horn approximation [21] from the stable part of the pattern lattice.

Acknowledgements: this research received funding from the Basic Research Program at the National Research University Higher School of Economics (Russia) and from the BioIntelligence project (France).

References

1. Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.: FreeSpan: frequent pattern-projected sequential pattern mining. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. (2000) 355–359

2. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: PrefixSpan Mining Sequential Patterns Efficiently by Prefix Projected Pattern Growth. In: 17th International Conference on Data Engineering. (2001) 215–226
3. Yan, X., Han, J., Afshar, R.: CloSpan: Mining Closed Sequential Patterns in Large Databases. In: In SDM. (2003) 166–177
4. Ding, B., Lo, D., Han, J., Khoo, S.C.: Efficient Mining of Closed Repetitive Gapped Subsequences from a Sequence Database. In: 2009 IEEE 25th International Conference on Data Engineering, IEEE (March 2009) 1024–1035
5. Raïssi, C., Calders, T., Poncelet, P.: Mining conjunctive sequential patterns. Data Min. Knowl. Discov. **17**(1) (2008) 77–93
6. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. 1st edn. Springer, Secaucus, NJ, USA (1997)
7. Ganter, B., Kuznetsov, S.O.: Pattern Structures and Their Projections. In Delugach, H., Stumme, G., eds.: Conceptual Structures: Broadening the Base SE - 10. Volume 2120 of LNCS. Springer Berlin Heidelberg (2001) 129–142
8. Kuznetsov, S.O.: On stability of a formal concept. Annals of Mathematics and Artificial Intelligence **49**(1-4) (2007) 101–115
9. Klimushkin, M., Obiedkov, S.A., Roth, C.: Approaches to the Selection of Relevant Concepts in the Case of Noisy Data. In: Proceedings of the 8th international conference on Formal Concept Analysis. ICFCA'10, Springer (2010) 255–266
10. Roth, C., Obiedkov, S., Kourie, D.G.: On succinct representation of knowledge community taxonomies with formal concept analysis A Formal Concept Analysis Approach in Applied Epistemology. International Journal of Foundations of Computer Science **19**(02) (April 2008) 383–404
11. Casas-Garriga, G.: Summarizing Sequential Data with Closed Partial Orders. In: SDM. (2005)
12. Plantevit, M., Laurent, A., Laurent, D., Teisseire, M., Choong, Y.W.: Mining multidimensional and multilevel sequential patterns. ACM Transactions on Knowledge Discovery from Data **4**(1) (January 2010) 1–37
13. Kuznetsov, S.: Learning of Simple Conceptual Graphs from Positive and Negative Examples. Volume 1704 of LNCS. Springer (1999) 384–391
14. Merwe, D.V.D., Obiedkov, S., Kourie, D.: AddIntent: A new incremental algorithm for constructing concept lattices. In Goos, G., Hartmanis, J., Leeuwen, J., Eklund, P., eds.: Concept Lattices. Volume 2961. Springer (2004) 372–385
15. Fetter, R.B., Shin, Y., Freeman, J.L., Averill, R.F., Thompson, J.D.: Case mix definition by diagnosis-related groups. Med Care **18**(2) (February 1980) 1–53
16. Arnbjörnsson, E.: Acute appendicitis as a sign of a colorectal carcinoma. Journal of Surgical Oncology **20**(1) (May 1982) 17–20
17. Ferré, S.: The Efficient Computation of Complete and Concise Substring Scales with Suffix Trees. In Kuznetsov, S.O., Schmidt, S., eds.: Formal Concept Analysis SE - 7. Volume 4390 of Lecture Notes in Computer Science. Springer (2007) 98–113
18. Uno, T., Asai, T., Uchida, Y., Arimura, H.: An efficient algorithm for enumerating closed patterns in transaction databases. Discovery Science (2004) 16–31
19. Garriga, G., Khardon, R., De Raedt, L.: Mining closed patterns in relational, graph and network data. Ann. Math. Artif. Intell. (November 2012) 1–28
20. Mooney, C.H., Roddick, J.F.: Sequential pattern mining – approaches and algorithms. ACM Computing Surveys **45**(2) (February 2013) 1–39
21. Balcázar, J.L., Casas-Garriga, G.: On Horn Axiomatizations for Sequential Data. In: ICDT. (2005) 215–229

Language of Conclusions and Formal Framework for Data Mining with Association Rules

Jan Rauch

Faculty of Informatics and Statistics, University of Economics, Prague [⋆]

Abstract. FOFRADAR is a formal framework describing a process of data mining with association rules. Its purpose is to serve as a theoretical basis for automation of the data mining process. Association rule is understood as a couple of general Boolean attributes derived from columns of a data matrix and mutually related in an interesting way. FOFRADAR is based on a logical calculus of association rules, which is enhanced by languages and procedures making possible to deal with items of domain knowledge. Items of domain knowledge correspond to general expressions good understandable to domain experts. One of the languages of FOFRADAR is a language formulas which correspond to conclusions we can draw from results of data mining process. New features of this language are presented.

1 Introduction

A formal framework FOFRADAR (FOrmal FRAmework for Data mining with Association Rules) describing a process of data mining with association rules is introduced in [7]. Its goal is to describe the process such that formalized items of domain knowledge can be used both in formulation of reasonable analytical questions and in interpretation of results of a mining procedure. FOFRADAR is assumed to serve as a theoretical basis for the EverMiner project [9, 14].

The goal of the EverMiner project is to study data mining as a permanent knowledge driven process. It is assumed that there is a knowledge repository containing both relevant domain knowledge and hypotheses on new items of knowledge based on results of the analysis. We also assume there are tools that formulate reasonable data mining tasks, search in the analysed data for true patterns relevant to the formulated tasks, filter out found patterns which can be understood as the consequences of items of knowledge stored in the repository, synthesize hypotheses on new items of knowledge from the remaining patterns and store these hypotheses in the knowledge repository.

Let us emphasize that EverMiner is rather a long-term project which can bring interesting partial results. A natural part of the EverMiner project is a study of possibilities of automation of data mining. A formal description of the

[⋆] The work described here has been supported by Grant No. IGA 20/2013 of the University of Economics, Prague.

data mining process is a necessary prerequisite of its automation. We start with mining of association rules which are known patterns used in data mining.

However, we deal with more general association rules than introduced in [1]. The association rule is understood as an expression $\varphi \approx \psi$ where φ and ψ are Boolean attributes derived from columns of analysed data matrices and $\approx$ stands for a condition concerning a contingency table of φ and ψ [10]. Symbol $\approx$ is called *4ft-quantifier*. Boolean attributes are derived from basic Boolean attributes i.e expressions $A(\alpha)$. Here A is an attribute corresponding to a column of an analysed data matrix with possible values (i.e. categories) $a_1, \ldots, a_k$ and α is a subset of the set of categories, $\alpha \subset \{a_1, \ldots, a_k\}$. Basic Boolean attribute $A(\alpha)$ is true in a row o of a given data matrix $\mathcal{M}$ if $A(o) \in \alpha$ where $A(o)$ is a value of attribute A in row o. This means that we do not deal only with Boolean attributes - conjunctions of attribute-value pairs $A(a)$ where $a \in \{a_1, \ldots, a_k\}$ but with general Boolean attributes derived from columns of an analyzed data matrix. The 4ft-Miner procedure mines for such association rules [11].

FOFRADAR is a result of enhancing of a logical calculus of association rules [10] by additional languages and procedures. It is strongly related to the rules of the above introduced form $\varphi \approx \psi$ and to the possibilities of the 4ft-Miner procedure to mine for such rules. The goal of this paper is to present new considerations on language of formulas which correspond to conclusions of a data mining process. Before that main features of FOFRADAR are introduced. We proceed very informally, formal approach is introduced in [7, 8], see also [10].

The structure of the paper is as follows. An overview of FOFRADAR is in section 2. Particular languages and procedures of FOFRADAR are described in sections 3 – 6 in a way introduced in section 2. Language of conclusions is discussed in section 7. Remarks to related works are in section 8.

2 FOFRADAR Overview

FOFRADAR is sketched in Fig. 1 together with relations of its elements to the CRISP-DM. It is a result of an enhancement of a calculus of association rules by

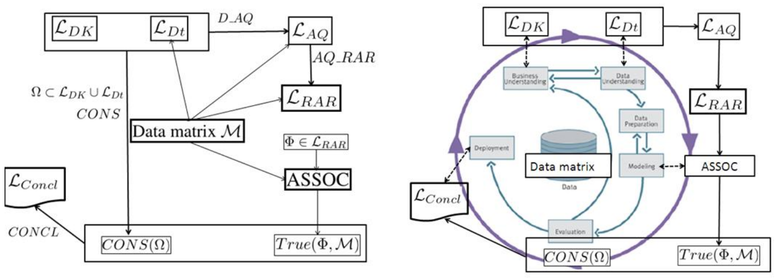


Fig. 1. *FOFRADAR* and CRISP-DM

several languages and procedures. Overview of elements of FOFRADAR follows.

Language $\mathcal{L}_{DK}$ – formulas of this language correspond to items of domain knowledge, they can be considered as results of business understanding. Language $\mathcal{L}_{DK}$ includes SI-formulas expressing mutual influence of attributes. Formula $BMI \uparrow\uparrow Diastolic$ meaning that if Body Mass Index of patients increases then diastolic blood pressure increases too is an example. Additional details are in section 3.

Language $\mathcal{L}_{Dt}$ – formulas of this language correspond to relevant information on analyzed data, see section 3. *Language* $\mathcal{L}_{AQ}$ – formulas of this language correspond to analytical questions. Analytical questions – formulas of $\mathcal{L}_{AQ}$ are formulated using formulas of languages $\mathcal{L}_{DK}$ and $\mathcal{L}_{Dt}$. This can be seen as an application of a *procedure* D_AQ, see section 4.

The core of the data mining process is the procedure ASSOC [3]. Its input consists of an analysed data matrix $\mathcal{M}$ and of a definition of a set of association rules to be verified to solve an analytical question given by a formula Θ of the language $\mathcal{L}_{AQ}$. The set of association rules to be verified is given by a formula Φ of a language $\mathcal{L}_{RAR}$ (i.e. a language of definitions of sets of Relevant Association Rules). This set is denoted as $\mathcal{S}(\Phi)$. We assume that the formula Φ is a result of an application of a procedure AQ_RAR to a given formula Θ, we write $\Phi = AQ\_RAR(\Theta)$. Output of the ASSOC procedure is a set $True(\mathcal{S}(\Phi), \mathcal{M})$ of all rules $\varphi \approx \psi$ which belong to $\mathcal{S}(\Phi)$ and which are true in $\mathcal{M}$. We use the procedure 4ft-Miner as an implementation of the ASSOC, see section 5.

An answer to a given analytical question is based on a comparison of the set $True(\mathcal{S}(\Phi), \mathcal{M})$ and a set $CONS(\Omega)$ of association rules which can be understood as consequences of a set Ω of items of domain or data knowledge used in the formulation of the solved analytical question Θ. An example of a set of consequences of SI-formula $BMI \uparrow\uparrow Diastolic$ is in section 6. Possible conclusion of analysis i.e. formulas of language $\mathcal{L}_{Concl}$ are introduced in section 7.

3 Asociation rules, Domain and Data Knowledge

We use a concrete data matrix to introduce a logical calculus of association rules in a very informal way. This is done in section 3.1, a formal definition is given in [10]. Language $\mathcal{L}_{DK}$ of domain knowledge is introduced in section 3.2. We will not describe the language $\mathcal{L}_{Dt}$ of data knowledge here in more details. Information that given data matrix *Entry* concerns only male patients is an example of an item of data knowledge.

3.1 Calculus $\mathcal{LC}_{\mathcal{E}}$ of Association Rules

We deal with association rules $\varphi \approx \psi$ where φ and ψ are Boolean attributes derived from basic Boolean attributes of the form $A(\alpha)$ and $\approx$ is a 4ft-quantifier. Here A is an attribute corresponding to a column of an analysed data matrix with possible values (i.e. categories) $a_1, \ldots, a_k$ and $\alpha \subset \{a_1, \ldots, a_k\}$. This means that a set of all basic Boolean attributes we can derive from a given column is given by a set of categories for this column. Consequently, a set of all Boolean attributes concerning a given data matrix is determined by sets of possible values for particular columns of this data matrix.

We usually consider data matrices containing only natural numbers. There is only a finite number of possible values for each column. Let us assume that the number of possible values of a column is t and the possible values in this column are integers $1, \ldots, t$. Then all the possible values in a data matrix are described by the number of its columns and by the numbers of possible values for each column. These numbers determine a type of a data matrix and also a type of a logical calculus of association rules.

A *type of a logical calculus of association rules* is a K-tuple $\mathcal{T} = \langle t_1, \ldots, t_K \rangle$ where $K \geq 2$ is an integer and $t_i \geq 2$ are integers for $i = 1, \ldots, K$. A *data matrix of type* $\mathcal{T}$ has columns – attributes $A_1, \ldots, A_K$. Possible values (categories) for attribute A_i are $1, \ldots, t_i$ and *a type of attribute* A_i *is* t_i where $i = 1, \ldots, K$. A language $\mathcal{L}_\mathcal{T}$ of association rules of the type $\mathcal{T}$ is given by the attributes $A_1, \ldots, A_K$ and 4ft-quantifiers $\approx_1, \ldots, \approx_Q$. Association rule of $\mathcal{L}_\mathcal{T}$ is each rule $\varphi \approx \psi$ built from $A_1, \ldots, A_K$ and $\approx_1, \ldots, \approx_Q$. A logical calculus $\mathcal{LC}_\mathcal{T}$ of association rules of a type $\mathcal{T}$ consists of a language $\mathcal{L}_\mathcal{T}$, a set of all data matrices of type $\mathcal{T}$ and of instructions on how to decide if a given association rule is true in a given data matrix.

We use a data matrix *Entry* to introduce an example of a calculus of association rules. *Entry* is a part of the data set STULONG [1]. Data matrix *Entry* concerns 1 417 patients – men that have been examined at the beginning of the study. Each row of *Entry* describes one patient. *Entry* has 64 columns corresponding to particular attributes $A_1, \ldots, A_{64}$ – characteristics of patients. We use only first 12 attributes introduced in Tab. 1. These attributes and their categories have alternative names also introduced in Tab. 1 together with their frequencies in data matrix *Entry*. Types of these 12 attributes are also in Tab. 1. In Table 1, there is also a C-type for these attributes. C-type is introduced in the next section.

We can say that there is a type $\mathcal{T}_\mathcal{E} = \langle 4, 4, 4, 3, 3, 13, 7, 9, 10, 2, 2, 2, t_{13}, \ldots, t_{64} \rangle$ of a logical calculus $\mathcal{LC}_\mathcal{E}$ of association rules. Data matrix *Entry* is a data matrix of the type $\mathcal{T}_\mathcal{E}$ and each its update is also a data matrix of the type $\mathcal{T}_\mathcal{E}$. $A_1, \ldots, A_{64}$ are attributes of language $\mathcal{L}_\mathcal{E}$ of calculus $\mathcal{LC}_\mathcal{E}$. The alternative name of the attribute A_1 is *M_Status*, the alternative names of its categories 1,2,3,4 are *married*, *divorced*,*single*, *widover* respectively; similarly for additional attributes and categories. Let us note that there are missing values and thus the sum of frequencies of categories of particular attributes can be less than 1417.

[1] The study (STULONG) was realized at the 2nd Department of Medicine, 1st Faculty of Medicine of Charles University and University Hospital in Prague, under the supervision of Prof. F. Boudík, MD, DSc., with collaboration of M. Tomečková, MD, PhD and Prof. J. Bultas, MD, PhD. The data were transferred to the electronic form by the European Centre of Medical Informatics, Statistics and Epidemiology of Charles University and Academy of Sciences CR(head. Prof. J. Zvárová, PhD, DSc.). The data resource is on the web pages `http://euromise.vse.cz/challenge2004/`

Table 1. Attributes and categories of $\mathcal{LC}_\mathcal{E}$ calculus of association rules

Attribute				Names of categories	
Name					
Def.	Alternative	Type	C-type	Definition	Alternative / frequency
A_1	*M_Status*	4	N	1,2,3,4	*married*/1207, *divorced*/104, *single*/95, *widover*/10
A_2	*Education*	4	O	1,2,3,4	*basic*/151, *apprentice*/405, *secondary*/444, *university*/397
A_3	*Responsibility*	4	N	1,2,3,4	*manager*/286, *independent*/435, *others*/636, *pensioner*/25
A_4	*Alcohol*	3	O	1,2,3	*no*/131, *occasionally*/748, *regularly*/462
A_5	*Coffee*	3	O	1,2,3	*no*/488, *1-2 cups*/643, *3+ cups*/258
A_6	*BMI*	13	O	1,. . . , 13	$(16;21\rangle/39$, $(21;22\rangle/50$, . . . , . . . , $(31;32\rangle/28$, $> 32/64$
A_7	*Diastolic*	7	O	1, . . . , 7	$\langle 50;70)/74$, $\langle 70;80)/281$, . . . , . . . , $\langle 110;120)/43$, $\langle 120;150)/16$
A_8	*Systolic*	9	O	1, . . . , 9	$\langle 90;110)/69$, $\langle 110;120)/207$, . . . , . . . , $\langle 170;180)/43$, $\langle 180;220)/33$
A_9	*Cholesterol*	10	O	1, . . . , 10	$\langle 100;160)/45$, $\langle 160;180)/97$, . . . , . . . , $\langle 300;320)/57$, $\langle 320;540)/57$
A_{10}	*Hypertension*	2	N	1,2	*yes*/220, *no*/1192
A_{11}	*Ictus*	2	N	1,2	*yes*/2, *no*/1408
A_{12}	*Infarction*	2	N	1,2	*yes*/34, *no*/1378
$A_{13}, \dots, A_{64}$		see `http://euromise.vse.cz/challenge2004/data/entry/`			

Examples of basic Boolean attributes of the calculus $\mathcal{LC}_\mathcal{E}$ are: $A_1(1)$ – alternatively *M_Status*(*married*), $A_2(1,2)$ – alternatively *Education*(*basic*, *apprentice*), $A_6(1,2,3)$ – alternatively *BMI*($(16;21\rangle$, $(21;22\rangle$, $(21;22\rangle$) i.e. *BMI*$(16;22\rangle$.

Examples of Boolean attributes are: *M_Status*(*married*) $\wedge$ *BMI*$(16;22\rangle$ and *Hypertension*(*yes*) $\vee$ *Ictus*(*yes*) $\vee$ *Infarction*(*yes*).

Association rule $\varphi \approx \psi$ is true in a data matrix $\mathcal{M}$ if a condition given by the 4ft-quantifier $\approx$ is satisfied for a contingency table of ψ and φ in $\mathcal{M}$. This contingency table is also called *4ft-table* $4ft(\varphi,\psi,\mathcal{M})$ *of* φ *and* ψ *in* $\mathcal{M}$ and denoted as $4ft(\varphi,\psi,\mathcal{M})$. It is is a quadruple $\langle a,b,c,d\rangle$ of non-negative integers where a is the number of rows of $\mathcal{M}$ satisfying both φ and ψ, b is the number of rows satisfying φ and not satisfying ψ etc., see Fig. 2. Two 4ft-quantifiers are

$\mathcal{M}$	ψ	$\neg\psi$
φ	a	b
$\neg\varphi$	c	d

Fig. 2. 4ft table $4ft(\varphi,\psi,\mathcal{M})$ of φ and ψ in $\mathcal{M}$

introduced below, about 40 additional 4ft-quantifiers are defined in [3, 10].

4ft-quantifier $\Rightarrow_{p,B}$ of *founded implication* is defined for $0 < p \leq 1$ and $B > 0$ in [3] by the condition $\frac{a}{a+b} \geq p \wedge a \geq B$. Here $F_{\Rightarrow_{p,B}}$ is the associated function of $\Rightarrow_{p,B}$. Rule $\varphi \Rightarrow_{p,B} \psi$ means that at least $100p$ per cent of objects satisfying φ satisfy also ψ and that there are at least B rows of $\mathcal{M}$ satisfying both φ and ψ.

4ft-quantifier $\sim^+_{q,B}$ of *above average dependence* is for $0 < q$ and $B > 0$ defined in [10] by the condition $\frac{a}{a+b} \geq (1+q)\frac{a+c}{a+b+c+d} \wedge a \geq B$. Rule $\varphi \sim^+_{q,B} \psi$ means that among the rows satisfying φ, there are at least $100p$ per cent more rows atisfying ψ than among all rows of $\mathcal{M}$ and that there are at least B rows of $\mathcal{M}$ satisfying both φ and ψ.

This means that we can say that the calculus logical calculus $\mathcal{LC_E}$ of association rules has two 4ft-quantifiers: $\Rightarrow_{p,B}$ and $\sim^+_{q,B}$. It is easy to add additional 4ft-quantifiers defined in [3, 10].

Correct deduction rules $\frac{\varphi \approx \psi}{\varphi' \approx \psi'}$ where both $\varphi \approx \psi$ and $\varphi' \approx \psi'$ are association rules play a very important role in the FOFRADAR. Deduction rule $\frac{\varphi \approx \psi}{\varphi' \approx \psi'}$ is correct if it holds for each data matrix $\mathcal{M}$: *if* $\varphi \approx \psi$ *is true in* $\mathcal{M}$ *then also* $\varphi' \approx \psi'$ *is true in* $\mathcal{M}$. The rules $\frac{A(1) \Rightarrow_{p,B} B(1)}{A(1) \Rightarrow_{p,B} B(1,2)}$ and $\frac{A(1) \Rightarrow_{p,B} B(1)}{A(1) \Rightarrow_{p,B} B(1) \vee C(1)}$ are very simple examples of correct deduction rules. There are relatively simple criteria making possible to decide if a given deduction rule $\frac{\varphi \approx \psi}{\varphi' \approx \psi'}$ is correct. These criteria are known for most of important 4ft-quantifiers [10].

3.2 Language of Domain Knowledge

Language $\mathcal{L}_{DK}$ of domain knowledge is an enhancement of a language $\mathcal{L_T}$ of a calculus $\mathcal{LC_T}$ of association rules of type $\mathcal{T} = \langle t_1, \ldots, t_K \rangle$ [8]. The following items of domain knowledge can be expressed by formulas of $\mathcal{L}_{DK}$: (i) *C-types of basic attributes*, (ii) *groups of basic attributes*, (iii) *simple mutual influence of attributes.*

C-types of basic attributes are defined by a K-tuple $\mathcal{L}_{CT} = \langle \sigma_1, \ldots, \sigma_K \rangle$ where $\sigma_i \in \{N, O, C\}$ for $i = 1, \ldots, K$. This K-tuple is called *C-types of attributes.* If $\sigma_i = N$ then *the attribute* A_i *is nominal*, i.e., we do not assume to deal with ordering of its categories $1, \ldots, t_i$. If $\sigma_i = O$ then *the attribute* A_i *is ordinal* and we assume to use ordering of its categories $1, \ldots, t_i$. If $\sigma_i = C$ then *the attribute* A_i *is cyclical*, i.e., we assume ordering of its categories and in addition we assume that the category 1 follows the category t_i. An attribute *WeekDays* with categories *Su*, *Mo*, *Tu*, *We*, *Th*, *Fr*, *Sa* is an example of a cyclical attribute. C-types of attributes of calculus $\mathcal{LC_E}$ are given in Tab. 1.

We use two types of groups of basic attributes − basic and additional. There are L *basic groups* $G_1, \ldots G_L$ *of basic attributes* − subsets of $\{A_1, \ldots, A_K\}$ satisfying $L < K$, $\cup_{i=1}^{L} G_i = \{A_1, \ldots, A_K\}$ and $G_i \cap G_j = \emptyset$ for $i \neq j$, $i, j = 1, \ldots L$. Calculus $\mathcal{LC_E}$ has 11 basic groups, see `http://euromise.vse.cz/challenge2004/data/entry/`. The additional groups of basic attributes are usually defined for ad hoc analyses. We use here two such groups for calculus $\mathcal{LC_E}$: group *Personal* consisting of 6 attributes *M_Status*, *Education*, *Responsibility*, *Alcohol Coffee*, and *BMI* and group *Measurement* consisting of 3 attributes *Diastolic*, *Systolic*, and *Cholesterol*.

Mutual simple influence among attributes is expressed by *SI-formulas*. There are several types of SI-formulas. Below, we assume that A_i, A_j, $i \neq j$ are ordinal attributes and φ is a Boolean attribute. Examples of types of SI-formulas follow:

- *ii*-formula (i.e. increases - increases) $A_i \uparrow\uparrow A_j$ meaning *if* A_i *increases then* A_j *increases too*, *BMI* $\uparrow\uparrow$ *Diastolic* being an example
- *id*-formula (i.e. increases - decreases) $A_i \uparrow\downarrow A_j$ meaning *if* A_i *increases then* A_j *decreases*, *Education* $\uparrow\downarrow$ *Diastolic* being an example
- i^+b^+-formula has a form $A_i \uparrow^+ \varphi$ and its meaning is: *if* A *increases, then relative frequency of* φ *increases too*, *BMI* $\uparrow^+$ *Hypertension*(*yes*) being an example
- i^+b^--formula, i^-b^+-formula, i^-b^--formula have form $A_i \uparrow^- \varphi$, $A_i \downarrow^+ \varphi$, $A_i \downarrow^- \varphi$ respectively, their meaning is analogous to that of $A \uparrow^+ \varphi$.

4 From Domain Knowledge to Analytical Questions

Items of domain knowledge are used to formulate analytical questions. We introduce two types of such questions – GG-questions and negative GG_SI-question.

GG-question has form $[\mathcal{M} : G'_1, \ldots, G'_U \approx^? G''_1, \ldots G''_V]$ where $\mathcal{M}$ is a data matrix and $G'_1, \ldots, G'_U$ and $G''_1, \ldots G''_V$ are groups of attributes. A simple example is a formula Θ_1 defined as $\Theta_1 = [\mathit{Entry} : \mathit{P}\text{ersonal} \approx^? \mathit{M}\text{easurement}]$. Its meaning is: *In the data matrix* Entry*, are there any interesting relations between combinations of values of attributes of group Personal on one side and combinations of values of attributes of group Measurement on the other side?*

Negative GG_SI-question – its general form is
$[\mathcal{M} : (\Omega_1, \ldots, \Omega_P) \not\rightarrow G'_1, \ldots, G'_U \approx^? G''_1, \ldots G''_V]$ where $\mathcal{M}$, $G'_1, \ldots, G'_U$ and $G''_1, \ldots G''_V$ are as above and $\Omega_1, \ldots, \Omega_P$ are SI-formulas. A formula Θ_2 defined as $\Theta_2 = [\mathit{Entry} : \mathit{B}\text{MI} \uparrow\uparrow \mathit{D}\text{iastolic} \not\rightarrow \mathit{P}\text{ersonal} \approx^? \mathit{M}\text{easurement}]$ is a simple example of negative GG_SI-question. Its meaning is: *In the data matrix* Entry*, are there any interesting relations between combinations of values of attributes of group Personal on one side and combinations of values of attributes of group Measurement on the other side which are not consequences of BMI* $\uparrow\uparrow$ *Diastolic?*

The questions Θ_1 and Θ_2 are examples of formulas of a language $\mathcal{LAQ}_{\mathcal{E}}$ of analytical questions which is an enhancement of the language $\mathcal{L}_{\mathcal{E}}$ of calculus $\mathcal{LC}_{\mathcal{E}}$. Let us remember that a procedure DK_AQ is a part of the FOFRADAR. Its goal is to generate reasonable analytical questions. Input of DK_AQ consists of a list of groups and a list of SI-formulas of the language $\mathcal{L}_{DK}$. Let us only note that DK_AQ can be realized by suitable nested cycle statements.

We deal with the calculus $\mathcal{LC}_{\mathcal{E}}$. We assume that the only item of known domain knowledge is *BMI* $\uparrow\uparrow$ *Diastolic* and we are going to solve the question $\mathcal{Q}_2$. Our goal is to introduce in more details a language $\mathcal{L}_{Concl}$ of conclusions we can accept on results of mining association rules $\varphi \approx \psi$. Our goal is not to get new medical knowledge. Let as note that similar question is solved in [12], however, without details on a corresponding language $\mathcal{L}_{Concl}$.

5 Applying 4ft-Miner

5.1 Principles

We use the procedure 4ft-Miner [11] to solve the analytical question $\Theta_2 = [Entry : B\text{MI} \uparrow\uparrow D\text{iastolic} \not\to P\text{ersonal} \approx^? M\text{easurement}]$ introduced above. We deal with association rules, thus we formulate this question such that it deals with association rules:

$$\Theta_2 = [Entry : B\text{MI} \uparrow\uparrow D\text{iastolic} \quad \not\to \quad \mathcal{B}(P\text{ersonal}) \approx^? \mathcal{B}(M\text{easurement})] \ .$$

Here $\mathcal{B}(P\text{ersonal})$ denotes a set of all relevant Boolean attributes derived from attributes of the group Personal, similarly for $\mathcal{B}(M\text{easurement})$. We search association rules $\varphi_P \approx^? \psi_M$ which are true in data matrix *Entry*, cannot be understood as consequences BMI $\uparrow\uparrow$ Diastolic, $\approx^?$ is a suitable 4ft-quantifier, $\varphi_P \in \mathcal{B}(P\text{ersonal})$ and $\psi_M \in \mathcal{B}(M\text{easurement})$.

There are very fine possibilities to define a set of relevant association rules, they are given by input parameters of 4ft-Miner. A definition of values of these parameters can be understood as an expression of a language $\mathcal{L}_{RAR}$. There are enough experience [10] making possible to construct a procedure AQ_RAR assigning to each analytical question Θ (i.e. a formula of language $\mathcal{L}_{AQ}$) a formula $AQ\_RAR(\Theta)$ of language $\mathcal{L}_{RAR}$ such that $AQ\_RAR(\Theta)$ defines parameters for a run of the 4ft-Miner procedure suitable to solve Θ.

In section 5.2 a formula $\Phi = AQ\_RAR(\Theta_2)$ is introduced. It defines a set $\mathcal{S}(\Phi)$ of association rules relevant to the question Θ_2. Input of the 4ft-Miner procedure consists of the formula Φ and data matrix *Entry*. The output is a set $True(\mathcal{S}(\Phi), Entry)$ of all rules $\varphi \approx \psi$ which belong to $\mathcal{S}(\Phi)$ and which are true in *Entry*, see section 5.3.

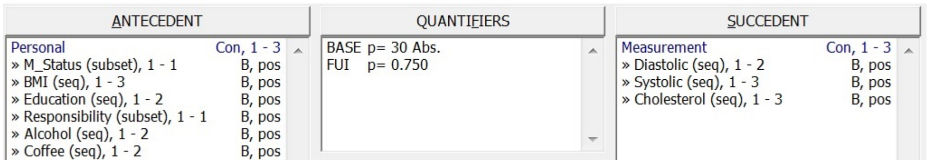
5.2 From Analytical Questions to Parameters of 4ft-Miner

In Fig. 3, there are input parameters of the 4ft-Miner. They can be seen as the formula $\Phi = AQ\_RAR(\Theta_2)$. We search for association rules $\varphi_P \approx^? \psi_M$ where $\approx^?$ is a suitable 4ft-quantifier, $\varphi_P \in \mathcal{B}(P\text{ersonal})$ and $\psi_M \in \mathcal{B}(M\text{easurement})$.

Let us also note that $\Phi = AQ\_RAR(\Theta_2)$ is constructed to get a reasonable output without necessity to modify parameters. Actually, the application of 4ft-Miner requires modifications of an initial setting of parameters. This can also be included into the AQ_RAR procedure. Description of this possibility is not the goal of this paper.

The set $\mathcal{B}(Personal)$ is defined in Fig. 3 in the column `ANTECEDENT` in the row `Personal Conj, 1-3` and in the six consecutive rows. This means that φ_P is a conjunction of 1 - 3 basic Boolean attributes derived from attributes of the group Personal introduced in section 3.2, see also Tab. 1.

A set of all basic Boolean attributes derived from attribute $M_$Status is defined by the row `M_Status(subset), 1-1 B, pos`. This means that basic Boolean attributes $M_$Status($m$arried), $M_$Status($d$ivorced), $M_$Status(single),



```
ANTECEDENT                              QUANTIFIERS              SUCCEDENT
Personal                  Con, 1 - 3    BASE p= 30 Abs.          Measurement              Con, 1 - 3
» M_Status (subset), 1 - 1   B, pos     FUI   p= 0.750           » Diastolic (seq), 1 - 2     B, pos
» BMI (seq), 1 - 3           B, pos                              » Systolic (seq), 1 - 3      B, pos
» Education (seq), 1 - 2     B, pos                              » Cholesterol (seq), 1 - 3   B, pos
» Responsibility (subset), 1 - 1  B, pos
» Alcohol (seq), 1 - 2       B, pos
» Coffee (seq), 1 - 2        B, pos
```

Fig. 3. Input parameters of the 4ft-Miner procedure

and $M\_\mathrm{Status}(widover)$ are generated. A set of all basic Boolean attributes derived from attribute Responsibility is defined similarly.

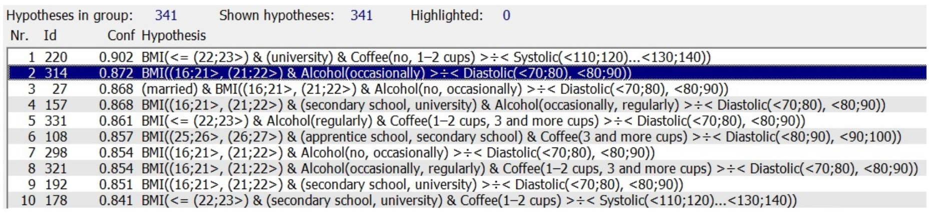
A set of all basic Boolean attributes $B\mathrm{MI}(\alpha)$ derived from attribute $B\mathrm{MI}$ is defined by the row `BMI(int), 1-3 B, pos`. This means that all $B\mathrm{MI}(\alpha)$ are generated such that α is a set of 1 - 3 consecutive categories (i.e. sequence of categories). Expression $B\mathrm{MI}(\langle 21;22\rangle, \langle 22;23\rangle)$ i.e. $B\mathrm{MI}(21;23\rangle$ is an example of such basic Boolean attribute. Sets of Boolean attributes derived from attributes Education, Alcohol and Coffee are defined similarly.

The set $\mathcal{B}(Measurement)$ is defined analogously in Fig. 3 in the column `SUCCEDENT`. In the column `QUANTIFIERS`, the quantifier $\approx^?$ is specified as a 4ft-quantifier $\Rightarrow_{0.75,30}$ of founded implication.

The formula $\Phi = AQ\_RAR(\Theta_2)$ can be seen as a triple $\langle ANT_{\Theta_2}, \Rightarrow_{0.75,30}, SUC_{\Theta_2}\rangle$ where ANT_{Θ_2} and SUC_{Θ_2} are definitions of sets $\mathcal{B}(ANT_{\Theta_2})$ and $\mathcal{B}(SUC_{\Theta_2})$ of Boolean attributes respectively. The triple $\langle ANT_{\Theta_2}, \Rightarrow_{0.75,30}, SUC_{\Theta_2}\rangle$ defines a set $\mathcal{S}(ANT_{\Theta_2}, \Rightarrow_{0.75,30}, SUC_{\Theta_2})$ of association rules $\varphi \Rightarrow_{0.75,30} \psi$ such that $\varphi \in \mathcal{B}(ANT_{\Theta_2})$, $\psi \in \mathcal{B}(SUC_{\Theta_2})$ and φ and ψ have no common basic attributes. The definitions ANT_{Θ_2} and SUC_{Θ_2} can be seen as sets of parameters in columns `ANTECEDENT` and `SUCCEDENTS` in Fig. 3 respectively. Then we have $\mathcal{B}(ANT_{\Theta_2}) = \mathcal{B}(Personal)$ and $\mathcal{B}(SUC_{\Theta_2}) = \mathcal{B}(Measurement)$.

5.3 4ft-Miner Output

The task specified in Fig. 3 was solved in 2 minutes (PC with 4GB RAM and Intel(R) Core(TM) i5-3320 processor at 2.6 GHz). 10^7 association rules were generated and tested, there are 341 output true rules. List of 10 rules with the highest confidence is in Fig. 4.



```
Hypotheses in group:   341      Shown hypotheses:   341      Highlighted:   0
Nr.  Id      Conf Hypothesis
  1  220    0.902 BMI(<= (22;23>) & (university) & Coffee(no, 1–2 cups) >÷< Systolic(<110;120)...<130;140))
  2  314    0.872 BMI((16;21>, (21;22>) & Alcohol(occasionally) >÷< Diastolic(<70;80), <80;90))
  3   27    0.868 (married) & BMI((16;21>, (21;22>) & Alcohol(no, occasionally) >÷< Diastolic(<70;80), <80;90))
  4  157    0.868 BMI((16;21>, (21;22>) & (secondary school, university) & Alcohol(occasionally, regularly) >÷< Diastolic(<70;80), <80;90))
  5  331    0.861 BMI(<= (22;23>) & Alcohol(regularly) & Coffee(1–2 cups, 3 and more cups) >÷< Diastolic(<70;80), <80;90))
  6  108    0.857 BMI((25;26>, (26;27>) & (apprentice school, secondary school) & Coffee(3 and more cups) >÷< Diastolic(<80;90), <90;100))
  7  298    0.854 BMI((16;21>, (21;22>) & Alcohol(no, occasionally) >÷< Diastolic(<70;80), <80;90))
  8  321    0.854 BMI((16;21>, (21;22>) & Alcohol(occasionally, regularly) & Coffee(1–2 cups, 3 and more cups) >÷< Diastolic(<70;80), <80;90))
  9  192    0.851 BMI((16;21>, (21;22>) & (secondary school, university) >÷< Diastolic(<70;80), <80;90))
 10  178    0.841 BMI(<= (22;23>) & (secondary school, university) & Coffee(1–2 cups) >÷< Systolic(<110;120)...<130;140))
```

Fig. 4. Example of 4ft-Miner output

The rule $B\mathrm{MI}(16;22) \wedge \mathrm{Alcohol}(\mathit{occasionally}) \Rightarrow_{0.872,34} D\mathrm{iastolic}\langle 70;90)$ is the second strongest one. This rule means that it holds in data matrix *Entry*: at least 87.2 per cent of patients satisfying $B\mathrm{MI}(16;22) \wedge \mathrm{Alcohol}(\mathit{occsionally})$ satisfy also $D\mathrm{iastolic}\langle 70;90)$ and there are at least 34 patients satisfying both $B\mathrm{MI}(16;22) \wedge \mathrm{Alcohol}(\mathit{occsionally})$ and $D\mathrm{iastolic}\langle 70;90)$, this information about 34 patients can be seen only in detailed output, not in Fig. 4.

Most of found rules have both the attribute BMI in antecedent (i.e. left part of a rule) and the attribute Diastolic in succedent (i.e. right part of a rule). We can expect that lot of such rules can be seen as consequences of SI-formula $B\mathrm{MI} \uparrow\uparrow D\mathrm{iastolic}$ introduced in section 3.2.

6 Consequences of SI-Formulas

Let Γ be an SI-formula and $\approx$ be a 4ft-quantifier, then $Cons(\Gamma, \approx)$ denotes a set of association rules which can be considered as consequences of Γ. The set $Cons(\Gamma, \approx)$ is defined in four steps [8].

1. A set $AC(\Gamma, \approx)$ of *atomic consequences of* Γ for $\approx$ is defined as a set of very simple rules $\kappa \approx' \lambda$ which can be, according to the domain expert, considered as direct consequences of Γ.
2. A set $AgC(\Gamma, \approx)$ of *agreed consequences of* Γ for $\approx$ is defined. A rule $\rho \approx' \sigma$ belongs to $AgC(\Gamma, \approx)$ if the following conditions are satisfied:
   – $\rho \approx' \sigma \notin AC(\Gamma, \approx)$
   – there is no $\kappa \approx' \lambda \in AC(\Gamma, \approx)$ such that $\rho \approx' \sigma$ logically follows from $\kappa \approx' \lambda$
   – there is $\kappa \approx' \lambda \in AC(\Gamma, \approx)$ such that, according to the domain expert, it is possible to agree that $\rho \approx' \sigma$ says nothing new in addition to $\kappa \approx' \lambda$.
3. A set $LgC(\Gamma, \approx)$ of *logical consequences of* Γ for $\approx$ is defined. A rule $\varphi \approx' \psi$ belongs to $LgC(\Gamma, \approx)$ if the following conditions are satisfied:
   – $\varphi \approx' \psi \notin (AC(\Gamma, \approx) \cup AgC(\Gamma, \approx))$
   – there is $\tau \approx' \omega \in AC(\Gamma, \approx) \cup AgC(\Gamma, \approx)$ such that $\varphi \approx' \psi$ logically follows from $\tau \approx' \omega$.
4. We define $Cons(\Gamma, \approx) = AC(\Gamma, \approx) \cup AgC(\Gamma, \approx) \cup LgC(\Gamma, \approx)$.

We outline a way in which a set $Cons(BMI \uparrow\uparrow Diastolic, \Rightarrow_{0.75,30})$ can be defined. Note that the 4ft-quantifier $\Rightarrow_{0.75,30}$ is used in the example in section 5. A rule $B\mathrm{MI}(l\mathrm{ow}) \Rightarrow_{0.75,30} D\mathrm{iastolic}(l\mathrm{ow})$ saying that at least 75 per cent of patients satisfying $B\mathrm{MI}(l\mathrm{ow})$ satisfy also $D\mathrm{iastolic}(l\mathrm{ow})$ and that there are at least 30 patients satisfying both $B\mathrm{MI}(l\mathrm{ow})$ and $D\mathrm{iastolic}(l\mathrm{ow})$ can be considered as a simple consequence of $B\mathrm{MI} \uparrow\uparrow D\mathrm{iastolic}$. In addition, if we consider $B\mathrm{MI}(l\mathrm{ow}) \Rightarrow_{0.75,80} D\mathrm{iastolic}(l\mathrm{ow})$ as a consequence of $B\mathrm{MI} \uparrow\uparrow D\mathrm{iastolic}$, then also each rule $B\mathrm{MI}(l\mathrm{ow}) \Rightarrow_{p,B} D\mathrm{iastolic}(l\mathrm{ow})$ where $p \geq 0.75 \wedge B \geq 30$ is a consequence of $B\mathrm{MI} \uparrow\uparrow D\mathrm{iastolic}$. The only problem is to define suitable coefficients $l$ow for both attributes BMI and Diastolic.

Attribute BMI has 13 categories: $(16;21\rangle$, $(21;22\rangle$, $(22;23\rangle$, $(23;24\rangle$, $\ldots$, $(31;32\rangle$, > 32. Attribute Diastolic has 7 categories: $\langle 50;70)$, $\langle 70;80)$, $\langle 80;90)$, $\ldots$,

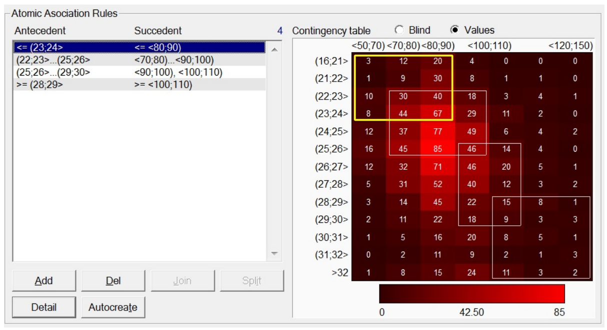
$\langle 110; 120), \langle 120; 150)$. We can decide that each basic Boolean attribute $B\text{MI}(\alpha)$ satisfying condition $\alpha \subseteq \{(16; 21\rangle, (21; 22\rangle, (22; 23\rangle, (23; 24\rangle\}$ will be considered as $B\text{MI}(low)$, and similarly, each basic Boolean attribute $D\text{iastolic}(\beta)$ where $\beta \subseteq \{\langle 50; 70), \langle 70; 80), \langle 80; 90)\}$ will be considered as $S\text{yst}(low)$. We can say that rules $B\text{MI}(low) \Rightarrow_{0.75,30} D\text{iastolic}(low)$ are defined by a rectangle $\mathcal{A}_{low} \times \mathcal{S}_{low}$:

$$\mathcal{A}_{low} \times \mathcal{S}_{low} = \{(16; 21\rangle, (21; 22\rangle, (22; 23\rangle\ (23; 24\rangle\} \times \{\langle 50; 70), \langle 70; 80), \langle 80; 90)\}\ .$$

The 4ft-Miner procedure is accompanied by the *LMDataSource* module which makes possible to define the set $AC(B\text{MI} \uparrow\uparrow D\text{iastolic}, \Rightarrow_{0.75,30})$ as a union of several similar, possibly overlapping, rectangles $\mathcal{A}_1 \times \mathcal{S}_1, \ldots, \mathcal{A}_R \times \mathcal{S}_R$ such that $B\text{MI}(\alpha) \Rightarrow_{p,B} D\text{iastolic}(\beta) \in AC(BMI \uparrow\uparrow Diastolic, \Rightarrow_{0.75,30})$ if and only if it holds $p \geq 0.75 \wedge B \geq 30$ and there is an $i \in \{1, \ldots, R\ \}$ such that $\alpha \subseteq \mathcal{A}_i$ and $\beta \subseteq \mathcal{S}_i$. An example is in Fig. 5, four rectangles are used. Very informally speaking, we can see $AC(BMI \uparrow\uparrow Diastolic, \Rightarrow_{0.75,30})$ as a union

$$\mathcal{A}_{low} \times \mathcal{S}_{low} \cup \mathcal{A}_{below\ avrg} \times \mathcal{S}_{below\ avrg} \cup \mathcal{A}_{above\ avrg} \times \mathcal{S}_{above\ avrg} \cup \mathcal{A}_{high} \times \mathcal{S}_{high}$$

where *avrg* abbreviates *average*.



Atomic Asociation Rules

Antecedent	Succedent
<= (23;24>	<= <80;90)
(22;23>...(25;26>	<70;80)...<90;100)
(25;26>...(29;30>	<90;100), <100;110)
>= (28;29>	>= <100;110)

4

Contingency table ○ Blind [✓ Values]

	<50;70)	<70;80)	<80;90)		<100;110)		<120;150)
(16;21>	3	12	20	4	0	0	0
(21;22>	1	9	30	8	1	1	0
(22;23>	10	30	40	18	3	4	1
(23;24>	8	44	67	29	11	2	0
(24;25>	12	37	77	49	6	4	2
(25;26>	16	45	85	46	14	4	0
(26;27>	12	32	71	46	20	5	1
(27;28>	5	31	52	40	12	3	2
(28;29>	3	14	45	22	15	8	1
(29;30>	2	11	22	18	9	3	3
(30;31>	1	5	16	20	8	5	1
(31;32>	0	2	11	9	2	1	3
>32	1	8	15	24	11	3	2

Fig. 5. Definition of $AC(BMI \uparrow\uparrow Syst, \Rightarrow_{0.7,80})$

The rule $B\text{MI}(16; 22\rangle \wedge A\text{lcohol}(\text{occasionally}) \Rightarrow_{0.87,34} D\text{iastolic}\langle 70; 90)$ is an example of an agreed consequence. This rule does not logically follow from an atomic consequence $B\text{MI}(16; 22\rangle \Rightarrow_{0.87,34} D\text{iastolic}\langle 70; 90)$ but it is possible to agree that it says nothing new to the rule $B\text{MI}(16; 22\rangle \Rightarrow_{0.87,34} D\text{iastolic}\langle 70; 90)$.

The rule $B\text{MI}(16; 22\rangle \wedge M\_\text{Status}(m\text{arried}) \Rightarrow_{0.78,31} D\text{iastolic}\langle 80; 100)$ is an example of a logical consequence of an agreed consequence. This is because $B\text{MI}(16; 22\rangle \Rightarrow_{0.78,31} D\text{iastolic}\langle 80; 90)$ is an atomic consequence, $B\text{MI}(16; 22\rangle \wedge M\_\text{Status}(m\text{arried}) \Rightarrow_{0.78,31} D\text{iastolic}\langle 80; 90)$ is its agreed consequence and $B\text{MI}(16; 22\rangle \wedge M\_\text{Status}(m\text{arried}) \Rightarrow_{0.78,31} D\text{iastolic}\langle 80; 100)$ is a logical consequence of $B\text{MI}(16; 22\rangle \wedge M\_\text{Status}(m\text{arried}) \Rightarrow_{0.78,31} D\text{iastolic}\langle 80; 90)$.

This way, a set $Cons(BMI \uparrow\uparrow Diastolic, \Rightarrow_{0.75,30})$ is produced. Let us note that deduction rules $\frac{\varphi \Rightarrow_{p,B} \psi}{\varphi' \Rightarrow_{p,B} \psi'}$ (see end of section 3.1) play a crucial role in these considerations. Additional examples of these considerations are in [8, 12]. Similar considerations are valid for additional 4ft-quantifiers.

We can summarise: The input of the 4ft-Miner is a triple $\langle ANT, \approx, SUC\rangle$ and a data matrix $\mathcal{M}$. The triple $\langle ANT, \approx, SUC\rangle$ defines a set $\mathcal{S}(ANT, \approx, SUC)$ of rules. Output of the 4ft-Miner is a set $True(\mathcal{S}(ANT, \approx, SUC), \mathcal{M})$ of all rules from $\mathcal{S}(ANT, \approx, SUC)$ which are true in $\mathcal{M}$. We have outlined that there is a procedure $CONS$ input of which is an SI-formula Γ and a 4ft-quantifier $\approx$ and output of $CONS$ is a set $Cons(\Gamma, \approx)$ of all rules belonging to $\mathcal{S}(ANT, \approx, SUC)$ which can be considered as consequences of Γ. In addition, we have introduced a set $Cons(BMI \uparrow\uparrow \mathit{Diastolic}, \Rightarrow_{0.75,30})$.

7 Language of Conclusions

The goal of this section is to introduce formulas of the language $\mathcal{L}_{Concl}$ which represent conclusions of analysis. The conclusions are formulated on the basis of a comparison of a set $True(\mathcal{S}(ANT, \approx, SUC), \mathcal{M})$ resulting from the run of the procedure 4ft-Miner and sets $Cons(\Gamma, \approx)$ for relevant SI-formulas Γ. An SI-formula Γ is relevant if it is used in the analytical question or if it can be formulated from the attributes which occur in $True(\mathcal{S}(ANT, \approx, SUC), \mathcal{M})$.

When dealing with a particular SI-formula Γ, we are interested in relations of sets of rules $True(\mathcal{S}(ANT, \approx, SUC), \mathcal{M})$ and $Cons(\Gamma, \approx)$. The conclusions can be formulated on the basis of their intersection $True(\mathcal{S}(ANT, \approx, SUC), \mathcal{M}) \cap Cons(\Gamma, \approx)$ and difference $True(\mathcal{S}(ANT, \approx, SUC), \mathcal{M}) \setminus Cons(\Gamma, \approx)$. These can be also produced by the 4ft-Miner procedure. In addition, it is possible to sort and filter rules of these sets in various ways. Relations of $True(\mathcal{S}(ANT, \approx, SUC), \mathcal{M})$ and $Cons(\Gamma, \approx)$ can be also investigated by SQL tools. This can lead to variety of conclusions on result of application of the procedure 4ft-Miner.

We outline four of them. We use the analytical question Θ_2 introduced in section 4 as $\Theta_2 = [\mathit{Entry} : \text{BMI} \uparrow\uparrow \text{Diastolic} \not\rightarrow \text{Personal} \approx^? \text{Measurement}]$. A formula $AQ\_RAR(\Theta_2)$ of language $\mathcal{L}_{RAR}$ is introduced in sections 5.1 and 5.2. It defines a set $\mathcal{S}(AQ\_RAR(\Theta_2))$ of association rules we consider relevant to the question Θ_2. The formula $AQ\_RAR(\Theta_2)$ is specified as $\langle ANT_{\Theta_2}, \Rightarrow_{0.75,30}, SUC_{\Theta_2}\rangle$ where $\mathcal{B}(ANT_{\Theta_2}) = \mathcal{B}(\mathit{Personal})$ and $\mathcal{B}(SUC_{\Theta_2}) = \mathcal{B}(\mathit{Measurement})$. The procedure 4ft-Miner produces the set $True(\mathcal{S}(ANT_{\Theta_2}, \Rightarrow_{0.75,30}, SUC_{\Theta_2}), \mathit{Entry})$ of all relevant association rules true in the data matrix *Entry*. We denote it as $True(\Theta_2, \mathit{Entry})$.

We denote a difference $True(\Theta_2, \mathit{Entry}) \setminus Cons(BMI \uparrow\uparrow \mathit{Diastolic}, \Rightarrow_{0.75,30})$ of sets $True(\Theta_2, \mathit{Entry})$ and $Cons(BMI \uparrow\uparrow \mathit{Diastolic}, \Rightarrow_{0.75,30})$ as Dif_Tr_Con. We assume that both the set $Cons(BMI \uparrow\uparrow \mathit{Diastolic}, \Rightarrow_{0.75,30})$ and the set $True(\Theta_2, \Rightarrow_{0.75,30}, \mathit{Entry})$ are not empty. Then one or more from the following possibilities P1, P2, P3, P4 can occur:

P1: $True(\Theta_2, \Rightarrow_{0.75,30}, \mathit{Entry}) = Cons(BMI \uparrow\uparrow \mathit{Diastolic}, \Rightarrow_{0.75,30})$. This means that the conclusion can be: *In the data matrix* Entry*, all potentially interesting relations between combinations of values of attributes of group Personal* *on one side and combinations of values of attributes of group Measurement on the other side are consequences of BMI* $\uparrow\uparrow$ *Diastolic.*

P2: There is a rule $\varphi \Rightarrow_{p,B} \psi \in Dif\_Tr\_Con$ (where $p \geq 0.75$ and $B \geq 30$) such that attribute *BMI* does not occur in φ or attribute *Diastolic* does not occur in ψ. The rule $Education(apprentice) \wedge Responsibility(independent) \Rightarrow_{0.78,54} Diastolic\langle 80; 100)$ is an example. This means that the conclusion can be: *In the data matrix* Entry*, there are the following interesting true rules which are not consequences of BMI* $\uparrow\uparrow$ *Diastolic:* a list of rules $\varphi \Rightarrow_{p,B} \psi \in Dif\_Tr\_Con$ satisfying that attribute *BMI* does not occur in φ or attribute *Diastolic* does not occur in ψ *follows.*

P3: There is a rule $\varphi \Rightarrow_{p,B} \psi \in Dif\_Tr\_Con$ (where $p \geq 0.75$ and $B \geq 30$) such that attribute *BMI* occurs in φ and attribute *Diastolic* occurs in ψ. This rule is then true in data matrix *Entry* and it is not a consequence BMI $\uparrow\uparrow$ Diastolic, thus we can consider it as an exception to BMI $\uparrow\uparrow$ Diastolic. This means that the conclusion can be: *In the data matrix* Entry*, there are the following interesting true rules which can be considered as exceptions from BMI* $\uparrow\uparrow$ *Diastolic:* a list of rules $\varphi \Rightarrow_{p,B} \psi \in Dif\_Tr\_Con$ such that attribute *BMI* occurs in φ and attribute *Diastolic* occurs in ψ *follows.*

This approach to exceptions differs from that in [15]. Let us note that we can modify the definition of $Cons(BMI \uparrow\uparrow Diastolic, \Rightarrow_{0.75,30})$ such that the rule $BMI(21; 22\rangle \wedge M\_Status(married) \Rightarrow_{0.77,36} Diastolic\langle 80; 100)$ which is true in *Entry* does not belong to $Cons(BMI \uparrow\uparrow Diastolic, \Rightarrow_{0.75,30})$.

P4: There is an additional SI-formula Γ' such that there are enough rules in $Cons(\Gamma', \Rightarrow_{0.75,30}) \cap True(\Theta_2, \Rightarrow_{0.75,30}, Entry)$. An example is the SI-formula BMI $\uparrow\uparrow$ Systolic. We can define a set $Cons(BMI \uparrow\uparrow Systolic, \Rightarrow_{0.75,30})$ such that there are 182 rules $\varphi \Rightarrow_{p,B} \psi$ true in *Entry* where $p \geq 0.75$, $B \geq 30$, attribute *BMI* occurs in φ and attribute *Systolic* occurs in ψ and all of these rules belong to $Cons(BMI \uparrow\uparrow Systolic, \Rightarrow_{0.75,30})$. This means that the conclusion can be (here we consider $BMI \uparrow\uparrow Systolic$ as an unknown item of domain knowledge): *In the data matrix* Entry*, there are lot of rules which can be considered as consequences of yet unknown item of knowledge BMI* $\uparrow\uparrow$ *Systolic. It is reasonable to investigate BMI* $\uparrow\uparrow$ *Systolic as a working hypothesis.*

Let us note that additional conclusions can be formulated on the basis of a difference $Cons(\Gamma, \approx) \setminus True(\mathcal{S}(ANT, \approx, SUC), \mathcal{M})$ of sets $Cons(\Gamma, \approx)$ and $True(\mathcal{S}(ANT, \approx, SUC), \mathcal{M})$.

8 Related work

The FOFRADAR framework deals with association rules of the form $\varphi \approx \psi$ where φ and ψ are general Boolean attributes derived from columns of analysed data matrices and $\approx$ stands for a general condition concerning a contingency table of φ and ψ. A crucial feature of this approach is dealing with basic Boolean attributes of the form $A(\alpha)$ where α is a *subset of categories.* This makes possible to deal with rules like BMI($l$ow) $\Rightarrow_{p,B}$ Diastolic($l$ow) where "$l$ow" stands for a suitable subsets of categories. This way, dealing with items of domain knowledge like BMI $\uparrow\uparrow$ Diastolic can be converted to dealing with suitable association rules when suitable deduction rules $\frac{\varphi \approx \psi}{\varphi' \approx \psi'}$ are applied, see section 6 and also section 5.2.

An additional important feature of rules $\varphi \approx \psi$ is a possibility to deal with disjunction of basic Boolean attributes. This is especially important when dealing with attributes with low frequencies. Examples of such attributes of data matrix *Entry* are *Hyperlipidemia*(*yes*) – with frequency 54, *Diabetes*(*yes*) – 30, *Ictus*(*yes*) – 2, *Infarction*(*yes*) – 34. All these frequencies are very low and thus it is crucial to deal with disjunction of these attributes instead of with conjunctions, the frequencies of conjunctions are almost null. In addition, disjunctions $Hyperlipidemia(yes) \vee Diabetes(yes)$, $Hyperlipidemia(yes) \vee Ictus(yes), \ldots$ can be interpreted as "patient has problems". An example of dealing with such conjunctions is in [12]. Disjunctions of basic Boolean attributes are also involved in the FOFRADAR approach, deduction rules $\frac{\varphi \approx \psi}{\varphi' \approx \psi'}$ are very important in this case.

"Classical" association rules introduced in [1] deal neither with basic Boolean attributes $A(\alpha)$ nor disjunctions of basic Boolean attributes. Thus the approach to dealing with domain knowledge used in FOFRADAR cannot be fully applied when dealing with "classical" association rules. Let us note that the procedure 4FT introduced in [6] mines even for more general rules than the 4ft-Miner procedure used in this paper.

An approach to use ontologies expressing a hierarchy to interpret a resulting set of "classical" association rules is described in [4]. Very informally speaking, this is based on partitioning a set of resulting rules to sets of known rules, novel rules, missing rules and contradictory rules depending on their relation to an ontology in question. This is similar to assigning a set of consequences to an SI-formula as described in section 6. However, domain knowledge expressed by SI-formulas differs from domain knowledge expressed by a given ontology.

The goal of FOFRADAR is to be a formal description of the data mining process with association rules. Thus, using ontologies in FOFRADAR approach is a challenge. Additional challenge is related to inclusion of suitable means of inductive databases languages [13] as well as the additional approaches to deal with domain knowledge in data mining with "classical" association rules, see e.g. [2, 5].

9 Conclusions

We have presented new considerations on language $\mathcal{L}_{Concl}$. Formulas of this language correspond to conclusions of data mining process with association rules. $\mathcal{L}_{Concl}$ is one of languages of the formal framework FOFRADAR the goal of which is to formally describe the whole process of data mining with association rules. FOFRADAR is intended as a theoretical basis for automation of data mining process with association rules. The results presented here will be used together with former results [7, 8, 11, 12] to start experiments with automation of data mining process with association rules.

The FOFRADAR framework deals with association rules which are substantially more general than "classical" association rules defined in [1] and used in mainstream applications of association rules. There are various approaches to deal with domain knowledge in "classical" association rules data mining which can be included to FOFRADAR. However, this is rather a long process requiring additional effort.

References

1. Agrawal, R., Imielinski, T., Swami, A.: (1993) Mining Associations between Sets of Items in Large Databases. In: Buneman, P., Jajodia, S. (eds.) Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 207–216. ACM Press, Fort Collins
2. Delgado, M. et al.: (1998) Mining association rules with improved semantics in medical databases, *Artificial Intelligence in Medicine*, **21**(1–3), 2001, pp. 241–245
3. Hájek, P., Havránek, T.: (1978) Mechanising Hypothesis Formation - Mathematical Foundations for a General Theory. Springer, Berlin Heidelberg New York
4. Mansingh, G., Osei-Bryson, K.-M., Reichgelt. H.: (2011) Using ontologies to facilitate post-processing of association rules byh domain experts. Information Sciences, **181**(3), pp. 419–434
5. Ordonez, C., Ezquerra, N., Santana, C.A.: (2006) Constraining and Summarizing Association Rules in Medical Data, *Knowledge and Information Systems (KAIS)*, **9**(3), pp. 259–283.
6. Ralbovský, M., Kuchař, T.: (2009) Using Disjunctions in Association Mining. In: Perner, P. (ed.) Advances in Data Mining, Springer, Berlin, pp. 339–351
7. Rauch, J.: (2012) Formalizing Data Mining with Association Rules. In: IEEE International Conference on Granular Computing. Los Alamitos : IEEE Computer Society, 2012, pp. 485–490.
8. Rauch, J.: (2012) Domain Knowledge and Data Mining with Association Rules - a Logical Point of View. In: Foundations of Intelligent Systems. Springer, Berlin, pp. 11–20
9. Rauch, J.: (2012) EverMiner: consideration on knowledge driven permanent data mining process. International Journal of Data Mining, Modelling and Management, **4** (3), pp. 224–243
10. Rauch, J.: (2013) Observational Calculi and Association Rules. Springer, Berlin
11. Rauch, J., Šimůnek, M.: (2005) An Alternative Approach to Mining Association Rules. In: Lin, T. Y. et al. (eds) Data Mining: Foundations, Methods, and Applications, Springer, Berlin, pp. 219–238
12. Rauch, J., Šimůnek, M.: (2011) Applying Domain Knowledge in AssociationRules Mining Process - First Experience. In: Kryszkiewicz, M. et al.: (eds) Foundations of Intelligent Systems, Springer, Berlin, pp. 113–122
13. Romei, A., Turini, F.: (2011) Inductive database languages: requirements and examples. Knowledge and Information Systems, **26**(3), pp. 351–384
14. Šimůnek, M., Rauch J.: EverMiner Towards Fully Automated KDD Process. In: Funatsu, K., Hasegava, K. (eds.) New Fundamental Technologies in Data Mining, pp. 221-240. InTech, Rijeka (2011)
15. Suzuki, E. (2004) Discovering interesting exception rules with rule pair. In J. Fuernkranz (Ed.), Proceedings of the ECML/PKDD Workshop on Advances in Inductive Rule Learning pp. 163–178

Lower and upper queries for graph-mining[⋆]

Amina Kemmar [1], Yahia Lebbah [1,3], Samir Loudni [2], and Mohammed Ouali [1]

[1] Laboratoire LITIO, Université d'Oran
BP 1524, El-M'Naouer, 31000 Oran, Algérie
[2] Université de Caen Basse-Normandie, UMR 6072 GREYC, F-14032 Caen, France
[3] Laboratoire I3S/CNRS, Université de Nice - Sophia Antipolis, Nice, France
{kemmami,ylebbah}@yahoo.fr,samir.loudni@unicaen.fr,mohammed.ouali@univ-oran.dz

Abstract. Canonical encoding is one of the key operations required by subgraph mining algorithms for candidates generation. They enable to query the exact number of frequent subgraphs. Existing approaches make use of canonical encodings with an exponential time complexity. As a consequence, mining all frequent patterns for large graphs is computationally expensive. In this paper, we propose to relax the canonicity property, leading to two encodings, *lower and upper encodings*, with a polynomial time complexity, allowing to tightly enclose the exact set of frequent subgraphs. These two encodings allow two kinds of queries, lower and upper queries, to get respectively a subset and a superset of frequent patterns.
Lower and upper encodings have been integrated in `Gaston`. Experiments performed on large and dense synthetic graphs show that, these two encodings are very effective compared to `Gaston` and `gSpan`, while on large real world sparse graphs they remain very competitive.

1 Introduction

Frequent subgraph pattern mining is one of the most well-studied problems in the graph mining domain. It concerns the discovery of subgraph patterns that occur frequently in a collection of graphs or in a single large graph. This problem arises in many data mining tasks that include: chemoinformatics [13] and computational biology [6], to name but a few. In [10], an approach based on closed graphs is proposed, to predict biological activity of chemical compounds. Closed sets of graphs allow one to adapt standard approaches from applied lattice theory and Formal Concept Analysis to graph mining. An other approach is proposed in [2], where graphs are represented by (multi)sets of standard subgraphs representing substructures which are biologically meaningful, given by domain expert. In this paper, we consider a *database of graphs*, where each graph represents a transaction.

[⋆] This work is supported by TASSILI research program 11MDU839 (France, Algeria). This work is also supported by PNR research project 70/TIC/2011 (Algeria).

There are two general approaches used to solve the frequent subgraph pattern mining problem. The first approach, represented by [8,9], extends the Apriori-based candidate generation approach [1] to graph pattern mining. The second approach, represented by [3,7,11,15], adopt a pattern-growth principle [5] by growing patterns from a single graph directly. All these algorithms are complete: they are guaranteed to discover all frequent subgraphs.

Due to the completeness requirements, most of these algorithms need to perform graph isomorphism operations in order to check whether two subgraphs are identical or not in the candidate generation. This problem is equivalent to compare *canonical encodings* [4] of graphs. Even though these two problems are not known to be either in P or in NP-complete, in practice however, most existing canonical encodings have complexities which are of exponential nature. As a consequence, many existing complete algorithms impose strong limitations on the types of graph datasets that can be mined in a reasonable amount of time, as those derived from chemical compounds [3,7,8,13,15]: graphs that are sparse, contain small frequent subgraphs, and have very few vertex and edge labels. For large and dense graphs that contain frequent cyclic subgraphs, mining the complete set of frequent patterns is computationally expensive, since numerous subgraph isomorphisms are performed, making these algorithms not effective.

To overcome these limitations, we propose in this paper a *relaxation of the canonicity property*, leading to two encodings, *lower and upper encodings*, with a polynomial time complexity. The two encodings have been integrated in `Gaston`, leading to two approximate algorithms, called `Gaston-Low` and `Gaston-Up`. Because of its incomplete nature, the number of patterns discovered by `Gaston-Low` (resp. `Gaston-Up`) is a lower (resp. upper) bound of the exact number of frequent graphs. Thus, these two encodings allow one to enclose the number of frequent graphs generated by a canonical encoding. The lower and upper encodings allow two kinds of queries: lower and upper queries. The lower query uses the lower encoding and enables to get a subset of frequent patterns. And the upper query uses the upper encoding and enables to get a superset of frequent patterns.

Experiments performed on synthetic and real world datasets show that, on large and dense graphs, `Gaston-Low` and `Gaston-Up` are very effective compared to `Gaston` and `gSpan`, while on large real-life sparse graphs it remains very competitive.

This paper is organized as follows. Section 2 introduces preliminaries on graph mining. Section 3 reviews some canonical encodings. Section 4 motivates our proposals. Section 5 explains our two encodings. Section 6 is devoted to experimentations. Finally, we conclude and draw some perspectives.

2 Preliminaries

In this section we briefly review some basic concepts and fix the notations.

A *labelled graph* can be represented by a 4-tuple, $G = (V, E, L, l)$, where V is a finite set of vertices, $E \subset V \times V$ is a set of edges, L is a set of labels, and $l : V \cup E \rightarrow L$ is a function assigning a label to every element of $V \cup E$.

A *dense graph* is a graph in which the number of edges is close to the maximal number of edges. The graph density D is defined as: $D = 2|E|/(|V|(|V|-1))$. $(|V|(|V|-1)/2)$ is the number of edges in a complete graph. Clearly, the given formula of D computes the proximity of the number of edges to the maximum number of edges. A graph is *cyclic* if it has many cycles. As the number of cycles increases, the number of edges increases, and then the density of the graph increases too.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs.

- G_1 and G_2 are *isomorphic* if there is a bijection function $f : V_1 \rightarrow V_2$ satisfying: (i) $\forall\, u \in V_1, l_{G_1}(u) = l_{G_2}(f(u))$, (ii) $\forall \{u, v\} \in E(G_1), \{f(u), f(v)\} \in E(G_2)$, and (iii) $\forall \{u, v\} \in E(G_1)\ l_{G_1}(\{u, v\}) = l_{G_2}(\{f(u), f(v)\})$.
- G_2 is a *subgraph* of G_1, iff $V_2 \subset V_1$, and $E_2 \subset E_1 \wedge (\forall \{v_1, v_2\} \in E_2 \Rightarrow v1 \in V_2$ and $v_2 \in V_2)$.
- G_1 is *subgraph isomorphic* to G_2, denoted $G_1 \subseteq G_2$, if G_1 is isomorphic to a subgraph of G_2. Deciding whether a graph is subgraph isomorphic to another graph is NP-complete [14].

An *encoding function* is a function that assigns to a given graph a *code* (i.e. a sequence of bits, a string, or a sequence of numbers).

Definition 1 (Canonical encoding). *The encoding function ϕ is canonical when for any two graphs G_1, G_2: G_1 and G_2 are isomorphic iff $\phi(G_1) = \phi(G_2)$.*

Given a transaction database $\mathcal{D}$ which contains a family of graphs. The *frequency* of a graph G in $\mathcal{D}$ is defined by $freq(G, \mathcal{D}) = \#\{G_{\mathcal{D}} \in \mathcal{D} \,|\, G \subseteq G_{\mathcal{D}}\}$. The *support* of a graph is defined by

$$support(G, \mathcal{D}) = \frac{freq(G, \mathcal{D})}{|\mathcal{D}|}$$

Let s_{min} be some predefined minimum support. The *Frequent Subgraph Discovery Problem $FSDP$* consists in finding connected undirected graphs G' that are subgraphs of at least $(s_{min} \times |\mathcal{D}|)$ graphs of $\mathcal{D}$:

$$FSDP(\mathcal{D}, s_{min}) = \{G' | support(G', \mathcal{D}) \geq s_{min}\}.$$

The query $FSDPquery(\mathcal{D}, s_{min}, \phi)$ consists in finding graphs $FSDP(\mathcal{D}, s_{min})$ by using some encoding function ϕ.

Generally, we can distinguish between the algorithms for computing frequent subgraphs according to the way the three following problems are handled:

1. **Candidates generation problem:** The candidates are initialized with frequent edges (1-candidates). The k-candidates (i.e., having k edges) are generated, by adding one edge to each $(k-1)$-candidate. This process can be done with a breadth-first strategy as well as a depth-first strategy.
2. **Subgraph encoding problem:** A canonical encoding is assigned to each generated graph. Verifying that the candidate is new, consists in checking that its encoding does not belong to the encodings of the generated candidates. This paper contributes in this step (see section 5).

3. **Frequency computation problem:** Once a new candidate is generated, we have to compute its frequency. It can be achieved by finding all the transactions of the database that contain this new candidate.

3 Canonical encoding

Developing algorithms that can efficiently compute the canonical encoding is critical to ensure the scalability to very large and dense graph datasets. It is not proven if the canonical encoding of graphs is in the class of NP-complete problems, nor in polynomial class [4].

The encoding function of the FSG algorithm [9] is based on exploring the adjacency matrices of the considered graph. To get the canonical encoding, gSpan algorithm [15] performs various DFS searches on the graph. In [11], the authors have proposed an efficient miner algorithm in which they used an appropriate canonical encoding for the three graph structures: paths, trees and cycles. Paths and trees are encoded efficiently in polynomial time. But for cyclic graphs, the encoding is of exponential nature. As pointed out in [12], the more there are cycles in the graph, the more its encoding is expensive. In fact, for cyclic graphs, the complexity of the encoding in Gaston is $O(|C|(2|C|)!|V||E|^{|C|})$, where E represents the number of edges in the cyclic graph, $|C|$ is the minimal number of edges to remove to obtain a spanning tree. Clearly, $|C|$ is strongly related to the number of cycles in the graph.

So, to overcome this limitation, we propose in this paper two efficient encodings for cyclic graphs: *lower-encoding* and *upper-encoding* (see section 5). Lower-encoding allows one to generate a significant subset of the frequent cyclic graphs, whereas the upper-encoding allows one to generate all frequent graphs with duplicates. Roughly speaking, these two encodings define an interval enclosing the exact frequent graphs. The following section shows the effectiveness of our approach for dense graph datasets.

4 Motivating example

Let us consider the graph database shown in Figure 1. When executing the state of the art `Gaston` miner [11] to extract all subgraphs with a support $s_{min} = 50\%$, it takes 10 minutes, and the number of frequent cyclic graphs found is $1,334,095$, which is very considerable compared to the size of the database. This is essentially due to the canonical encoding which is expensive for these graphs. Our main idea is to use a non-canonical encodings, called *lower and upper encodings*, to enclose the complete frequent subgraphs, whilst ensuring reasonable computing times. Applying our lower and upper encodings on the example of Figure 1 gives rise to the following results: (i) the lower query enables to generate $1,309,066$ cyclic graphs in 1.1 minutes. Moreover, the percentage of missed frequent cyclic subgraphs is about 1.87%, which remains reasonably small compared to the saved CPU time, (ii) the upper query allows one to generate $1,468,364$ subgraphs in 2.43 minutes.

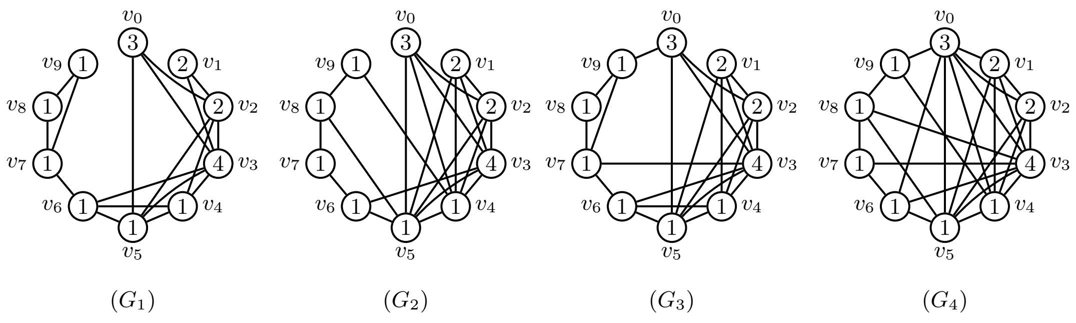


Fig. 1. An example of a dense graph transaction database

5 Lower and upper queries

Due to the incomplete nature of the lower encoding, the number of discovered frequent graphs is a lower bound of the exact number of frequent graphs, and the converse for the upper encoding. These two encodings allow two kinds of queries on the transaction database: the lower query and the upper query.

5.1 A lower-bounding graph encoding and the lower query

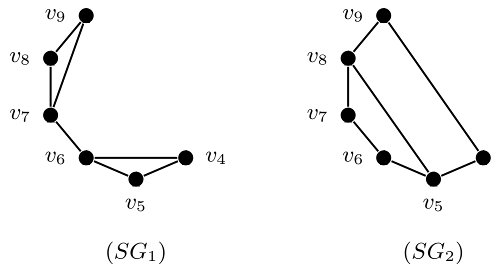
Definition 2 (Lower Encoding).
Let ϕ_L be an encoding function. ϕ_L is lower if the following property holds:
Given two graphs G_1 and G_2, if G_1 and G_2 are isomorphic then $\phi_L(G_1) = \phi_L(G_2)$.

Definition 2 establishes the main property of a lower encoding. Based on this definition, we propose an instance of a lower encoding, called `Node-Seq`, that fully makes use of different invariants (i.e., degrees and labels). It is built by concatenating codes of all vertices of the graph, `vcode`(v_i), $i = 1..|V|$, according to a lexicographic order.

Definition 3 (Node-Seq). *The node sequence code of $G = (V, E)$ is obtained by concatenating codes of all its vertices:* `Node-Seq`(G) = (`vcode`(v_1), ..., `vcode`$(v_{|V|})$)*, where* `vcode`$(v_i) <_l$ `vcode`(v_{i+1})*, and the relation $<_l$ defines a lexicographic order among vertices. The vertex code is defined by* `vcode`$(v_i) =$ $(deg(v_i), lab(v_i), \langle lab(e_{i1}), deg(v_{i1}), lab(v_{i1})\rangle, \ldots, \langle lab(e_{im}), deg(v_{im}), lab(v_{im})\rangle)$*, where:*

- *$deg(v_i)$ is the degree of v_i,*
- *$\langle lab(e_{ij}), deg(v_{ij}), lab(v_{ij})\rangle$ is defined as follows: $lab(e_{ij})$ is the label of the j^{the} edge incident to v_i, and m is the number of its incident edges; $deg(v_{ij})$ (resp. $lab(v_{ij})$) is the degree (resp. label) of the vertex associated to the end-point of the edge e_{ij} incident to v_i.*

Let us note that edges e_{ij} incident to v_i are ordered according to their labels. If these labels are equal, we consider the degrees and the labels of their endpoint vertices.



$v_i \in SG_1$	$v_i \in SG_2$	`vcode`(v_i)
v_4	v_4	$(2, 1, 1, 2, 1, 1, 3, 1)$
v_5	v_6	$(2, 1, 1, 2, 1, 1, 3, 1)$
v_6	v_5	$(3, 1, 2, 1, 1, 2, 1, 1, 3, 1)$
v_7	v_8	$(3, 1, 2, 1, 1, 2, 1, 1, 3, 1)$
v_8	v_7	$(2, 1, 1, 2, 1, 1, 3, 1)$
v_9	v_9	$(2, 1, 1, 2, 1, 1, 3, 1)$

$$\texttt{Node-Seq}(SG_1) = \texttt{Node-Seq}(SG_2) = (2,1,1,2,1,1,3,1)(2,1,1,2,1,1,3,1)(2,1,1,2,1,1,3,1)$$
$$(2,1,1,2,1,1,3,1)(3,1,2,1,1,2,1,1,3,1)(3,1,2,1,1,2,1,1,3,1)$$

Fig. 2. Example of two non-isomorphic graphs having the same encoding Node-Seq

Proposition 1. `Node-Seq` *is a lower encoding and it computes a lower bound of the exact number of frequent graphs. It can be achieved in $O(n\ m\ log(m) + n\ log(n))$ in the worst case, where $n = |V|$ and $m = |E|$.*

Sketch of the proof: Since `Node-seq` encoding uses only labels, degrees, and a lexicographic sorting, it is necessarily a lower-encoding. Its non-canonicity is stated by the counter-example given in Figure 2. In fact, (SG_1) and (SG_2) are non-isomorphic, but have the same `Node-seq` encoding. Traversing the vertices is done in $O(n)$. The incident edges of each vertex are enumerated and sorted at most $O(m\ log(m))$ times. The whole n encodings are sorted. Thus, the time complexity is $O(n\ m\ log(m) + n\ log(n))$.□

Let be ϕ_L some lower encoding function, and ϕ_C a canonical encoding. The lower query $FSDPquery(\mathcal{D}, s_{min}, \phi_L)$ computes a subset of frequent subgraphs:

$$FSDPquery(\mathcal{D}, s_{min}, \phi_L) \subseteq FSDPquery(\mathcal{D}, s_{min}, \phi_C).$$

5.2 An upper-bounding graph encoding and the upper query

Definition 4 (Upper Encoding). *Let ϕ_U be an encoding function. ϕ_U is upper if the following property holds: Given two graphs G_1 and G_2, if G_1 and G_2 are not isomorphic then $\phi_U(G_1) \neq \phi_U(G_2)$.*

We propose an upper encoding function which is inspired from the lower encoding, in three steps:

1. Let n be the number of vertices of the considered graph $G(V, E)$ to encode. We associate to each vertex vi a code $lst(v_i)$ containing respectively, its

degree, its label, and a sorted list of degrees and labels of its incident vertices. Then, the n vertices are sorted according to their codes $lst(v_i), i = 1..n$. Following the given order, the first vertex receives identifier 1, the second vertex identifier 2, and so on until the last vertex which receives the identifier n.

2. Each vertex v_i is represented with the following code `vcode`$(v_i) = (Id(v_i)$, $lab(v_i)$, $\langle lab(e_{i1}), Id(v_{i1})\rangle$, $\ldots, \langle lab(e_{im}), Id(v_{im})\rangle)$, where:
   - $Id(v_i)$ is the identifier of v_i, generated in the previous step,
   - $lab(e_{ij})$ is the label of the j^{the} edge incident to v_i, and m is the number of its incident edges,
   - The incident edges $e_{ij}, i = 1..m$ are ordered according to their labels and degrees.
3. Finally, the upper encoding, called `ID-Seq` , is built by concatenating codes of all vertices of the graph, according to the order found in the first step.

Proposition 2. `ID-Seq` *is an upper encoding and it computes an upper bound of the exact number of frequent graphs. It can be achieved in* $O(n\ log(n) + 2n\ m\ log(m))$ *in the worst case, where* $n = |V|$ *and* $m = |E|$.

Sketch of the proof: This upper encoding uses the identifiers of the vertices, which allows one to encode the topological structure of the graph. That is why we can not find two non isomorphic graphs with the same `ID-Seq`. The proof of the complexity is the same as the one given for `Node-Seq` encoding except that we should take additionally into account the first step. For each vertex v_i, the sorting operation to generate $lst(v_i)$ requires $O(m\ log(m))$. The complexity of the first step is $n\ m\ log(m)$, which is the additional processing time compared to the complexity of the lower encoding `Node-Seq`. □

Since assigning identifiers to vertices may be done in different ways, two isomorphic graphs can be explored multiple times giving rise to different `ID-Seq` encodings. Thus, the way the vertices are ordered and then numbered, is essential to avoid duplicates. Our vertices ordering seems a good compromise in our experimentations. But, if we want a canonical encoding, we should explore all of the orderings, which is clearly of exponential nature.

Let be ϕ_U some upper encoding function, and ϕ_C a canonical encoding. As stated for the lower encoding functions, the upper query $FSDPquery(\mathcal{D}, s_{min}, \phi_U)$ computes a superset of frequent subgraphs:

$$FSDPquery(\mathcal{D}, s_{min}, \phi_C) \subseteq FSDPquery(\mathcal{D}, s_{min}, \phi_U).$$

6 Experimental evaluation

In this section, we study the performance of the proposed encodings. We used both real and synthetic datasets. For comparison, we considered two algorithms, `gSpan` and `Gaston`.

We have integrated our lower and upper encodings in the `Gaston` algorithm to encode cyclic graphs through `Gaston-Low` [1] and `Gaston-Up` respectively. `Gaston-Low` enables to compute a lower query with `Node-Seq` encoding, whereas `Gaston-Up` an upper query with `ID-Seq` encoding. First, we generated a series of synthetic graph datasets with different settings of parameters of our graph generator *DenseGraphGen*. We study the influence of different parameters (density, frequency and the number of edge and node labels) on the performances of our encodings on synthetic datasets and we compare our results with those obtained by `Gaston` and `gSpan` (Section 6.1). Second, we compare the results of the four algorithms on real datasets (Section 6.2).

For each experiment, we report the CPU time (in seconds), the number of frequent cyclic graphs and their densities. For all experiments, we impose a time limit of $3,600$ seconds. When an algorithm cannot complete the extraction within the time limit, it will be indicated by the symbol $(-)$ in the table. All experiments were conducted on 2.50GHz Intel core i5-321M machines with 8GB main memory, running the Linux operating system.

6.1 Performances evaluation on synthetic datasets

The synthetic datasets are generated using our graph generator `DenseGraphGen`. The datasets are obtained as follows: First, we generate a dense graph G_D to guarantee that potentially frequent subgraphs are enough dense. Second, we generate $|N|$ transactions. Each transaction uses G_D as a kernel. Third, to make different all the transactions of the dataset, for each transaction, we add $|V|$ nodes and $|E|$ edges connecting randomly nodes of V with those of G_D. Then, $|EI|$ edges are added randomly between nodes of G_d. Finally, $|L_v|$ and $|L_e|$ labels are chosen randomly.

GD. Name	cyclic0	cyclic1	cyclic2	cyclic3	cyclic4	cyclic5	cyclic6	cyclic7	cyclic8
Avg. #edges	38	50	62	73	85	96	107	118	134
Avg. density	0.2	0.26	0.32	0.38	0.44	0.50	0.56	0.62	0.70

Table 1. Characteristics of the graph datasets used for our experiments: $|N| = 1,000$, $|V| = 10$, $|EI| = 5$, and $|Lv| = |Le| = 10$.

We generated a series of graph datasets using different graph kernels (with different densities: $|E|$ is varying). Such a choice enables us to obtain increasing cyclic graphs in each dataset. Their main characteristics are given in Table 1.

6.1.1 Influence of the density Table 2 compares the performances of the four algorithms on the datasets illustrated in Table 1, with $s_{min} = 100\%$. In all of these experiments, $|N| = 1,000$, $|V| = 10$, $|EI| = 5$, $|Lv| = |Le| = 10$.

From table 2, we can draw the following remarks. First, for small values of the density, `Gaston-Low`, `Gaston-Up` and `Gaston` perform quite similarly both in terms of runtime and number of frequent cyclic subgraphs discovered. Second,

[1] The implementation of Gaston-LB and Gaston-UB can be downloaded from `https://sites.google.com/site/akemmar/LUmining`

	runtime (s)				# cyclic Gr.		
GD. Name	`Gaston-Low`	`Gaston`	`Gaston-Up`	`gSpan`	`Gaston-Low`	`Gaston`	`Gaston-Up`
cyclic0	0.24	**0.2**	0.3	22.03	**1,496**	**1,496**	**1,496**
cyclic1	1.19	**0.88**	1.6	117.67	**10,729**	**10,729**	**10,729**
cyclic2	5.73	**4.89**	8.03	–	**59,996**	**59,997**	**59,997**
cyclic3	**13.03**	**13.02**	18.04	–	**132,915**	**132,915**	**132,915**
cyclic4	**58.38**	115.93	80.35	–	**619,081**	**619,081**	620,446
cyclic5	**122.39**	387.97	**168.19**	–	1,221,435	1,223,554	1,223,862
cyclic6	**303.46**	1,516.99	**409.4**	–	2,999,054	2,999,054	2,999,054
cyclic7	**744.19**	–	**1,045.94**	–	6,928,308	–	6,928,308
cyclic8	–	–	–	–	–	–	–

Table 2. Characteristics of the graph datasets used for our experiments: $|N| = 1,000$, $|V| = 10$, $|EI| = 5$, and $|Lv| = |Le| = 10$. Performances of `Gaston`, `gSpan`, `Gaston-Low` and `Gaston-Up` on different synthetic datasets of increasing density ($s_{min} = 100\%$).

`gSpan` is not competitive, even for very low densities. Third, for high values of the density (≥ 0.44), the difference in performance between `Gaston-Low`, `Gaston-Up` and `Gaston` widens considerably: `Gaston-Low` (resp. `Gaston-Up`) is up to 5 (resp. 4) times faster than `Gaston`. Moreover, the percentage of frequent cyclic subgraphs omitted by `Gaston-Low` is very negligible. For all datasets, `Gaston-Low` finds at least 99.82% of all frequent cyclic subgraphs. For the *cyclic*7 dataset, `Gaston` is not able to complete the extraction of all frequent subgraphs in one hour. Indeed, with the increase of density value, the search space of graph isomorphism for the candidate generation and the running time increases drastically and `Gaston` spends more time to encode these cyclic graphs. This is not the case for `Gaston-Low`, which requires less time thanks to its polynomial time encoding. For `Gaston-Up`, the number of frequent cyclic graphs is the same for all datasets, except for *cyclic*4 and *cyclic*5 for which the percentage of duplicates is at most 0.2%. From these results, the two encodings succeeded to enclose tightly the exact number of subgraphs within a very competitive runtime.

6.1.2 Influence of the support threshold

Table 3 shows the results obtained by the three algorithms on the dataset *cyclic*5 with values of s_{min} ranging from 100% to 40%. The results of `gSpan` are not reported since it fails to complete the extraction of all frequent subgraphs in the time limit. As we can see, as s_{min} decreases, the running time of `Gaston` increases drastically and becomes prohibitively expensive below a certain threshold, while `Gaston-Low` remains effective: it outperforms `Gaston` by a factor from 2 to 3. Moreover, `Gaston-Low` finds a significant number of frequent cyclic subgraphs with a percentage of missed subgraphs of at most 0.17%. For `Gaston-Up`, the upper encoding generates more graphs than `Gaston`, particularly for small values of s_{min}. However, `Gaston-Up` takes less time than `Gaston` thanks to its polynomial time encoding.

6.1.3 Influence of the number of edge and node labels

For these series of experiments, we considered the same number of labels for nodes and edges, given by the parameter $|L|$. Table 4 shows the results obtained by the three

	runtime (s)			# cyclic Gr.		
s_{min}(%)	`Gaston-Low`	`Gaston`	`Gaston-Up`	`Gaston-Low`	`Gaston`	`Gaston-Up`
100	**186.05**	476.94	**210.78**	1,221,435	1,223,554	1,223,862
80	**163.44**	464.55	**190.43**	1,225,059	1,225,678	1,225,830
70	**248.5s**	584.49	**257.13**	**1,484,175**	**1,484,175**	1,805,651
65	**219.72s**	494.67	**252.62**	**1,228,323**	**1,228,323**	1,517,283
60	**482.45s**	842.23	**554.09**	**1,819,606**	**1,819,606**	3,132,861
50	**1,743.58**	2,156.34	**1,787.1**	**4,104,185**	**4,104,185**	**6,671,993**
40	–	–	–	–	–	–

Table 3. Comparing the performances of `Gaston`, `Gaston-Low` and `Gaston-Up` on the *cyclic*5 dataset for different minimum support threshold s_{min}.

algorithms on the dataset *cyclic*6 with values of $|L|$ ranging from 1 to max. $|L| = max$ means that all node and edge labels are different.

Algorithm	runtime (s)			# cyclic Gr.		
GD. Name	`Gaston-Low`	`Gaston`	`Gaston-Up`	`Gaston-Low`	`Gaston`	`Gaston-Up`
cyclic6-Lmax	**299.9**	1,679.33	**496.6**	3,200,963	3,200,963	3,200,963
cyclic6-L20	**279.09**	1,489.16	**432.33**	3,200,963	3,200,963	3,200,963
cyclic6-L15	**291.87**	1,488.12	**518.57**	3,200,963	3,200,963	3,200,963
cyclic6-L10	**287.32**	1,494.4	**503.14**	3,200,963	3,200,963	3,200,963
cyclic6-L5	**650.27**	–	**641.33**	3,157,417	–	3,171,563
cyclic6-L3	**1,704.35**	2,939.96	1,933.91	3,057,301	3,057,301	3,192,363
cyclic6-L2	–	–	–	–	–	–
cyclic6-L1	–	–	–	–	–	–

Table 4. Comparing the performances of `Gaston`, `Gaston-Low` and `Gaston-Up` on the *cyclic*6 dataset for different values of labels. $S_{min} = 100\%$, $|N| = 1,000$, $|V| = 10$, $|E| = 5$ and $|EI| = 5$.

From the results of Table 4, we can observe three interesting points. First, as $|L|$ increases, the overall running time of the three algorithms decreases. Indeed, the higher this value, the less the number of isomorphisms are performed, leading to fast candidate generation. Second, compared with `Gaston`, `Gaston-Low` performs faster. For $|L| \geq 10$, `Gaston-Low` finds the exact number of frequent cyclic subgraphs and it is up to 5 orders of magnitude faster than `Gaston`. For $|L| = 5$, `Gaston` cannot complete the extraction of all frequent subgraphs in the time limit (i.e. one hour), while `Gaston-Low` finds a great number of frequent cyclic subgraphs in a reasonable amount of time. Third, again, `Gaston-Up` clearly outperforms `Gaston`. Moreover, it finds the exact number of frequent cyclic subgraphs, except for `cyclic6-L3` where the percentage of duplicates is at most 4%.

6.2 Real world datasets

To study the behaviour of our encoding on datasets which are not dense and contain a few number of frequent cyclic graphs, we performed experiments on real world datasets coming from the biological domain encoding molecules. Each molecule corresponds to a graph where atoms are represented using nodes and bonds between them are represented by edges. They are obtained from different sites: (a) The National Cancer Institute (NCI); (b) Developmental Therapeutics Program (DTP); (c) The Prediction of Ames Mutagenicity(PAM) [2].

Algorithm						runtime (s)				# cyclic Gr.		
GD. Name	$\|N\|$	$\|V\|$	$\|Lv\|$	$\|Le\|$	D	`Gaston-Low`	`Gaston`	`Gaston-Up`	`gSpan`	`Gaston-Low`	`Gaston`	`Gaston-Up`
CAN2DA99	5,210	25	82	3	0.06	**1.38**	**1.34**	**1.50**	25.41	**51** (13/2)	**52** (13/2)	139
Open-2012-05-01	189	28	20	3	0.09	1.63	**0.41**	5.12	–	8,098 (23/2)	10,532 (23/2)	59,333
new-oct99 -aug00-3D	1,170	50	23	3	0.08	**2.33**	**2.26**	3.03	104.15	632 (15/7)	721 (15/9)	2,195
2DA99 2012-05-01	108,954	19	131	3	0.06	9.21	**8.86**	11.15	146.70	**17** (10/1)	**17** (10/1)	49
aug00-2D	188,937	38	186	3	0.06	**208.57**	**207.66**	**246.58**	–	239 (12/21)	271 (12/21)	909
Chemical-340	340	26	161	3	0.13	**0.02**	**0.02**	**0.04**	0.62	**64** (11/10)	**65** (11/10)	210
Compound-422	422	39	21	4	0.10	**0.44**	**0.22**	**0.92**	7.68	**2,529** (20/1)	**2,533** (20/1)	3,794
mechset3d	1,536	46	42	4	0.03	**2.77**	**2.77**	**3.5**	100.00	513 (13/10)	573 (13/11)	1,788
cans03sd	4,2247	26	161	3	0.13	**9.57**	**9.43**	**10.72**	131.08	**39** (13/1)	**39** (13/1)	107
divii	29	22	8	2	0.09	75.98	**16.26**	261.68	–	411,922 (28/2)	429,009 (28/2)	1,198,933
N6512	6,512	17	28	4	0.32	**1.65**	**1.62**	**2.42**	34.40	**66** (14/7)	**72** (14/8)	220

Table 5. Comparing the performances of `Gaston-Lower`, `Gaston-Upper`, `Gaston` and `gSpan` on real world datasets for $s_{min} = 10\%$. $|N|$: number of graphs, $|V|$: the average number of vertices and D: the average density of the dataset.

Table 5 shows the runtime and the number of frequent patterns found by the four algorithms for different datasets. We also report for each algorithm, the size of the largest frequent patterns obtained (number of its edges) and the number of its occurrence (in parenthesis). Comparing the relative performance of `Gaston-Low` and `Gaston` on the NCI datasets, we can see that overall, they perform quite similarly in terms of runtime even on large datasets (i.e. `NCI-2DA99` and `NCI-aug00-2D`). Moreover, the percentage of patterns missed by `Gaston-Low` remains reasonably low, except for `NCI-Open` dataset where this percentage is about 23%. Even though `Gaston-Low` is incomplete, the size of the greatest subgraph is practically the same than the one found by `Gaston`. Despite the great number of duplicates generated, `Gaston-Up` remains comparable with `Gaston-Low` and `Gaston` in terms of runtime. Finally, compared to `gSpan`, the three other algorithms achieve better performance, with a factor from 16 to 45. The same conclusions can be drawn for DTP and PAM datasets.

7 Conclusion

In this paper, we have proposed a relaxation of the canonicity property, leading to lower and upper encodings, giving rise to lower and upper queries, for the

[2] (a) `http://cactus.nci.nih.gov/download/nci/`; (b) `http://dtp.nci.nih.gov/`; (c) `http://doc.ml.tu-berlin.de/toxbenchmark/index.html#v2`

frequent subgraph discovery problem. Our encodings can be achieved in a polynomial time complexity. The two encodings have been integrated in the state of the art `Gaston` miner. Experiments we performed on synthetic databases as well as on a real world dataset from the biological domain show that our lower encoding is very effective on dense graphs: the lower query is able to extract a larger number of frequent cyclic subgraphs in a reasonable amount of time, while `Gaston` needs much more time to extract the complete set of frequent subgraphs. On the other hand, the upper query allows one to have a tight and valid approximation of the missed graphs using the upper-encoding. As future works, we intend to improve our lower and upper encodings in order to enhance their performances on non-dense graphs. We should also investigate more deeply real world datasets, in order to show the effectiveness of our approach on dense graphs.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of VLDB'94*, pages 487–499. Morgan Kaufmann, 1994.
2. V. G. Blinova, D. A. Dobrynin, V. K. Finn, Sergei O. Kuznetsov, and E. S. Pankratova. Toxicology analysis by means of the jsm-method. *Bioinformatics*, 19(10):1201–1207, 2003.
3. C. Borgelt and M. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings of ICDM '02*, pages 51–58, Washington, DC, USA, 2002. IEEE Computer Society.
4. Scott Fortin. The graph isomorphism problem. Technical report, TR-96-20, Departement of computer science, University of Alberta, Canada, 1996.
5. J. Han and J. Pei. Mining frequent patterns by pattern-growth: methodology and implications. *SIGKDD Explor. Newsl.*, 2(2):14–20, December 2000.
6. J. Huan, D. Bandyopadhyay, W. Wang, J. Snoeyink, J. Prins, and A. Tropsha. Comparing graph representations of protein structure for mining family-specific residue-based packing motifs. *J. of Computational Biology*, 12(6):657–671, 2005.
7. J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of I ICDM '03*, pages 549–552, Washington, DC, USA, 2003. IEEE Computer Society.
8. A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of ECML-PKDD'00*, pages 13–23, London, UK, 2000. Springer-Verlag.
9. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of ICDM'01*, pages 313–320, Washington, DC, USA, 2001. IEEE Computer Society.
10. Sergei O. Kuznetsov and Mikhail V. Samokhin. Learning closed sets of labeled graphs for chemical applications. In *Proceedings of the 15th international conference on Inductive Logic Programming*, ILP'05, pages 190–208, Berlin, Heidelberg, 2005. Springer-Verlag.
11. S. Nijssen and J. Kok. The gaston tool for frequent subgraph mining. *Electron. Notes Theor. Comput. Sci.*, 127:77–87, March 2005.
12. Siegfried Nijssen. *Mining Structured Data.* PhD thesis, Leiden University, 2006.
13. G. Poezevara, B. Cuissart, and B. Crémilleux. Extracting and summarizing the frequent emerging graph patterns from a dataset of graphs. *J. Intell. Inf. Syst.*, 37(3):333–353, 2011.

14. Ronald C. Read and Derek G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1:339–363, 1977.
15. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. *Order A Journal On The Theory Of Ordered Sets And Its Applications*, 02:721–724, 2002.

API design for machine learning software: experiences from the scikit-learn project

Lars Buitinck[1], Gilles Louppe[2], Mathieu Blondel[3], Fabian Pedregosa[4], Andreas C. Müller[5], Olivier Grisel[6], Vlad Niculae[7], Peter Prettenhofer[8], Alexandre Gramfort[4,9], Jaques Grobler[4], Robert Layton[10], Jake Vanderplas[11], Arnaud Joly[2], Brian Holt[12], and Gaël Varoquaux[4]

[1] ILPS, Informatics Institute, University of Amsterdam
[2] University of Liège
[3] Kobe University
[4] Parietal, INRIA Saclay
[5] University of Bonn
[6] Independent consultant
[7] University of Bucharest
[8] Ciuvo GmbH
[9] Institut Mines-Telecom, Telecom ParisTech, CNRS LTCI
[10] University of Ballarat
[11] University of Washington
[12] Samsung Electronics Research Institute

Abstract. *scikit-learn* is an increasingly popular machine learning library. Written in Python, it is designed to be simple and efficient, accessible to non-experts, and reusable in various contexts. In this paper, we present and discuss our design choices for the application programming interface (API) of the project. In particular, we describe the simple and elegant interface shared by all learning and processing units in the library and then discuss its advantages in terms of composition and reusability. The paper also comments on implementation details specific to the Python ecosystem and analyzes obstacles faced by users and developers of the library.

1 Introduction

The *scikit-learn* project[1] (Pedregosa et al., 2011) provides an open source machine learning library for the Python programming language. The ambition of the project is to provide efficient and well-established machine learning tools within a programming environment that is accessible to non-machine learning experts and reusable in various scientific areas. The project is not a novel domain-specific language, but a library that provides machine learning idioms to a general-purpose high-level language. Among other things, it includes classical learning algorithms, model evaluation and selection tools, as well as preprocessing procedures. The library is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings.

[1] http://scikit-learn.org

scikit-learn is a library, i.e. a collection of classes and functions that users import into Python programs. Using *scikit-learn* therefore requires basic Python programming knowledge. No command-line interface, let alone a graphical user interface, is offered for non-programmer users. Instead, interactive use is made possible by the Python interactive interpreter, and its enhanced replacement IPython (Perez and Granger, 2007), which offer a MATLAB-like working environment specifically designed for scientific use.

The library has been designed to tie in with the set of numeric and scientific packages centered around the NumPy and SciPy libraries. NumPy (Van der Walt et al., 2011) augments Python with a contiguous numeric array datatype and fast array computing primitives, while SciPy (Haenel et al., 2013) extends it further with common numerical operations, either by implementing these in Python/NumPy or by wrapping existing C/C++/Fortran implementations. Building upon this stack, a series of libraries called *scikits* were created, to complement SciPy with domain-specific toolkits. Currently, the two most popular and feature-complete ones are by far *scikit-learn* and *scikit-image*,[2] which does image processing.

Started in 2007, *scikit-learn* is developed by an international team of over a dozen core developers, mostly researchers from various fields (e.g., computer science, neuroscience, astrophysics). The project also benefits from many occasional contributors proposing small bugfixes or improvements. Development proceeds on GitHub[3], a platform which greatly facilitates this kind of collaboration. Because of the large number of developers, emphasis is put on keeping the project maintainable. In particular, code must follow specific quality guidelines, such as style consistency and unit-test coverage. Documentation and examples are required for all features, and major changes must pass code review by at least two developers not involved in the implementation of the proposed change.

scikit-learn's popularity can be gauged from various indicators such as the hundreds of citations in scientific publications, successes in various machine learning challenges (e.g., Kaggle), and statistics derived from our repositories and mailing list. At the time of writing, the project is watched by 1365 people and forked 693 times on GitHub; the mailing list receives more than 300 mails per month; version control logs show 183 unique contributors to the codebase and the online documentation receives 37,000 unique visitors and 295,000 pageviews per month.

Pedregosa et al. (2011) briefly presented *scikit-learn* and benchmarked it against several competitors. In this paper, we instead present an in-depth analysis of design choices made when building the library, detailing how we organized and operationalized common machine learning concepts. We first present in section 2 the central application programming interface (API) and then describe, in section 3, advanced API mechanisms built on the core interface. Section 4 briefly describes the implementation. Section 5 then compares *scikit-learn* to other major projects in terms of API. Section 6 outlines some of the objectives

[2] http://scikit-image.org
[3] https://github.com/scikit-learn

for a *scikit-learn* 1.0 release. We conclude by summarizing the major points of this paper in section 7.

2 Core API

All objects within *scikit-learn* share a uniform common basic API consisting of three complementary interfaces: an *estimator* interface for building and fitting models, a *predictor* interface for making predictions and a *transformer* interface for converting data. In this section, we describe these three interfaces, after reviewing our general principles and data representation choices.

2.1 General principles

As much as possible, our design choices have been guided so as to avoid the proliferation of framework code. We try to adopt simple conventions and to limit to a minimum the number of methods an object must implement. The API is designed to adhere to the following broad principles:

Consistency. All objects (basic or composite) share a consistent interface composed of a limited set of methods. This interface is documented in a consistent manner for all objects.

Inspection. Constructor parameters and parameter values determined by learning algorithms are stored and exposed as public attributes.

Non-proliferation of classes. Learning algorithms are the only objects to be represented using custom classes. Datasets are represented as NumPy arrays or SciPy sparse matrices. Hyper-parameter names and values are represented as standard Python strings or numbers whenever possible. This keeps *scikit-learn* easy to use and easy to combine with other libraries.

Composition. Many machine learning tasks are expressible as sequences or combinations of transformations to data. Some learning algorithms are also naturally viewed as meta-algorithms parametrized on other algorithms. Whenever feasible, such algorithms are implemented and composed from existing building blocks.

Sensible defaults. Whenever an operation requires a user-defined parameter, an appropriate default value is defined by the library. The default value should cause the operation to be performed in a sensible way (giving a baseline solution for the task at hand).

2.2 Data representation

In most machine learning tasks, data is modeled as a set of variables. For example, in a supervised learning task, the goal is to find a mapping from input variables $X_1, \ldots X_p$, called features, to some output variables Y. A sample is then defined as a pair of values $([x_1, \ldots, x_p]^{\mathrm{T}}, y)$ of these variables. A widely used representation of a dataset, a collection of such samples, is a pair of matrices with numerical values: one for the input values and one for the output values. Each row of these matrices corresponds to one sample of the dataset and each column to one variable of the problem.

In *scikit-learn*, we chose a representation of data that is as close as possible to the matrix representation: datasets are encoded as NumPy multidimensional arrays for dense data and as SciPy sparse matrices for sparse data. While these may seem rather unsophisticated data representations when compared to more object-oriented constructs, such as the ones used by Weka (Hall et al., 2009), they bring the prime advantage of allowing us to rely on efficient NumPy and SciPy vectorized operations while keeping the code short and readable. This design choice has also been motivated by the fact that, given their pervasiveness in many other scientific Python packages, many scientific users of Python are already familiar with NumPy dense arrays and SciPy sparse matrices. From a practical point of view, these formats also provide a collection of data loading and conversion tools which make them very easy to use in many contexts. Moreover, for tasks where the inputs are text files or semi-structured objects, we provide *vectorizer* objects that efficiently convert such data to the NumPy or SciPy formats.

For efficiency reasons, the public interface is oriented towards processing batches of samples rather than single samples per API call. While classification and regression algorithms can indeed make predictions for single samples, *scikit-learn* objects are not optimized for this use case. (The few online learning algorithms implemented are intended to take mini-batches.) Batch processing makes optimal use of NumPy and SciPy by preventing the overhead inherent to Python function calls or due to per-element dynamic type checking. Although this might seem to be an artifact of the Python language, and therefore an implementation detail that leaks into the API, we argue that APIs should be designed so as not to tie a library to a suboptimal implementation strategy. As such, batch processing enables fast implementations in lower-level languages (where memory hierarchy effects and the possibility of internal parallelization come into play).

2.3 Estimators

The *estimator* interface is at the core of the library. It defines instantiation mechanisms of objects and exposes a `fit` method for learning a model from training data. All supervised and unsupervised learning algorithms (e.g., for classification, regression or clustering) are offered as objects implementing this interface. Machine learning tasks like feature extraction, feature selection or dimensionality reduction are also provided as estimators.

Estimator initialization and actual learning are strictly separated, in a way that is similar to partial function application: an estimator is initialized from a set of named constant hyper-parameter values (e.g., the C constant in SVMs) and can be considered as a function that maps these values to actual learning algorithms. The constructor of an estimator does not see any actual data, nor does it perform any actual learning. All it does is attach the given parameters to the object. For the sake of convenient model inspection, hyper-parameters are set as public attributes, which is especially important in model selection settings. For ease of use, default hyper-parameter values are also provided for all built-in estimators. These default values are set to be relevant in many common situations in order to make estimators as effective as possible *out-of-box* for non-experts.

Actual learning is performed by the `fit` method. This method is called with training data (e.g., supplied as two arrays `X_train` and `y_train` in supervised learning estimators). Its task is to run a learning algorithm and to determine model-specific parameters from the training data and set these as attributes on the estimator object. As a convention, the parameters learned by an estimator are exposed as public attributes with names suffixed with a trailing underscore (e.g., `coef_` for the learned coefficients of a linear model), again to facilitate model inspection. In the partial application view, `fit` is a function from data to a model of that data. It always returns the estimator object it was called on, which now serves as a model of its input and can be used to perform predictions or transformations of input data.

From the start, the choice to let a single object serve dual purpose as estimator and model has mostly been driven by usability and technical considerations. From the user point of view, having two coupled instances (i.e., an estimator object, used as a factory, and a model object, produced by the estimator) indeed decreases the ease of use and is also more likely to unnecessarily confuse newcomers. From the developer point of view, decoupling estimators from models also creates parallel class hierarchies and increases the overall maintenance complexity of the project. For these practical reasons, we believe that decoupling estimators from models is not worth the effort. A good reason for decoupling however, would be that it makes it possible to ship a model in a new environment without having to deal with potentially complex software dependencies. Such a feature could however still be implemented in *scikit-learn* by making estimators able to export a fitted model, using the information from its public attributes, to an agnostic model description such as PMML (Guazzelli et al., 2009).

To illustrate the initialize-fit sequence, let us consider a supervised learning task using logistic regression. Given the API defined above, solving this problem is as simple as the following example.

```python
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(penalty="l1")
clf.fit(X_train, y_train)
```

In this snippet, a `LogisticRegression` estimator is first initialized by setting the `penalty` hyper-parameter to `"l1"` for ℓ_1 regularization. Other hyper-parameters (such as `C`, the strength of the regularization) are not explicitly given and thus set to the default values. Upon calling `fit`, a model is learned from the training arrays `X_train` and `y_train`, and stored within the object for later use. Since all estimators share the same interface, using a different learning algorithm is as simple as replacing the constructor (the class name); to build a random forest on the same data, one would simply replace `LogisticRegression(penalty="l1")` in the snippet above by `RandomForestClassifier()`.

In *scikit-learn*, classical learning algorithms are not the only objects to be implemented as estimators. For example, preprocessing routines (e.g., scaling of features) or feature extraction techniques (e.g., vectorization of text documents) also implement the *estimator* interface. Even stateless processing steps, that do not require the `fit` method to perform useful work, implement the estimator interface. As we will illustrate in the next sections, this design pattern is indeed of prime importance for consistency, composition and model selection reasons.

2.4 Predictors

The *predictor* interface extends the notion of an estimator by adding a `predict` method that takes an array `X_test` and produces predictions for `X_test`, based on the learned parameters of the estimator (we call the input to `predict` "`X_test`" in order to emphasize that `predict` generalizes to new data). In the case of supervised learning estimators, this method typically returns the predicted labels or values computed by the model. Continuing with the previous example, predicted labels for `X_test` can be obtained using the following snippet:

```
y_pred = clf.predict(X_test)
```

Some unsupervised learning estimators may also implement the `predict` interface. The code in the snippet below fits a k-means model with $k = 10$ on training data `X_train`, and then uses the `predict` method to obtain cluster labels (integer indices) for unseen data `X_test`.

```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=10)
km.fit(X_train)
clust_pred = km.predict(X_test)
```

Apart from `predict`, predictors may also implement methods that quantify the confidence of predictions. In the case of linear models, the `decision_function` method returns the distance of samples to the separating hyperplane. Some predictors also provide a `predict_proba` method which returns class probabilities.

Finally, predictors must provide a `score` function to assess their performance on a batch of input data. In supervised estimators, this method takes as input arrays `X_test` and `y_test` and typically computes the coefficient of determination between `y_test` and `predict(X_test)` in regression, or the accuracy in classification. The only requirement is that the `score` method return a value that quantifies the quality of its predictions (the higher, the better). An unsupervised estimator may also expose a `score` function to compute, for instance, the likelihood of the given data under its model.

2.5 Transformers

Since it is common to modify or filter data before feeding it to a learning algorithm, some estimators in the library implement a *transformer* interface which defines a `transform` method. It takes as input some new data `X_test` and yields as output a transformed version of `X_test`. Preprocessing, feature selection, feature extraction and dimensionality reduction algorithms are all provided as transformers within the library. In our example, to standardize the input `X_train` to zero mean and unit variance before fitting the logistic regression estimator, one would write:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
```

Of course, in practice, it is important to apply the same preprocessing to the test data `X_test`. Since a `StandardScaler` estimator stores the mean and standard deviation that it computed for the training set, transforming an unseen test set `X_test` maps it into the appropriate region of feature space:

```
X_test = scaler.transform(X_test)
```

Transformers also include a variety of learning algorithms, such as dimension reduction (PCA, manifold learning), kernel approximation, and other mappings from one feature space to another.

Additionally, by leveraging the fact that `fit` always returns the estimator it was called on, the `StandardScaler` example above can be rewritten in a single line using method chaining:

```
X_train = StandardScaler().fit(X_train).transform(X_train)
```

Furthermore, every transformer allows `fit(X_train).transform(X_train)` to be written as `fit_transform(X_train)`. The combined `fit_transform` method prevents repeated computations. Depending on the transformer, it may skip only an input validation step, or in fact use a more efficient algorithm for the transformation. In the same spirit, clustering algorithms provide a `fit_predict` method that is equivalent to `fit` followed by `predict`, returning cluster labels assigned to the training samples.

3 Advanced API

Building on the core interface introduced in the previous section, we now present advanced API mechanisms for building meta-estimators, composing complex estimators and selecting models. We also discuss design choices allowing for easy usage and extension of *scikit-learn*.

3.1 Meta-estimators

Some machine learning algorithms are expressed naturally as meta-algorithms parametrized on simpler algorithms. Examples include ensemble methods which build and combine several simpler models (e.g., decision trees), or multiclass and multilabel classification schemes which can be used to turn a binary classifier into a multiclass or multilabel classifier. In *scikit-learn*, such algorithms are implemented as *meta-estimators*. They take as input an existing base estimator and use it internally for learning and making predictions. All meta-estimators implement the regular estimator interface.

As an example, a logistic regression classifier uses by default a one-vs.-rest scheme for performing multiclass classification. A different scheme can be achieved by a meta-estimator wrapping a logistic regression estimator:

```
from sklearn.multiclass import OneVsOneClassifier

ovo_lr = OneVsOneClassifier(LogisticRegression(penalty="l1"))
```

For learning, the `OneVsOneClassifier` object *clones* the logistic regression estimator multiple times, resulting in a set of $\frac{K(K-1)}{2}$ estimator objects for K-way classification, all with the same settings. For predictions, all estimators perform a binary classification and then vote to make the final decision. The snippet exemplifies the importance of separating object instantiation and actual learning.

Since meta-estimators require users to construct nested objects, the decision to implement a meta-estimator rather than integrate the behavior it implements into existing estimators classes is always based on a trade-off between generality and ease of use. Relating to the example just given, all *scikit-learn* classifiers are designed to do multiclass classification and the use of the `multiclass` module is only necessary in advanced use cases.

3.2 Pipelines and feature unions

A distinguishing feature of the *scikit-learn* API is its ability to compose new estimators from several base estimators. Composition mechanisms can be used to combine typical machine learning workflows into a single object which is itself an estimator, and can be employed wherever usual estimators can be used. In particular, *scikit-learn*'s model selection routines can be applied to composite estimators, allowing global optimization of all parameters in a complex workflow. Composition of estimators can be done in two ways: either sequentially through `Pipeline` objects, or in a parallel fashion through `FeatureUnion` objects.

`Pipeline` objects chain multiple estimators into a single one. This is useful since a machine learning workflow typically involves a fixed sequence of processing steps (e.g., feature extraction, dimensionality reduction, learning and making predictions), many of which perform some kind of learning. A sequence of N such steps can be combined into a `Pipeline` if the first $N-1$ steps are transformers; the last can be either a predictor, a transformer or both.

Conceptually, fitting a pipeline to a training set amounts to the following recursive procedure: i) when only one step remains, call its `fit` method; ii) otherwise, `fit` the first step, use it to `transform` the training set and `fit` the rest of the pipeline with the transformed data. The pipeline exposes all the methods the last estimator in the pipe exposes. That is, if the last estimator is a predictor, the pipeline can itself be used as a predictor. If the last estimator is a transformer, then the pipeline is itself a transformer.

`FeatureUnion` objects combine multiple transformers into a single one that concatenates their outputs. A union of two transformers that map input having d features to d' and d'' features respectively is a transformer that maps its d input features to $d' + d''$ features. This generalizes in the obvious way to more than two transformers. In terms of API, a `FeatureUnion` takes as input a list of transformers. Calling `fit` on the union is the same as calling `fit` independently on each of the transformers and then joining their outputs.

`Pipeline` and `FeatureUnion` can be combined to create complex and nested workflows. The following snippet illustrates how to create a complex estimator that computes both linear PCA and kernel PCA features on `X_train` (through a `FeatureUnion`), selects the 10 best features in the combination according to an ANOVA test and feeds those to an ℓ_2-regularized logistic regression model.

```
from sklearn.pipeline import FeatureUnion, Pipeline
from sklearn.decomposition import PCA, KernelPCA
from sklearn.feature_selection import SelectKBest

union = FeatureUnion([("pca", PCA()),
                      ("kpca", KernelPCA(kernel="rbf"))])

Pipeline([("feat_union", union),
          ("feat_sel", SelectKBest(k=10)),
          ("log_reg", LogisticRegression(penalty="l2"))
         ]).fit(X_train, y_train).predict(X_test)
```

3.3 Model selection

As introduced in Section 2.3, hyper-parameters set in the constructor of an estimator determine the behavior of the learning algorithm and hence the performance of the resulting model on unseen data. The problem of *model selection* is therefore to find, within some hyper-parameter space, the best combination of hyper-parameters, with respect to some user-specified criterion. For example, a decision tree with too small a value for the maximal tree depth parameter will tend to underfit, while too large a value will make it overfit.

In *scikit-learn*, model selection is supported in two distinct meta-estimators, `GridSearchCV` and `RandomizedSearchCV`. They take as input an estimator (basic or composite), whose hyper-parameters must be optimized, and a set of hyper-parameter settings to search through. This set is represented as a mapping of parameter names to a set of discrete choices in the case of grid search, which exhaustively enumerates the "grid" (cartesian product) of complete parameter combinations. Randomized search is a smarter algorithm that avoids the combinatorial explosion in grid search by sampling a fixed number of times from its parameter distributions (see Bergstra and Bengio, 2012).

Optionally, the model selection algorithms also take a cross-validation scheme and a score function. *scikit-learn* provides various such cross-validation schemes, including k-fold (default), stratified k-fold and leave-one-out. The score function used by default is the estimator's `score` method, but the library provides a variety of alternatives that the user can choose from, including accuracy, AUC and F_1 score for classification, R^2 score and mean squared error for regression.

For each hyper-parameter combination and each train/validation split generated by the cross-validation scheme, `GridSearchCV` and `RandomizedSearchCV` fit their base estimator on the training set and evaluate its performance on the validation set. In the end, the best performing model on average is retained and exposed as the public attribute `best_estimator_`.

The snippet below illustrates how to find hyper-parameter settings for an SVM classifier (SVC) that maximize F_1 score through 10-fold cross-validation on the training set.

```
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVC

param_grid = [
  {"kernel": ["linear"], "C": [1, 10, 100, 1000]},
  {"kernel": ["rbf"], "C": [1, 10, 100, 1000],
   "gamma": [0.001, 0.0001]}
]

clf = GridSearchCV(SVC(), param_grid, scoring="f1", cv=10)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

In this example, two distinct hyper-parameter grids are considered for the linear and radial basis function (RBF) kernels; an SVM with a linear kernel accepts a γ parameter, but ignores it, so using a single parameter grid would waste computing time trying out effectively equivalent settings. Additionally, we see that `GridSearchCV` has a `predict` method, just like any other classifier: it delegates the `predict`, `predict_proba`, `transform` and `score` methods to the best estimator (optionally after re-fitting it on the whole training set).

3.4 Extending scikit-learn

To ease code reuse, simplify implementation and skip the introduction of superfluous classes, the Python principle of *duck typing* is exploited throughout the codebase. This means that estimators are defined by interface, not by inheritance, where the interface is entirely implicit as far as the programming language is concerned. Duck typing allows both for extensibility and flexibility: as long as an estimator follows the API and conventions outlined in Section 2, then it can be used in lieu of a built-in estimator (e.g., it can be plugged into pipelines or grid search) and external developers are not forced to inherit from any *scikit-learn* class.

In other places of the library, in particular in the vectorization code for unstructured input, the toolkit is also designed to be extensible. Here, estimators provide hooks for user-defined code: objects or functions that follow a specific API can be given as arguments at vectorizer construction time. The library then calls into this code, communicating with it by passing objects of standard Python/NumPy types. Again, such external user code can be kept agnostic of the *scikit-learn* class hierarchy.

Our rule of thumb is that user code should not be tied to *scikit-learn*—which is a *library*, and not a *framework*. This principle indeed avoids a well-known problem with object-oriented design, which is that users wanting a "banana" should not get "a gorilla holding the banana and the entire jungle" (J. Armstrong, cited by Seibel, 2009, p. 213). That is, programs using *scikit-learn* should not be intimately tied to it, so that their code can be reused with other toolkits or in other contexts.

4 Implementation

Our implementation guidelines emphasize writing efficient but readable code. In particular, we focus on making the codebase easily maintainable and understandable in order to favor external contributions. Whenever practicable, algorithms implemented in *scikit-learn* are written in Python, using NumPy vector operations for numerical work. This allows for the code to remain concise, readable and efficient. For critical algorithms that cannot be easily and efficiently expressed as NumPy operations, we rely on Cython (Behnel et al., 2011) to achieve competitive performance and scalability. Cython is a compiled programming language that extends Python with static typing. It produces efficient C extension modules that are directly importable from the Python run-time system. Examples of algorithms written in Cython include stochastic gradient descent for linear models, some graph-based clustering algorithms and decision trees.

To facilitate the installation and thus adoption of *scikit-learn*, the set of external dependencies is kept to a bare minimum: only Python, NumPy and SciPy are required for a functioning installation. Binary distributions of these are available for the major platforms. Visualization functionality depends on Matplotlib (Hunter, 2007) and/or Graphviz (Gansner and North, 2000), but neither is required to perform machine learning or prediction. When feasible, external libraries are integrated into the codebase. In particular, *scikit-learn* includes modified versions of LIBSVM and LIBLINEAR (Chang and Lin, 2011; Fan et al., 2008), both written in C++ and wrapped using Cython modules.

5 Related software

Recent years have witnessed a rising interest in machine learning and data mining with applications in many fields. With this rise comes a host of machine learning packages (both open source and proprietary) with which *scikit-learn* competes. Some of those, including Weka (Hall et al., 2009) or Orange (Demšar et al., 2004), offer APIs but actually focus on the use of a graphical user interface (GUI) which allows novices to easily apply machine learning algorithms. By contrast, the target audience of *scikit-learn* is capable of programming, and therefore we focus on developing a usable and consistent API, rather than expend effort into creating a GUI. In addition, while GUIs are useful tools, they sometimes make reproducibility difficult in the case of complex workflows (although those packages usually have developed a GUI for managing complex tasks).

Other existing machine learning packages such as SofiaML[4] (Sculley, 2009) and Vowpal Wabbit[5] are intended to be used as command-line tools (and sometimes do not offer any type of API). While these packages have the advantage that their users are not tied to a particular programming language, the users will find that they still need programming to process input/output, and will do so in a variety of languages. By contrast, *scikit-learn* allows users to implement that entire workflow in a single program, written in a single language, and developed in a single working environment. This also makes it easier for researchers and developers to exchange and collaborate on software, as dependencies and setup are kept to a minimum.

Similar benefits hold in the case of specialized languages for numeric and statistical programming such as MATLAB and R (R Core Team, 2013). In comparison to these, though, Python has the distinct advantage that it is a *general purpose* language, while NumPy and SciPy extend it with functionality similar to that offered by MATLAB and R. Python has strong language and standard library support for such tasks as string/text processing, interprocess communication, networking and many of the other auxiliary tasks that machine learning programs (whether academic or commercial) routinely need to perform. While support for many of these tasks is improving in languages such as MATLAB and R, they still lag behind Python in their general purpose applicability. In many applications of machine learning these tasks, such as data access, data preprocessing and reporting, can be a more significant task than applying the actual learning algorithm.

Within the realm of Python, a package that deserves mention is the Gensim topic modeling toolkit (Řehůřek and Sojka, 2010), which exemplifies a different style of API design geared toward scalable processing of “big data”. Gensim’s method of dealing with large datasets is to use algorithms that have $O(1)$ space complexity and can be updated online. The API is designed around the Python concept of an *iterable* (supported in the language by a restricted form of co-routines called *generators*). While the text vectorizers part of *scikit-learn* also

[4] https://code.google.com/p/sofia-ml
[5] http://hunch.net/∼vw

use iterables to some extent, they still produce entire sparse matrices, intended to be used for batch or mini-batch learning. This is the case even in the stateless, O(1) memory vectorizers that implement the hashing trick of Weinberger et al. (2009). This way of processing, as argued earlier in Section 2.2, reduces various forms of overhead and allows effective use of the vectorized operations provided by NumPy and SciPy. We make no attempt to hide this batch-oriented processing from the user, allowing control over the amount of memory actually dedicated to *scikit-learn* algorithms.

6 Future directions

There are several directions that the *scikit-learn* project aims to focus on in future development. At present, the library does not support some classical machine learning algorithms, including neural networks, ensemble meta-estimators for bagging or subsampling strategies and missing value completion algorithms. However, tasks like structured prediction or reinforcement learning are considered out of scope for the project, since they would require quite different data representations and APIs.

At a lower-level, parallel processing is a potential point of improvement. Some estimators in *scikit-learn* are already able to leverage multicore processors, but only in a coarse-grained fashion. At present, parallel processing is difficult to accomplish in the Python environment; *scikit-learn* targets the main implementation, CPython, which cannot execute Python code on multiple CPUs simultaneously. It follows that any parallel task decomposition must either be done inside Cython modules, or at a level high enough to warrant the overhead of creating multiple OS-level processes, and the ensuing inter-process communication. Parallel grid search is an example of the latter approach which has already been implemented. Recent versions of Cython include support for the OpenMP standard (Dagum and Menon, 1998), which is a viable candidate technology for more fine-grained multicore support in *scikit-learn*.

Finally, a long-term solution for model persistence is missing. Currently, Python's `pickle` module is recommended for serialization, but this only offers a file format, not a way of preserving compatibility between versions. Also, it has security problems because its deserializer may execute arbitrary Python code, so models from untrusted sources cannot be safely "unpickled".

These API issues will be addressed in the future in preparation for the 1.0 release of *scikit-learn*.

7 Conclusion

We have discussed the *scikit-learn* API and the way it maps machine learning concepts and tasks onto objects and operations in the Python programming language. We have shown how a consistent API across the package makes *scikit-learn* very **usable** in practice: experimenting with different learning algorithm is as simple as substituting a new class definition. Through composition interfaces such as Pipelines, Feature Unions, and meta-estimators, these simple building blocks lead to an API which is **powerful**, and can accomplish a wide variety of learning tasks within a small amount of easy-to-read code. Through duck-typing, the consistent API leads to a library that is easily **extensible**, and allows user-defined estimators to be incorporated into the *scikit-learn* workflow without any explicit object inheritance.

While part of the *scikit-learn* API is necessarily Python-specific, core concepts may be applicable to machine learning applications and toolkits written in other (dynamic) programming languages. The power, and extensibility of the *scikit-learn* API is evidenced by the large and growing user-base, its use to solve real problems across a wide array of fields, as well as the appearance of third-party packages that follow the *scikit-learn* conventions. Examples of such packages include *astroML*[6] (Vanderplas et al., 2012), a package providing machine learning tools for astronomers, and *wiseRF*[7], a commercial random forest implementation. The source code of the recently-proposed sparse multiclass algorithm of Blondel et al. (2013), released as part of the *lightning*[8] package, also follows the *scikit-learn* conventions. To maximize ease of use, we encourage more researchers to follow these conventions when releasing their software.

Acknowledgments

The authors and contributors acknowledge active support from INRIA. Past and present sponsors of the project also include Google for funding scholarships through its Summer of Code program, the Python Software Foundation and Tinyclues for funding coding sprints.

Gilles Louppe and Arnaud Joly are research fellows of the Belgian Fonds de la Recherche Scientifique (FNRS) and acknowledge its financial support.

[6] http://astroml.org
[7] http://wise.io
[8] https://github.com/mblondel/lightning

Bibliography

S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith. Cython: the best of both worlds. *Comp. in Sci. & Eng.*, 13(2):31–39, 2011.

J. Bergstra and J. Bengio. Random search for hyper-parameter optimization. *JMLR*, 13:281–305, 2012.

M. Blondel, K. Seki, and K. Uehara. Block coordinate descent algorithms for large-scale sparse multiclass classification. *Machine Learning*, 93(1):31–52, 2013.

C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines. *ACM Trans. on Intelligent Systems and Technology*, 2(3):27, 2011.

L. Dagum and R. Menon. OpenMP: an industry standard API for shared-memory programming. *Computational Sci. & Eng.*, 5(1):46–55, 1998.

J. Demšar, B. Zupan, G. Leban, and T. Curk. Orange: From experimental machine learning to interactive data mining. In *Knowledge Discovery in Databases PKDD 2004*, Lecture Notes in Computer Science. Springer, 2004.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software—Practice and Experience*, 30(11):1203–1233, 2000.

A. Guazzelli, M. Zeller, W.-C. Lin, and G. Williams. Pmml: An open standard for sharing models. *The R Journal*, 1(1):60–65, 2009.

V. Haenel, E. Gouillart, and G. Varoquaux. Python scientific lecture notes, 2013. URL http://scipy-lectures.github.io/.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Comp. in Sci. & Eng.*, pages 90–95, 2007.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

F. Perez and B. E. Granger. IPython: a system for interactive scientific computing. *Comp. in Sci. & Eng.*, 9(3):21–29, 2007.

R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation, Vienna, Austria, 2013. URL http://www.R-project.org.

R. Řehůřek and P. Sojka. Software framework for topic modelling with large corpora. In *Proc. LREC Workshop on New Challenges for NLP Frameworks*, pages 46–50, 2010.

D. Sculley. Large scale learning to rank. In *NIPS Workshop on Advances in Ranking*, pages 1–6, 2009.

P. Seibel. *Coders at Work: Reflections on the Craft of Programming.* Apress, 2009.

J. Vanderplas, A. Connolly, Ž. Ivezić, and A. Gray. Introduction to astroML: Machine learning for astrophysics. In *Conf. on Intelligent Data Understanding (CIDU)*, pages 47–54, 2012.

S. van der Walt, S. C. Colbert, and G. Varoquaux. The NumPy array: a structure for efficient numerical computation. *Comp. in Sci. & Eng.*, 13(2):22–30, 2011.

K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proc. ICML*, 2009.

A declarative query language for statistical inference

Gitte Vanwinckelen, Hendrik Blockeel

Department of Computer Science, KU Leuven, Belgium

1 Introduction

Large volumes of experimental data are generated each day in computational sciences such as bioinformatics, chemistry, and physics. Statistical analyses are no longer only restricted to statisticians, but are increasingly being performed by non-experts [8]. The process is not only labor-intensive but also error-prone. Statistics provides us with a plethora of data analysis and inference methods, making it practically impossible for a scientist to have full knowledge of the existing methods and statistical assumptions that have to be satisfied to apply them. Scientists are often even unaware when a statistical assumption is violated.

We propose to tackle these problems with a formal declarative language in which experimental questions, and the necessary background information, can be formulated. A scientist could then formulate a hypothesis as a query, after which the remainder of the experimental process is performed automatically. This approach allows a scientist to focus on a correct understanding of high level concepts from statistics rather than technical details.

To clarify our goal, we start by comparing this idea to those underlying existing systems. Next, we propose a preliminary design of the language by looking into one specific task, namely, the computation of a confidence interval for the population mean of a random variable.

2 Related work

A great amount of software exists for machine learning and statistics, e.g., Weka and R. These packages require a thorough understanding of statistics to perform meaningful inferences. For instance, they will not give a warning if a user performs a hypothesis test on a sample that is too small to get useful results.

Statistical expert systems aim to remedy these problems by giving advice on the design and analysis of an experiment [5]. While they have the same goal as our declarative experimentation system, the implementation differs. Statistical expert systems are often based on interaction with natural language yes/no questions, and the control system is typically implemented with IF THEN rules. Our solution, on the contrary, allows the user to formulate a query in a formal declarative language together with the necessary constraints and assumptions. A database is used to perform statistical inference and answer the query.

While our system is database oriented, it is different from a statistical database system. Such a system allows the user to perform statistical analyses on a database by computing statistical aggregates, but it disallows access to individual records. Research in this area focuses mostly on data anonymization [2].

Also related are probabilistic databases, which represent uncertainty in the database [4]. It should be noted that our approach is not concerned with uncertainty in the database itself. We assume a given deterministic database that is seen as a sample from a population, and want to query that population, not the database itself. This crucially sets apart the two approaches.

An idea that is more closely related, is the construction of probabilistic models from a relational database by Singh and Graepel [9]; these models then describe the population that the database is a sample from. Methods such as this one will play a role in any implementation of our approach. We here focus, however, on creating a language and execution mechanism that supports a general type of queries, thus offering flexibility and ease of use to the user.

Lastly, our system is related to inductive database systems, which aim to integrate data mining and machine learning into database management systems [6]. They allow for users to query a database for patterns by formulating questions in a declarative language rather than running a predefined algorithm. An inductive database system for constraint based clustering was recently proposed by Adam and Blockeel [1]. While inductive database systems focus on inference with machine learning models, our focus is on statistical inference.

3 Query language design

The language requirements can be looked at from two viewpoints. From the statistical viewpoint, we want to formulate queries in a declarative manner to shield the user from low level choices, e.g., the choice of the method to compute a confidence interval for a population parameter. From the database viewpoint, our query language is based on SQL, but we want to query statistical populations, instead of a finite database. We illustrate the language design by introducing and explaining the 'ESTIMATE' statement.

$$
\begin{aligned}
\langle \mathit{ExpQL} \rangle ::= \ & \text{ESTIMATE } \langle \mathit{population\_statistic} \rangle \\
& \text{FROM } \langle \mathit{sample} \rangle \\
& [\text{ WHERE } \langle \mathit{condition} \rangle \ ] \\
& [\text{ ENSURING } \langle \mathit{statistical\_accuracy} \rangle \ ]
\end{aligned}
$$

$$\langle \mathit{population\_statistic} \rangle ::= \text{PROP } \langle \mathit{data} \rangle \mid \text{MEAN } \langle \mathit{data} \rangle$$

$$\langle \mathit{statistical\_accuracy} \rangle ::= \text{CONF } \langle \mathit{confidence} \rangle$$

The population parameter that we want to estimate is given by $\langle \mathit{population\_statistic} \rangle$. $\langle \mathit{sample} \rangle$ is the data that we have available to estimate the population parameter. It is a database table that has an attribute-value structure. With the WHERE $\langle \mathit{condition} \rangle$ clause we can specify a subpopulation. The ENSURING $\langle \mathit{statistical\_accuracy} \rangle$ clause imposes constraints on the population parameter estimator. For now, we focus on confidence intervals, and the constraints can apply to the confidence level, the width, or a combination of both.

4 Query execution

We illustrate the query execution with an example. Consider a database table *employee* that contains employees of a multinational corporation. Each record consists of the following properties: Id, name, length, gender, nationality, and hair color. We are interested in a 95% confidence interval with a maximum width of 5cm for the expected length of all Swedish male employees with blond hair:

```
ESTIMATE MEAN length
FROM employee
WHERE gender= ‘male’ AND nationality=‘Swedish’ AND haircolor=‘blond’
ENSURING CONF=0.95 AND WIDTH <= 5;
```

If a user specifies both a confidence level and a maximum width, this puts a constraint on the minimum size of the sample. If the sample is too small, we propose two alternative execution strategies to still answer the query.

First, it may be possible to couple the query system to a data generator, and use active learning to generate the necessary data. The data generator may be an actual physical experimentation system. An existing example is the Robot Scientist, which combines physical execution with active learning and automated hypothesis generation [7]. If this is not possible, the system can notify the user about the shortage of data, and request him or her to collect more data. It can still assist the user with the data collection, for instance, by computing the minimum number of samples needed.

Second, we can try to incorporate more data in an intelligent manner by relaxing the constraints. In our example query, the user assumes length is dependent on hair color, gender, and nationality, but perhaps one or more of these dependencies is very weak. For instance, some relationship exists between hair color and length; Scandinavians are on average taller than South Europeans, and are also blond more often. However, it is unclear if this relationship still holds for Swedish men only. If we know hair color and length are independent for Swedish men, we can remove the condition for blond hair and take into account men of any hair color to compute the confidence interval for the mean length.

To apply this approach, we need a method to detect independence between two variables. Many different tests are known for this purpose: The Chi-square test, the Student t-test, etc. Which should be used depends on the types of variables, i.e, categorical or continuous, and their distributions.

In our preliminary experiments on data collected from the UCI repository [3], we investigate constructing a confidence interval for the mean of a continuous variable, which is dependent on a binary variable. A first approach is to test for independence with a t-test that tests for the difference between the means of two independent samples. However, one should be careful with null hypothesis testing. If the null hypothesis is not rejected we have not proven it, instead have insufficient evidence to disprove it. An alternative procedure, which we are still looking into, that does not suffer from this problem is equivalence testing [10].

There is a probability that the test will not detect dependence. This will cause a discrepancy between the imposed confidence level from the query, and the true confidence level. To still provide the user with a correct answer, we aim to quantify this difference.o Our experiments are a first step towards this goal.

The experiments indicate that when a t-test does not detect a difference between the two means for different values of the binary variable, we can safely include the extra data to compute the confidence interval. Because of the additional data, the interval width decreases and this helps us provide a confidence interval with both the required confidence level and maximum width. When the t-test does detect a difference between the means, however, the confidence level is significantly smaller than the requested level, so the approach is not applicable.

5 Conclusion

We presented preliminary ideas on the design of an experimentation system that consists of a declarative language and an inference engine. The language would allow to formulate a hypothesis about a data population, whereafter the inference engine automatically provides an answer, based on a limited sample,. In a first stage of this research we introduced the ESTIMATE statement that can be used to formulate a query about a population statistic. We focused on the computation of a confidence interval for the population mean. The system is responsible for choosing the appropriate execution strategy to satisfy the requested confidence level and width. To this purpose, we proposed two different approaches; supplementing the database with new data, or attempting to relax some of the constraints that the user imposed on the data.

We plan to extend this work with additional queries in order to arrive at a complete the language. Our focus lies on queries relevant to machine learning researchers. Furthermore, while the preliminary language design is based on SQL, we also plan to investigate probabilistic logic programming languages (e.g., Problog), and first order knowledge representation languages (e.g., FO(.)).

References

1. A. Adam and H. Blockeel. A query language for constraint-based clustering. *Benelearn*, pages 1–7, 2013.
2. N.R. Adam and J.C. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.
3. K. Bache and M. Lichman. UCI repository. http://archive.ics.uci.edu/ml, 2013.
4. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, July 2009.
5. D.J. Hand. Expert systems in statistics. *The Knowledge Engineering Review*, 1:2–10, 1984.
6. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.
7. R. King, M. Young, A. Clare, K. Whelan, and J.J. Rowland. The robot scientist project. In *Discovery Science*, pages 16–25, 2005.
8. J. Leek. The vast majority of statistical analysis is not performed by statisticians. http://simplystatistics.org/2013/06/14/the-vast-majority-of-statistical-analysis-is-not-performed-by-statisticians/, 2013. Accessed: 2013-06-24.
9. S. Singh and T. Graepel. Compiling relational database schemata into probabilistic graphical models. In *Neural Information Processing Systems (NIPS), Workshop on Probabilistic Programming*, 2012.
10. D. L. Streiner. Unicorns do exist: a tutorial on "proving" the null hypothesis. In *A guide to the statistically perplexed:selected readings for clinical researchers*, pages 211–223. University of Toronto Press, 2013.

Author Index