Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Magistrale

Hypergraph Mining for Social Networks

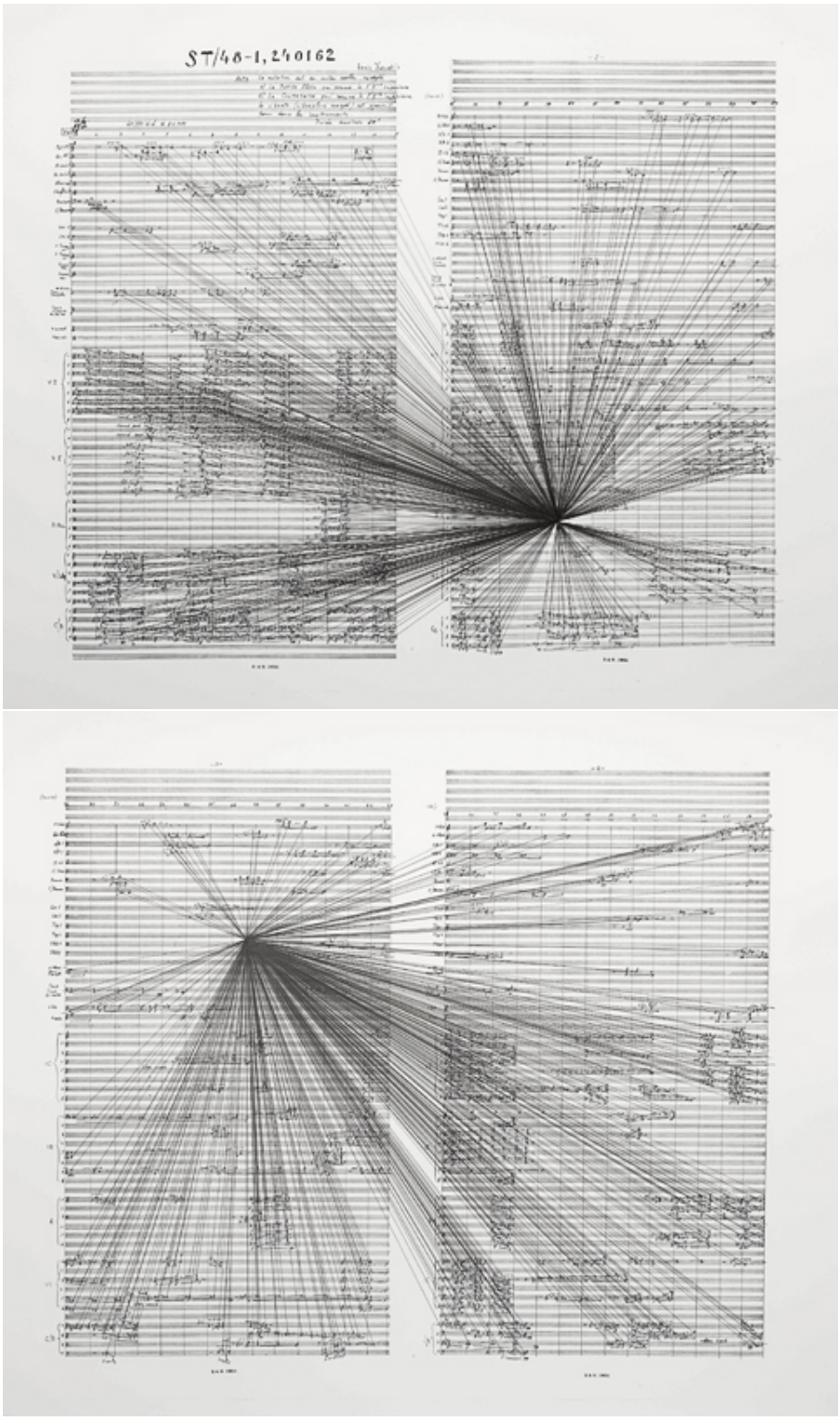
Tesi di Laurea in Complementi di Basi di Dati

Relatore:
Chiar.mo Prof.
Danilo Montesi

Presentata da:
Giacomo Bergami

Sessione I
Anno Accademico 2013/2014

CONTENTS



Iannis Xenakis: "Mass Black Implosion". Some examples of Graph musical notation on contemporary classical music. More on http://marcofusinato.com/projects/mass_black_implosion/01.php

PREFAZIONE

La tesi si prefigge lo scopo di mostrare come tutt'ora manchi un'analisi sistematica su come strutturare i dati a grafo, allo scopo di effettuare query e mining : di fatti già da tempo la ricerca accademica ha messo in luce come strutture dati per relazioni n-arie richiedano l'utilizzo di ipergrafi per poter mappare nel modo più espressivo possibile i dati, ma tutt'ora ci si limita ad una rappresentazione a grafo, che non sempre può essere comoda.

Dopo una dissertazione iniziale, si è dimostrato come questi ipergrafi possano in realtà essere rappresentati tramite delle strutture dati già note, ovvero dei database relazionali e delle matrici di adiacenza di grafi. Questo permette di non "reinventare la ruota", ma di utilizzare (almeno per un primo momento) delle strutture dati già note e già ottimizzate allo scopo, quali appunto i database relazionali. Un altro vantaggio dato da questa rappresentazione è la possibilità in alcuni contesti di effettuare una forte parallelizzazione delle computazioni: questa prospettiva, sebbene sia molto allettante, va oltre gli scopi iniziali di questa tesi, e pertanto viene analizzata unicamente in appendice.

La tecnica di mining su grafi è inoltre molto giovane, ed è nostra opinione che alcune tecniche che si sono rivelate vincenti nel campo pratico non siano state ancora prese in considerazione dal punto di vista teorico, e non siano molto citate in letteratura. Questa situazione fa porre molti interrogativi al lettore, che deve pertanto comprendere come questo campo di ricerca sia molto giovane (il primo libro che parlava espressamente del data mining su Grafi è stato edito solo in Luglio 2013); da ciò segue che inevitabilmente le tecniche qui riportate siano sì parte dello "stato dell'arte", ma non sono da ritenersi necessariamente le uniche e sole efficaci fin'ora sviluppate.

Altra tecnica già descritta in letteratura ma non ancora presa in seria considerazione è la definizione di un'algebra per il data mining: si è quindi deciso di trasporre quest'idea nell'ambito del *graph mining* allo scopo di generalizzare il meccanismo di alcuni algoritmi, indipendentemente dalla loro effettiva implementazione, e di rendere possibile in un futuro l'estensione della stessa con nuovi operatori sui grafi, che possono essere fatti derivare da studi futuri.

Lo scopo di questa tesi è quindi quello di presentare una *summa* di tutte le possibili tecniche che possono essere utilizzate per "interrogare" i dati strutturati ad (iper)grafo, per poi implementare su di essi la nostra algebra, di cui ci si è occupati nei primi capitoli.

In ultima analisi questa nuova definizione potrebbe sembrare solo un puro artificio logico se non si delineasse anche un contesto di applicazione pratico per le definizioni e teoremi forniti: allo scopo ci si è focalizzati sullo studio dei social network, che sono un altro ambito recente di ricerca e su cui molti interrogativi.

1

PREFACE

RICERCAR, The Theme Given by the King's Command Resolved According to the Canonic Style:
The initial Latin inscription of the "Musikalisches Opfer" by Johann Sebastian Bach

Nowadays, more and more data is collected in large amounts, such that the need of studying it both efficiently and profitably is arising; we want to acheive new and significant informations that weren't known before the analysis. At this time many graph mining algorithms have been developed, but an algebra that could systematically define how to generalize such operations is missing. In order to propel the development of a such automatic analysis of an algebra, We propose for the first time (to the best of my knowledge) some primitive operators that may be the prelude to the systematical definition of a hypergraph algebra in this regard.

1.1 WHY TO DEFINE AN ALGEBRA OVER HYPERGRAPHS?

At this point, I will use a philosophical methodology, called *maieutics*, whith which we like to prevent some counterarguments that are frequent and usual when a new methodology is introduced.

Why should we use hypergraphs instead of Graphs? I studied the hypergraph structure as a model for mapping datas because:

- [Fag83] firstly showed how databases could be expressed as hypergraphs: this encourages the research to go towards this initial intuition.
- The path model in stratification, that [Fre05] expressed in a graph, could be better expressed with an hyperedge, where all the preconditions are expressed in the tail of the hyperedge and the consequences in its head. Its weight could represent the accuracy mined correlation.
- [Gal+93] showed that hyperedges could better represent functional dependency in databases, since in this way we could easily detect which part of the data establishes such relation.

- [ERV05] points out how many complex networks could even be expressed in hypergrpahs, since a lot of financial relations involve more than two entities; some other relations, like friendship and collaboration, could be instead easily represented as binary transactions.

Moreover, I'll show how recent graph-mining techniques could be used even over hypergraphs, given that the algebra operators that I'll define will create an abstraction layer over the datas.

Why analyze HyperGraph data? [Ber+11] points out that nowadays a lot of data, including those of social networks ([MR13b; MR13a]) and Genetic Analysis[1], is represented in a graph form. In real world data also appears in a multidimensional way, as it maps on the same layer different kind of relations and correlates each layer with other interconnections, where each level represents a different point of view in the reality's representation. Those *explicit connections* could be also "mined" through the imposition of rules that must comply with certain clauses, in order to find *implicit correlations* inside the data. In order to define a structure where both data and relations are mixed together, I choose to use hypergraph in order to give a better representation of the concepts of shared data between the entities, correlation among the datas and the mined properties.

Why do we need an algebra for Hypergraph Data Mining? Mainly, as highlighted as well by [Ber+11], «*the literature still misses a systematic definition of a model for multidimensional networks*» wich, throug data mining, could point out those implicit relations over data that we could obtain by mapping those relation through Hypergraph Data Mining and by collecting the information gained from data mining operations over the corrispondent graph. By the way, [Ber+11] points out a way to categorize the data through an analysis of its size, but doesn't show how to obtain implicit relations through data mining. As my work defines a model that maps all the tabular operations into the hypergraph data world, this model could inherit also any measuring system over multilayered graphs. We could want also to design my model data in order to be aware of the temporal evolution of the network ([MR13a; Ber+11]).

1.2 INTRODUCTION TO DATA MINING

1.2.1 *Relational Data*

Concerning the data mining techniques over structured data, I wish to draw attention to classification, association and cluster analysis [TSK05]. **Classification** has the aim to categorize the data over some rules generally defined as a *classification model*, that could be obtained over some data used to train the model.

The **Clustering** algorithms divide data into groups, according to its spatial arrangement and their content, in order to detect some similarities inside the data.

1 http://www.geneontology.org.

- The clustering algorithm is said to be **hierarchical** if it divides the data in nested clusters that provide a set hierarchy over the data. If we stop the analysis at the first level of the hierarchy, then we have a **partitional clustering**.
- The clustering algorithm is said to be **exclusive** if each object is assigned to a single cluster; if otherwise an object could be assigned even to more than one cluster, then we have an **overlapping clustering**. It exists also a **fuzzy clustering**, where to each object a *membership weight* is assigned to each inferred cluster.
- A **complete clustering** assigns each object to a specific cluster, whereas in the **partial clustering** some nodes (called *outliers*) could be not included in any cluster, and hence is generally used for noise detection.

Definition 1.2.1. *Given a power set function $\mathcal{P}(S) = \{T | T \subseteq S \wedge T \neq \emptyset\}$, the set of all the possible clusterizations is given by $\mathcal{P}(\mathcal{P}(S))$, and hence a possible clusterization for S is a $C \in \mathcal{P}(\mathcal{P}(S))$. [TSK05] If $\cup_{c \in C} c = S$ then C is a **complete clustering**, otherwise ($\cup_{c \in C} c \subset S$) is a **partial clustering**. If $\forall c, d \in C . c \neq d \Rightarrow c \cap d = \emptyset$ then C is a **exclusive clustering**, otherwise ($\forall c, d \in C . c \neq d \Rightarrow c \cap d \neq \emptyset$) is a **overlapping clustering**. Hence, any clustering algorithm $\mathcal{A}$ could be generalized by a function $\mathbb{C}^{\mathcal{A}}$ where $\mathbb{C}^{\mathcal{A}} : S \to \mathcal{P}(\mathcal{P}(S))$*

An **Association** algorithm has the aim to discover some regularities over a sequence of items: given that such associations are usually performed over transactions above different measures of interestingness, their graph mining counterparts have the aim to perform such operations via (e.g.) subgraph mining and vertex clustering techniques.

[TSK05] points out that we should avoid model underfitting and overfitting, respectively due to a lack of representative candidates in the training data or to having collected so much data that is too adhering to the data model.

1.2.2 *Graph Mining: Problems*

The aim of graph mining is not only to retrive some brand new information over data, but also to remove noise that is always present inside large data. For this purpose, we want to adopt some statistical techniques that could cleanse our data and mine some interesting properties. Given that graph mining algorithms are still very computationally costly, we will ignore the time complexity factor, and we'll focus on the property of the depicted operations. In particular, real data could be collected in a graph form, which has the aim to ease the representation of the data in a way that remarks the interconnections of the single elements; moreover we could think to solve some interesting problems that deserve special attention.

Problem 1.1. *[Kim+13; Zac77] Given a social network where all the vertex represent the users and where the edges' weight represente the strength of the bond, we would like to detect wich are the most representitive communities.*

In [Zac77] we have two distinct factions ruled by two "enemies", where the edge's network form a directed graph which edges could be eventually weighted in order to remark the strenght

of the friendship. In [Kim+13] we have a on-line social network, where it is stated that the user-user linking models the mouth-to-mouth spread: their aim is to solve the Influence Maximization Problem by finding those people inside the net that are more influent, and hence contribute to the dissemination of information.

Problem 1.2. *[BH09; For10] Given an PPI (Protein Protein Interaction) network where the vertices represent the proteins and the unweighted edges represent the interaction between the substances, we want to identify those protein that that belong to the same complex. In this case we want to detect the proteins' functional groups ("proteins having the same or similar functions, which are expected to be involved in the same processes").*

Since these problems involve only one graph per time, we could use *intra-graph clustering* in order to detect the interested areas wich form a subgraph.

Problem 1.3. *[Des+05] We want to build a classifcation model for chemical compounds based on their common substructure: this could be particularly useful since* ***topological descriptors*** *(based on frequent subgraph search) and* ***geometric descriptors*** *(which are also based on the disposition of the graph nodes in a n-dimensioanl space, where usually $n = 3$) are superior to physicochemical properties. In order to acheive my goal, I could use the frequent subgraphs in a given graph database, and then use those to group via clustering the graphs by most frequent shared graphs.*

Another problem strictly related to the subgraph mining is the formation of the substructure interaction graphs, which has the aim to find new correlations inside the data:

Problem 1.4. *[TTM11] Given a bipartite graph $G = (\mathcal{G} \cup \mathcal{T}, E)$ where $E = \mathcal{G} \times \mathcal{T}$ and where $\mathcal{G} = \{ \mathcal{G}_i \}_{i \leqslant k}$ is the set of the chemical compounds and $\mathcal{T} = \{ \mathcal{T}_i \}_{i \leqslant l}$ is the set of the target proteins, we want to mine which substructure g of G_i interacts with the substructure t in T_j ($g \leftrightarrow t$).*

1.3 R and statistical analysis

R [Mat11] is a tool that has born for statistical analysis, and that nowadays[2] is used for data mining [Zha13; Van09]. We would like to check if such language could be used as a future environment for developing our hypergraph mining algebra.

1.4 Notation

- The **filter** function over over a set L returns the subset $l \subseteq L$ where each element of l statisfies the property P:

$$Filter(L, P) = \{ e \mid e \in L \wedge P(e) \}$$

2 http://www.revolutionanalytics.com/.

- The **map** function transforms each element in L with a function f:

$$Map(L, f) = \{\, f(e) \mid e \in L \,\}$$

- The **work environment** is mainly the place where all the functions, data and its operators are saved.
- V is the set of all the values that are represented inside the data.
- D is the set of all the values that are used for expressing a metric or a weight of some data properties. These values could be also be choosen in a $[0, 1]$ range ($D \equiv [0, 1]$).
- $t : T$ means that "t has **type** T", and $T \in \mathcal{T}$ where $\mathcal{T}$ is the set of all the (data) types.
- $T_1 <: T_2$ means that "T_1 is a **subtype** of T_2"
- Given a matrix M, with $M_{i,j}$ (or M_{ij} or $M[i, j]$) we want to select the j-th column of the i-th row.
- Given a matrix M, $rows(M)$ returns the list of all the **rows** in M and $nrows(M) = |rows(M)|$.
- Given a matrix M, $cols(M)$ returns the list of all the **columns** in M and $ncols(M) = |cols(M)|$.
- I assume that each data element o that is stored in the environment has an **index** $i = \varphi(o)$, where $\varphi\colon Obj \to Idx$; since we assume that to each index it corresponds one and only one object and vice versa, it exists also the inverse function $\varphi^{-1}(i) = o$. Hereby we could easily assume that each object colud be casted as its index when needed.
- $\mu_x.S \triangleq \min_x S$.
- Given a table t, $t.X$ or $t[X]$ means "retreive from table X the element of type X. Moreover $\pi_{\vec{L}}(t) \equiv t[\vec{L}]$.

1.4.1 *Dovetailing*

Since $|\mathbb{N}| = \aleph_0 = |\mathbb{N}^2|$, then we could code the integer couples $\mathbb{N}^2$ to $\mathbb{N}$ by **dovetailing**.

Definition 1.4.1 (Dovetailing function)**.** *Given two numbers $i, j \in \mathbb{N}$, it is possible to express them as a single integer via the **dovetailing** function [Odi92] as follows:*

$$dt(i, j) = j + \sum_{k=0}^{i+j} k$$

It is also possible to reverse this process, that is to obtain the i and j that previously formed the c value, and hence we could define a $dt^{-1}(c) = \langle i, j \rangle$ **reverse dovetailing function**.

Lemma 1.1 (Reverse Dovetailing function). *Given that a "diagonal" d is a set of cells (i, j) where $i + j = d$ and S_k a shorthand of $\sum_{i=0}^{i} = \frac{k(k+1)}{2}$, the **reverse dovetailing function** is defined as follows:*

$$dt^{-1}(c) = \langle d_{max} - (c - S_{d_{max}}), c - S_{d_{max}} \rangle$$

Proof. Suppose that now we want to define $dt^{-1}(c)$: given that c is inside a row d and then $S_d \leqslant c < S_{d+1}$, when we have that

$$\begin{cases} d^2 + d - 2c \leqslant 0 \\ d^2 + 3d + 2 - 2c \geqslant 0 \end{cases}$$

If $c > 0$ ($c = 0$ is trivial and $dt^{-1}(0) = \langle 0, 0 \rangle$) then we obtain that $d_{min} = \frac{-3+\sqrt{1+8c}}{2}$, $d_{max} = \frac{-1+\sqrt{1+8c}}{2}$, and $d_{min} < c \leqslant d_{max}$, hence $d \in [\lceil d_{min} \rceil, \lfloor d_{max} \rfloor]$. Since $d_{max} - d_{min} = 2$, when $\lceil d_{min} \rceil \neq d_{min}$ then $\lfloor d_{max} \rfloor - \lceil d_{min} \rceil = 1$ and then $d = \lceil d_{min} \rceil = \lfloor d_{max} \rfloor$ is the only possible solution; otherwise (when $-3 + \sqrt{1 + 8c}$ is not a perfect square) then $\lfloor d_{max} \rfloor$ is the expected solution for d. In order to reconstruct the original coordinates, we could see that the second coordinate is obtained with $j = c - S_{d_{max}}$, while the first one is $i = d_{max} - (c - S_{d_{max}})$ by the diagonal definition. Finally:

$$dt^{-1}(c) = \langle d_{max} - (c - S_{d_{max}}), c - S_{d_{max}} \rangle$$

□

Here we could express any tuple of items $\langle a_1, \ldots, a_n \rangle$ as $\langle a_1, \langle a_2, \ldots \langle a_{n-1}, a_n \rangle \ldots \rangle \rangle$; in order to keep track of how many times to unfold the couples in order to terminate the visit of the nested tuple, we could express the latter representation as $\langle n, \langle a_1, \ldots \langle a_{n-1}, a_n \rangle \ldots \rangle \rangle$. Given this representation, we could think to represent these tuples in a single number via dovetailing.

Definition 1.4.2 (Vectorial Dovetailing). *Given a vector $\vec{a} \in \mathbb{N}^n$, we could define its dt representation called **vectorial dovetailing** as follows:*

```
let dt⃗ l =
  let rec R ls =
    match ls with
    | [] -> 0
    | a::[] -> a
    | a::l' -> dt a (R l')
  in (dt |l| (R l))
```

Algorithm 1.1: Vectorial Dovetailing

The inverse function could be given as follows:

Definition 1.4.3 (Reverse Vectiorial Dovetailing). *Given a numerical representation of a tuple c, we could define the following **reverse vectorial dovetailing** function:*

```
let \overrightarrow{dt^{-1}} c =
  let rec Rdt n cl =
    match (dt^{-1} cl) with
    ⟨i,j⟩ ->
      if (n == 0) then []
      else if (n == 1) then [cl]
      else i::(Rdt (n-1) j)
  in match (dt^{-1} c) with
    ⟨i,j⟩ -> Rdt i j
```

Algorithm 1.2: Vectorial Dovetailing

Algorithm C.2 on page 142 and C.4 on page 146 provide an implementation of such dovetailing function in both R and Java.

Part I

DATA MODELLING

2

DATA MODEL

Contents

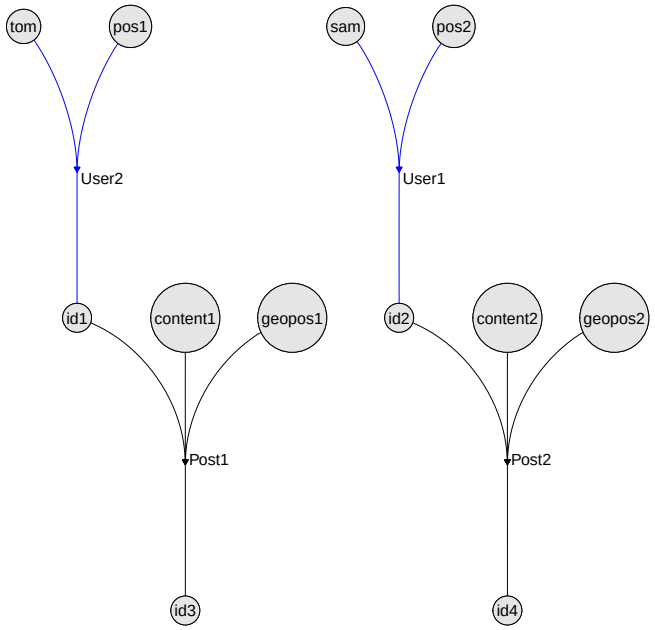
Our aim is to give a progressive introduction to our data model, in order to gradually explain which are our choices for providing a more powerful expressiveness inside our data. In this chapter we'll show the base data model, which is the hypergraph, that could be used to represent both multidimensional relations and correlations among the datasets.

2.1 BASIC HYPERGRAPH

Definition 2.1.1 (Hypergraph). *An **basic hypergraph** is defined as a couple $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of the vertices ($\mathcal{V} \subseteq V$), and $\mathcal{E} \in \mathcal{P}(\mathcal{P}(\mathcal{V}))$. If an hypergraph is directed[Gal+93], then each edge $e \in E$ is equipped by a head $H(e)$, a tail $T(e)$ and the union of the both is called $U(e) = H(e) \cup T(e)$; the notation $e \equiv a :: b$ could be used for defining an hyperedge with $H(e) = a$, $T(e) = b$.*

Example

1. *Figure 2.1 shows an example of an hypergraph, where the data-vertices are made explicit. Due to the `hyperdraw` implementation, we can't define same labels for different hyperedges, and hence*



(a) *Hypergraph*

User	ULocation	Uid
tom	pos1	id1
sam	pos2	id2

(b) *Tabular representation of the User relation*

Uid	Text	PLocation	Pid
id1	content1	getpos1	id3
id2	content2	getpos2	id4

(c) *Tabular representation of the Post relation*

Figure 2.1: An example of an Hypergraph; the code used for plotting this structure was given in Algorithm C.1 on page 141.

the i-th instance of a relation of a given type T is labelled as $T1$ for $i = 1$. In the given hypergraph different types of relations are hereby marked with different colours.

This hypergraph depicts data-driven relations that are marked by shared vertices: in this case it is quite simple to correlate each social-network user to his post. The hypergraph given in the picture could be formally defined as follows:

$$\mathcal{H} = (\mathcal{V}, \mathcal{E})$$

$$\mathcal{V} = \{\, tom, sam, pos1, pos2, id1, id2, id3, id4, content1, content2, geopos2, geopos1 \,\}$$

$$\begin{aligned}\mathcal{E} =& \{\{id1\} :: \{tom, pos1\}, \{id2\} :: \{sam, pos2\}, \{id3\} :: \{id1, content1, geopos1\},\\ & \{id4\} :: \{id2, content2, geopos2\}\}\end{aligned}$$

In this case it is clear that there is no need to make explicit the relation "post p belongs to user u", since this correlation is data driven: the post this correlation is obvious by analysing the shared data among the hyperedges.

Moreover we could see that vertices that don't belong to any hyperedge are meaningless since they won't be part of any entity or relation. At this point we could assume to ignore by design all the vertices from the *hypergraph* definition and to define the set of all the vertices as:

$$\mathcal{V} \triangleq \bigcup \{\, U(e) \mid E \in \mathcal{E}_D \cup \mathcal{E}_E \,\}$$

We could assume to not explicitly define $\mathcal{V}$ in all our next declarations, and hence we'll use $\mathcal{V}$ only as a useful shorthand.

Definition 2.1.2 (Data-hyperedge)**.** *Given the V set of all the possible values of the dataset, an **data-hyperedge** is defined as a directed hypergraph's edge $hd :: tl$, where $hd, tl \in \mathcal{P}(V)$, where hd is the head of the hyperedge and tl is its tail.*

In this review, we use a peculiar specialisation of the hyperedge definition, where we distinguish the data relations ($\mathcal{E}_D$) from the correlational ones ($\mathcal{E}_E$).

Definition 2.1.3 (Hypergraph Data Model)**.** *The **Hypergraph Data Model** (HDM) is defined as a couple $\mathcal{H} = (\mathcal{E}_D, \mathcal{E}_E)$ where the data relations ($\mathcal{E}_D$) and the correlational relations ($\mathcal{E}_E$) are set of hyperedges and disjoint sets ($\mathcal{E}_D \cap \mathcal{E}_E = \varnothing$). Each hyperedge $e \in \mathcal{E}_D \cup \mathcal{E}_E$ is defined as the following quadruple, where the first element is the schema of the relation driven by the data itself ($\mathbb{S}(e) \in \mathcal{P}(\mathcal{T})$), the second is the "label" or "type" of the relation ($\mathbb{T}(e) \in Label$), the third is the data-hyperedge ($\mathbb{D}(e) \in \mathcal{P}(\mathcal{V})$) and the fourth casts some more additional informaton, like a metric over the data hyperedge ($w(e) \in D$):*

$$e \equiv \langle \mathbb{S}(e), \mathbb{T}(e), \mathbb{D}(e), w(e) \rangle$$

Moreover the following axiom must be true:

$$HG1 \quad \forall e, e' \in \mathcal{E}_D.\mathbb{T}(e) = \mathbb{T}(e') \Rightarrow \mathbb{S}(e) = \mathbb{S}(e')$$

that induces a bijective function $sft\colon Label \to \mathcal{T}$*, where* sft *stands for "scheme·from·type". The correlational relations have a type but don't necessarily meet the property (HG1), since we would like to correlate with the same relation many different kinds of data. Moreover we define the following shorthands:*

$$U(e) \triangleq U(\mathbb{D}(e)) \quad H(e) \triangleq H(\mathbb{D}(e)) \quad T(e) \triangleq T(\mathbb{D}(e))$$

Example

2. *Let's see Figure 2.2: we could see that the 'Post' and 'User' hyperedges are data-driven, and that they explicit a correlation between the single atomical data parts in order to define the entities of the hypergraph. Moreover we'll have that* `Post<:`$\mathcal{E}_D$ *and* `User<:`$\mathcal{E}_D$*.*

Let's take a look at the 'Follow' relationships in red: even if these edges are data-driven, they don't define any interesting entity of the database, but they establish a correlation between two (or more) elements, and hence `Follow<:Correlation<:`$\mathcal{E}_E$*.*

The current hypergraph could be represented as $\mathcal{H} = (\mathcal{E}_D, \mathcal{E})$*, where* $\mathcal{V}$ *is here only showed in order to explicitly state the types of the data:*

$$\begin{aligned}\mathcal{V} =& \{tom : User, sam : User, pos1 : ULocation, pos2 : ULocation, id1 : Uid,\\ & id2 : Uid, id3 : Pid, id4 : Pid, id5 : Pid, content3 : Text,\\ & content1 : Text, content2 : Text, geopos2 : PLocation,\\ & geopos1 : PLocation\}\end{aligned}$$

$$\begin{aligned}\mathcal{E}_D =& \{\langle \{User, ULocation, Uid\}, \text{“}User\text{”}, \{id1\} :: \{tom, pos1\}, 1\rangle,\\ & \langle \{User, ULocation, Uid\}, \text{“}User\text{”}, \{id2\} :: \{sam, pos2\}, 1\rangle,\\ & \langle \{Uid, Text, PLocation, Pid\}, \text{“}Post\text{”}, \{id3\} :: \{id1, content1, geopos1\}, 1\rangle,\\ & \langle \{Uid, Text, PLocation, Pid\}, \text{“}Post\text{”}, \{id4\} :: \{id1, content2, geopos2\}, 1\rangle,\\ & \langle \{Uid, Text, PLocation, Pid\}, \text{“}Post\text{”}, \{id5\} :: \{id1, content3, geopos2\}, 1\rangle\\ & \}\end{aligned}$$

$$\begin{aligned}\mathcal{E}_E =& \{\langle \{UId, UId\}, \text{“}Follows\text{”}, \{id1\} :: \{id2\}, 1\rangle,\\ & \langle \{UId, UId\}, \text{“}Follows\text{”}, \{id2\} :: \{id1\}, 1\rangle\}\end{aligned}$$

These examples could guide to the following definition of the mappings between databases, graphs and hypergraphs.

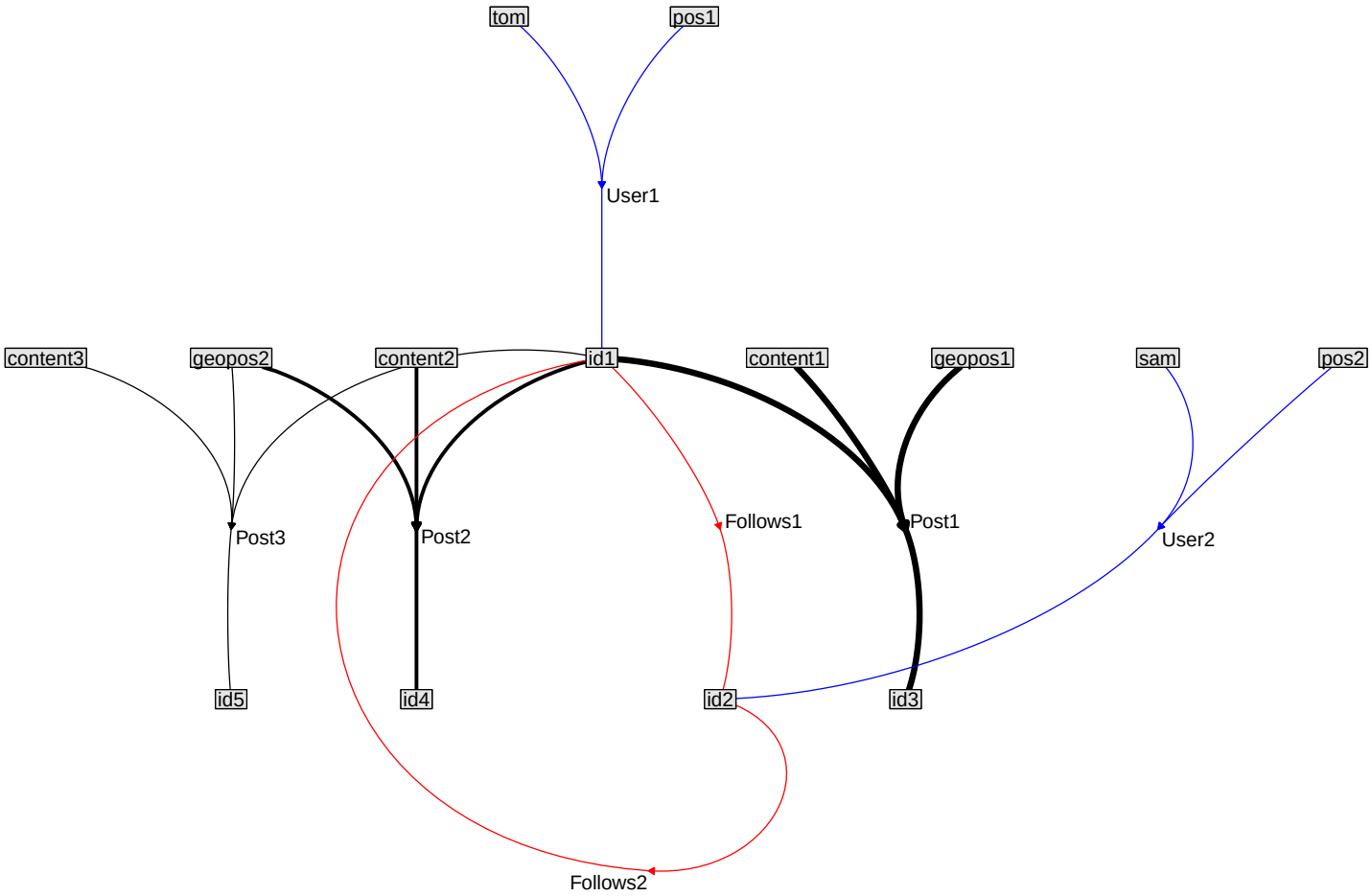


Figure 2.2: *An example of an Hypergraph Data Model, where some different types of Hyperedges are showed. Edges with different width are used when it's not visually clear where each arm of the hyperedge is directed.*

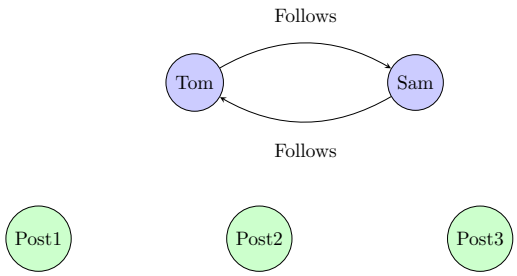


Figure 2.3: *Hypergraph Data Model: the same data in Figure 2.2 represented as a graph. Note that in the previous data model the correlation between users and their post is more natural and data-driven, and no explicit hyperedges are necessary to make it evident.*

Definition 2.1.4 (Hypergraph adjacency matrix)**.** *Given an hypergraph data model* $\mathcal{H} = (\mathcal{E}_D, \mathcal{E}_E)$*, its* ***adjacency matrix*** $A'(\mathcal{H})\colon V \to \mathcal{E}_D \cup \mathcal{E}_E \to \mathbb{D}$ *is defined as:*

$$A'_{ij}(\mathcal{H}) = \begin{cases} 1 & v_i \in U(e_j) \\ 0 & oth. \end{cases}$$

If we want to make explicit that we could correlate the hyperedges-tuples over some attributes, we could define the ***typed adjacency matrix*** $A(\mathcal{H})\colon \mathcal{E}_D \cup \mathcal{E}_E \to \mathcal{T} \to \mathbb{D}$ *where wemake explicit the values of the data as follows:*

$$A_{jt}(\mathcal{H}) = \begin{cases} v_i & \exists v_i \in U(e_j).v_i : t \\ \varepsilon & oth. \end{cases}$$

2.1.1 *Databases vs. Hypergraphs*

Definition 2.1.5 (COE framework)**.** *A* ***COE framework*** *[MMo6] is defined by a three level of datasets, where a collection* $c \in C$ *is a set of objects* ($c \in \mathcal{P}(O)$)*, an object* $o \in E$ *is a set of entities* ($o \in \mathcal{P}(E)$)*, and an entity* $e \in E$ *is a collection of data* $e \in \mathcal{P}(V)$*.*

Definition 2.1.6 (Database)**.** *A* ***database*** *is defined as a collection of tables* ($DB \in C$)*, where each table* $t \in DB$ *is defined as an object* ($t \in O$) *formed by entries* $e \in t$ *that are entities* ($e \in E$) *defined as* $e = \langle \mathbb{S}(e), \mathbb{T}(e), \mathbb{D}(e) \rangle$ *with type* $\mathbb{T}(e)$ *and schema* $\mathbb{S}(e)$*. Moreover the following axioms must be true:*

DB1 $\forall t, t' \in DB.t = t' \Leftrightarrow \mathbb{T}(t) = \mathbb{T}(t')$

DB2 $\forall t \in DB.\forall e, e' \in t.\mathbb{T}(e) = \mathbb{T}(e') \wedge \mathbb{S}(e) = \mathbb{S}(e') \wedge \mathbb{T}(e) = \mathbb{T}(t)$

The notation $e = \langle \mathbb{S}(e), \mathbb{T}(e), \mathbb{D}(e) \rangle$ with type $\mathbb{T}(e)$ given for table records make necessary to redefine all the basic operations over data, since in this case we must also consider that the record contains the information of the tuple's schema and datatype.

Example

3. *Let's provide a short example: the union operation between two data tables* t *and* t' *with the same schema could be represented as follows:*

$$t \cup t' = \left\{ \left\langle \mathbb{S}(e), \mathbb{T}(e) \cdot '' \vee '' \cdot \mathbb{T}(e'), e \right\rangle \;\middle|\; e \in t \vee e \in t' \right\}$$

where:

$$e \in t \Leftrightarrow \exists d.\mathbb{D}(e) = \mathbb{D}(d) \wedge d \in t$$

Definition 2.1.7 (Database→Hypergraph Data Model morphism). *The $DB \to \mathcal{H}$ transformation is given by:*

$$DB\mathcal{H}(db) = (\{\, \langle \mathbb{S}(e), \mathbb{T}(e), ID(e) :: e \backslash \{ID(e)\}, 1 \rangle \mid t \in db \wedge e \in t \wedge e \,\}, \varnothing)$$

Moreover banally (HG1) is derived by (DB2) via the definition of $\mathcal{E}_D$. We don't map the database's tuples into $\mathcal{E}_E$ since standard databases don't explicitly cast informations of data correlations, and hence $\mathcal{E}_E = \varnothing$.

Definition 2.1.8 (Hypergraph Data Model→Database morphism). *Given the set of all the possible types in the Hypergraph Data Model h and a set $SA(h) = \{\, \mathbb{T}(a) \mid a \in \mathcal{E}_D \vee a \in \mathcal{E}_E \,\}$, the $\mathcal{H} \to DB$ transformation is given by:*

$$\mathcal{H}DB(h) = \{\, \{\, \langle \mathbb{S}(e), \mathbb{T}(e), U(e) \rangle \mid e \in \mathcal{E}_D \cup \mathcal{E}_E \wedge \mathbb{T}(e) = t \,\} \mid t \in SA(h) \,\}$$

Given that $\mathcal{H}DB$ collects by definition in a same table all the vertices that have a same type t, we have only one table with that type, and hence condition (DB1) and the first and third part of (DB2) is satisfied. (HG1) and the first part of (DB2) gives the proof for the second part of (DB2).

2.1.2 *Graph vs. Hypergraph*

Definition 2.1.9 (Graph). *A **graph** is a $G = (V, E)$ where $E \subseteq V \times V$, where $\lambda \colon V \to D$ maps each vertex (i.e.) to its probability value, and $w \colon E \to D$ maps each edge to its weight.*

[DVMT13] shows how relational databases could be mapped in a graph data model even if, by doing this, the final graph nodes doesn't respect the initial database structure, as each node doesn't exactly match with a given entry of a database's table. In order to distinguish if the data is shared among the entities or if there is a generic correlation that is not made explicit by the data itself, it could be useful to treat graphs where each vertex represents a single entity, and where edges represent relations between those entities. [Vazo9] points out how «*[hypergraphs are] very useful for representing a population of elements and their attributes*». Furthermore we could map this model to hypergraphs as follows:

Definition 2.1.10 (Graph→Hypergraph Data Model morphism). *Given a graph $g = (V, E)$ we could define a morphism to hypergraphs as follows:*

$$G\mathcal{H}(g) = (\mathcal{E}_D, \mathcal{E}_E)$$

$$\mathcal{E}_D = \{\, \langle \mathbb{S}(v), \mathbb{T}(v), (ID(v) :: v \backslash ID(v)), \lambda(v) \rangle \mid v \in V \,\}$$

$$\mathcal{E}_E = \{\, \langle \mathbb{T}(v), (v \backslash ID(v)) :: (u \backslash ID(u)), w(u, v) \rangle \mid (u, v) \in E \,\}$$

By the way the hypergraph is more expressive than the graph. In order to show that, we shall introduce the following definitions:

Definition 2.1.11 (Prev). *The **prev** function over a hyperedge $f \in \mathcal{E}_D \cup \mathcal{E}_E$, is defined as the set of hyperedges where each of them are linked to f by the vertices in $T(e)$.*

$$\forall e, f \in \mathcal{E}_D \cup \mathcal{E}_E . e \in prev(f) \Leftrightarrow \exists v \in \mathcal{V} . v \in U(e) \cap T(f) \Leftrightarrow \exists v \in T(f) . v \in U(e)$$

Definition 2.1.12 (Next). *The **next** function over a hyperedge $f \in \mathcal{E}_D \cup \mathcal{E}_E$, is defined as the set of hyperedges where each of them are linked to f by the vertices in $H(e)$.*

$$\forall e, f \in \mathcal{E}_D \cup \mathcal{E}_E . e \in next(f) \Leftrightarrow \exists v \in \mathcal{V} . v \in U(e) \cap H(f) \Leftrightarrow \exists v \in H(f) . v \in U(e)$$

Definition 2.1.13 (Adj). *The **adj** function over a hyperedge $f \in \mathcal{E}_D \cup \mathcal{E}_E$, is defined as the set of hyperedges that are or in f's prev or in its next.*

$$\forall e, f \in \mathcal{E}_D \cup \mathcal{E}_E . e \in adj(f) \Leftrightarrow e \in prev(f) \cup next(f)$$

It is immediate to test that every hypergraph obtained by the $G\mathcal{H}$ morphism statisfies the following property, which states that each hyperedge $\mathcal{E}_E$ is preceeded and followed by another hyperedge in $\mathcal{E}_D$:

HG2 $\forall e \in \mathcal{E}_E . \forall f \in adj(e) . f \in \mathcal{E}_D$

since $\forall (a, b) \in E$ has $a, b \in V$ where each vertex is mapped in the $\mathcal{E}_D$. Moreover it is also simple to prove that each hypergraph obtained by the $G\mathcal{H}$ morphism statisfies the other following property, that isn't generally true for any data model hypergraphs, which states that each hypergraph's $\mathcal{E}_E$ is linked by only one hyperedge in the *prev* set and only one in the *next* one:

HG3 $\forall e \in \mathcal{E}_E . \exists ! f, g \in \mathcal{E}_D . \{ f \} = prev(e) \wedge \{ g \} = next(e)$

Concerning the hypergraph to graph transformations, [ABB06; SJY08] provide some basic transformations for undirected hyperedges.

2.1.2.1 *On Graph and Hypergraph visit algorithms*

Our aim is to define a subset of all the possible hypergraphs where, given a graph visit algorithm of cost $O(|V| + |E|)$, we have a visit cost of $O(|\mathcal{E}_E| + |\mathcal{E}_D|)$. Let's recall a simple graph DFS algorithm [BM10] in an Ocaml-like pseudocode:

```
module VQ = Queue.Make(Node);;
module EQ = Queue.Make(Edge);;

let gdfs (G:Graph) (r:Node) = {
    s:=VQ.empty;
    s:=VQ.push r s;
    r.mark();
    while (VQ.length s > 0) do {
      u:=S.pop s;
      E.iter (fun e ->
```

```
        e.mark();
        if (Vertex.notmark (Edge.next e)) then
          gdfs G (Edge.next e);
      ) (Node.out s);
    }
}
```

Algorithm 2.1: DFS in graphs

If an hypergraph satisfies the (HG2) condition, then we could define a similar visiting algorithm for hypergraphs as follows:

```
module V = Queue.Make(E_D);;
module E = Queue.Make(E_E);;

let hdfs (G:Graph) (r:E_D) = {
    s:=V.empty;
    s:=V.push r s;
    r.mark();
    while (V.length s > 0) do {
      u:=S.pop s;
      E.iter (fun e ->
        if (E_E.notmark e) {
          e.mark();
          E.iter (fun d ->
            if (E_D.notmark d) then
              hdfs G d;
          ) (E_E.next e);
        }
      ) (E_D.next s); (* next is the next previously defined *)
    }
}
```

Algorithm 2.2: DFS for hypergraphs with (HG2)

In this case, we have a computational cost of $O(|\mathcal{E}_E| + |\mathcal{E}_D|)$, since no vertex is examined, and only hyperedges are traversed. By the way, Data mining operations could produce hyperedges $e \in \mathcal{E}_E$ where $\exists f \in \mathcal{E}_E . f \in next(e)$. Given that such operations normally produce distinct vertices and distinct edges, in those cases we prefer to use a generic hypergraph visiting algorithm as the one proposed in [Gal+93], which is $O(\mathcal{H})$, and makes necessary to use the original hypergraph structure.

```
Procedure HVisit(r, H ):
begin
  for each i ∈ V do Pv[i]:= 0;
  for each E_j ∈ E_E ∪ E_D do Pe[E_j]:= 0;
  Pv[r]:= nil; Q:= {r};
  repeat
    i:=Q.pop();
```

```
    for each E_j ∈ next(i) (* each edge which has the vertex i in its tail *)
      if Pe[E_j]=0 do
        begin
          Pe[E_j] := i ;
          for each h∈H(E_j) do
            if Pv[h]=0 do begin
              Pv[h] := E_j;
              Q := Q ∪ {h}
            end
        end
  until Q = ∅
end.
```

Algorithm 2.3: DFS for generic hypergraphs

At this point we could use the hypergraph data model to map the data and the correlated data from databases or graphs, while we'll keep the basic hypergraph in order to represent correlations between the data properties. This will be clearer with Example 19 on page 79.

2.1.3 *Database with Uncertain Data vs. Typed Adjacency Matrix*

The reason why we introduce this other analysis will be more clear on Chapter 5 on page 65, where we introduce the relational operators over the hypergraph and hence we'll change the original hypergraph data structure that was here previously introduced. First, let's introduce a database where we introduce an uncertainty measure $w\colon E \to [0,1]$:

Definition 2.1.14 (Database with Uncertain Data)**.** *A **Database with Uncertain Data** (UD) is defined as a collection of tables ($DB \in C$), where each table $t \in DB$ is defined as an object ($t \in O$) formed by entries $e \in t$ that are entities ($e \in E$) defined as $e = \langle \mathbb{S}(e), \mathbb{T}(e), \mathbb{D}(e), w(e), \varphi(e) \rangle$ with type $\mathbb{T}(e)$ and schema $\mathbb{S}(e)$. Moreover the following axioms must be true:*

$$DB1 \quad \forall t, t' \in DB.t = t' \Leftrightarrow \mathbb{T}(t) = \mathbb{T}(t')$$

$$DB2 \quad \forall t \in DB.\forall e, e' \in t.\mathbb{T}(e) = \mathbb{T}(e') \wedge \mathbb{S}(e) = \mathbb{S}(e') \wedge \mathbb{T}(e) = \mathbb{T}(t)$$

From now on, we'll assume to treat always database with uncertain data, and hence we will refer to them as *databases*. Our final aim is to show that the *Typed Adjacency Matrix* (abbreviated as TAM) is isomorphic to such database. In order to do so, we need to define a w_A weight function and a $\mathbb{T}_A$ type function for each row in the TAM. Since this TAM's extensions are trivial and involve the fact that $w_A \equiv w$ and $\mathbb{T}_A \equiv \mathbb{T}$ since they use the same types of data, we won't re-define the UD data structure and then suppose to store those function in the work environment.

Since we want to later prove the isomorphism of the two data structures, we have to define the inverse function of the A operator.

Definition 2.1.15 (Inverse Typed Adjacency Matrix Transformation). *The A^{-1}: TAM $\rightarrow$ HDM operator is defined as follows:*

$$A^{-1}(\mathcal{A}) = (amu(\mathcal{A}, \mathcal{E}_D), amu(\mathcal{A}, \mathcal{E}_E))$$

where amu is defined as:

$$amu(\mathcal{A}, Z) = \left\{ \left\langle \bigcup_{x_i \in X} \{ t_i \mid x_i : t_i \}, \mathbb{T}(e), X, w(e) \right\rangle \;\middle|\; j \in rows(\mathcal{A}), X = \{ x_i \mid t \in cols(\mathcal{A}), \mathcal{A}_{jt} \neq \varepsilon \}, j \in Z \right\}$$

Definition 2.1.16 (Database↔TAM morphisms). *Given a database db and a TAM $\mathcal{A}$, we define the function dbA (that permits to transform a database into a TAM) and Adb (vice versa) as follows:*

$$dbA(db) = A(DB\mathcal{H}(db)) \quad Adb(\mathcal{A}) = \mathcal{H}DB(A^{-1}(\mathcal{A}))$$

2.2 SOME PROOFS

2.2.1 *Database and $\mathcal{E}_D$-Hypergraph isomorphism*

We want to show how databases are isomorphic to hypergraphs composed of $\mathcal{E}_D$ edges only: this proof has the aim to show how such hypergraphs could well represent n-ary relations.

Lemma 2.1.

$$\mathcal{H}DB(DB\mathcal{H}(db)) = db$$

Proof.

$$\mathcal{H}DB(DB\mathcal{H}(db)) = \{ \{ \langle \mathbb{S}(e), \mathbb{T}(e), U(e) \rangle \mid e \in \mathcal{E}_D \cup \varnothing \wedge \mathbb{T}(e) = tt \} \mid tt \in A(DB\mathcal{H}(db)) \}$$

where $\mathcal{E}_D \equiv \{ \langle \mathbb{S}(e), \mathbb{T}(e), ID(e) :: e \backslash ID(e), 0 \rangle \mid t \in db \wedge e \in t \}$

$$\mathcal{H}DB(DB\mathcal{H}(db)) = \{ \{ \langle \mathbb{S}(e), \mathbb{T}(e), e \rangle \mid t \in db \wedge e \in t \wedge \mathbb{T}(e) = tt \} \mid tt \in \{ \mathbb{T}(e) \mid t \in db \wedge e \in t \} \}$$

By (DB2) $\mathbb{T}(e) = tt \Rightarrow \mathbb{T}(e) = \mathbb{T}(t)$, and hence:

$$\mathcal{H}DB(DB\mathcal{H}(db)) = \{ \{ \langle \mathbb{S}(e), \mathbb{T}(t), e \rangle \mid t \in db \wedge e \in t \wedge \mathbb{T}(t) = tt \} \mid tt \in \{ \mathbb{T}(t) \mid t \in db \wedge e \in t \} \}$$

By (DB1) we have that only a single table has a given type tt, and hence:

$$\begin{aligned} \mathcal{H}DB(DB\mathcal{H}(db)) &= \{ \{ \langle \mathbb{S}(e), \mathbb{T}(t), e \rangle \mid t \in db \wedge e \in t \} \mid t \in db \} \\ &= \{ \{ \langle \mathbb{S}(e), \mathbb{T}(e), e \rangle \mid e \in t \} \mid t \in db \} \\ &= \{ \{ e \mid e \in t \} \mid t \in db \} \\ &= \{ t \mid t \in db \} \\ &= db \end{aligned}$$

□

By the way, we can't prove that $DB\mathcal{H}(\mathcal{H}DB(\mathcal{E}_D, \mathcal{E}_E)) = (\mathcal{E}_D, \mathcal{E}_E)$ because we assume that a standard database doesn't allow to distinguish correlations by data relations. We could prove the former statement only if we impose that the head of a given hyperedge contains its ID 1 by default, and if the hypergraph hyperedges' weight function are set to 1 by default.

Lemma 2.2.

$$DB\mathcal{H}(\mathcal{H}DB(\mathcal{E}_D, \mathcal{E}_E)) = (\mathcal{E}_D \cup \mathcal{E}_E, \varnothing) \wedge \forall e \in \mathcal{E}_D \cup \mathcal{E}_E.\ ID(e) = H(e)$$

Proof. Given that $\forall e \in \mathcal{E}.ID(e) = H(e)$ by initial hypothesis we have that $T(e) = e \backslash ID(e)$. Moreover:

$$db = \mathcal{H}DB(h) = \{\, \{\, \langle \mathbb{S}(e), \mathbb{T}(e), U(e) \rangle \mid e \in \mathcal{E}_D \cup \mathcal{E}_E \wedge \mathbb{T}(e) = tt \,\} \mid tt \in A(h) \,\}$$

and $DB\mathcal{H}(db) = (\{\, \langle \mathbb{S}(e), \mathbb{T}(e), ID(e) :: e \backslash ID(e) \rangle \mid t \in db \wedge e \in t) \,\}, \varnothing)$

The last formula could be rewrote as:

$$\begin{aligned} \mathcal{E}'_D &\equiv \{\, \langle \mathbb{S}(e), \mathbb{T}(e), H(e) :: T(e) \rangle \mid tt \in A(h) \wedge e \in \mathcal{E}_D \cup \mathcal{E}_E \wedge \mathbb{T}(e) = tt \,\} \\ &\equiv \{\, \langle \mathbb{S}(e), \mathbb{T}(e), H(e) :: T(e) \rangle \mid e \in \mathcal{E}_D \cup \mathcal{E}_E \,\} \\ &\equiv \mathcal{E}_D \cup \mathcal{E}_E \end{aligned}$$

The statement is proofed immediately. □

Corollary 2.1.

$$DB\mathcal{H}(\mathcal{H}DB(\mathcal{E}_D, \varnothing)) = (\mathcal{E}_D, \varnothing) \wedge \forall e \in \mathcal{E}_D.\ ID(e) = H(e)$$

Proof. Immediate by previous lemma. □

At this point,

Theorem 2.1. *Given the set $\mathcal{H}'$ of all the hypergraphs with only $\mathcal{E}_D$ hyperedges, the $DB\mathcal{H}$ and $\mathcal{H}DB$ functions with the HDM and $\mathcal{H}'$ datatypes form an isomorphism.*

Proof. The proof is immediate given the Lemma 2.1 and the Corollary 2.1. □

2.2.2 *The A, A^{-1} isomorphism*

In order to proof the isomorphism in the next subsection, we have to find that $\forall \mathcal{H} \in HDM.A^{-1}(A(\mathcal{H})) = \mathcal{H}$ and $\forall \mathcal{A}.A(A^{-1}(\mathcal{A})) = \mathcal{A}$.

Lemma 2.3.

$$\forall \mathcal{H} \in HDM.A^{-1}(A(\mathcal{H})) = \mathcal{H}$$

Proof. We can rewrite $\mathcal{H} \equiv (\mathcal{E}_D, \mathcal{E}_E)$; giving Definition 2.1.15 on the preceding page we have that $A^{-}1(A(\mathcal{H})) = (amu(A(\mathcal{H}), \mathcal{E}_D), amu(A(\mathcal{H}), \mathcal{E}_E))$, where we could see that the *amu* function selects each row and recreates the original hyperedge in $\mathcal{H}$, since each $A(\mathcal{H})$ row represents a tuple in $\mathcal{E}_D \cup \mathcal{E}_E$. □

Lemma 2.4.

$$\forall \mathcal{A} \in TAM.A(A^{-1}(\mathcal{A})) = \mathcal{A}$$

Proof. For any $\mathcal{A}$ we have that the lemma is proved iff. $\forall j \in \mathcal{E}_D \cup \mathcal{E}_E.\forall t \in \mathcal{T}.A_{jt}(A^{-1}(\mathcal{A})) = \mathcal{A}_{jt}$. If $\mathcal{A}_{jt} = \varepsilon$, then or the j doesn't exists as a row (and hence it won't exists as a hyperedge in the $A^{-1}(\mathcal{A})$) or $t \notin \mathbb{S}(j)$ (and hence no value of such type will be inserted in the j hyperedge in $A^{-1}(\mathcal{A})$). In the other case, we have that $j \in \mathcal{E}_D \cup \mathcal{E}_E$ and $t \in \mathbb{S}(j)$ and hence the $\mathcal{A}_{jt}$ element will be part of the j hyperedge in $A^{-1}(\mathcal{A})$ that will be consequently remapped in $\mathcal{A}_{jt}$ with the A transformation. □

Theorem 2.2. *The morphisms A, A^{-1} with the HDM and TAM datatypes form an isomorphism*

Proof. The proof is trivial thanks to the two previous lemmas. □

2.2.3 *The Adb, dbA isomorphism*

In this subsection we want to show how the databases are isomorphic to TAM with only data rows, that is rows that don't map relations ($\forall r \in rows(\mathcal{A}).r \in \mathcal{E}_D$).

Lemma 2.5.

$$\forall r \in rows(\mathcal{A}).r \in \mathcal{E}_D. \Rightarrow A^{-1}(\mathcal{A}) = (\mathcal{E}_D, \varnothing)$$

Proof. If $\mathcal{A}$ doesn't contain rows belonging to $\mathcal{E}_E$, all the rows will be mapped in a $\mathcal{E}_D$ hyperedge in a resulting hypergraph by A^{-1} definition. □

Lemma 2.6. *For each database d, $Adb(dbA(d)) = d$.*

Proof. By rewriting we have that $Adb(dbA(d)) = \mathcal{H}DB(A^{-1}(A(DB\mathcal{H}(d))))$; if we apply the Lemma 2.3 on the preceding page ($\forall \mathcal{H} \in HDM.A^{-1}(A(\mathcal{H})) = \mathcal{H}$) we obtain that $\mathcal{H}DB(A^{-1}(A(DB\mathcal{H}(d)))) = \mathcal{H}DB(DB\mathcal{H}(d))$ and by Lemma 2.1 on page 31 ($\mathcal{H}DB(DB\mathcal{H}(db)) = db$) we have that the lemma is proved. □

Lemma 2.7.

$$\forall \mathcal{A} \in TAM. \quad (\forall r \in rows(\mathcal{A}).r \in \mathcal{E}_D) \Rightarrow dbA(Adb(\mathcal{A})) = \mathcal{A}$$

Proof. By rewriting we have that $dbA(Adb(\mathcal{A})) = A(DB\mathcal{H}(HDB(A^{-1}(\mathcal{A}))))$; given that the current lemma satisfies the conditions of Lemma 2.5, when we have that $A(DB\mathcal{H}(HDB(A^{-1}(\mathcal{A})))) = A(DB\mathcal{H}(HDB(\mathcal{E}_D, \varnothing)))$. Moreover by Corollary 2.1 we have that $A(DB\mathcal{H}(HDB(\mathcal{E}_D, \varnothing))) = A(\mathcal{E}_D, \varnothing)$, and hence we have to prove that $A(\mathcal{E}_D, \varnothing) = \mathcal{A}$. Since A^{-1} is a function and then $\forall x, y.f(x) = f(y) \Rightarrow x = y$, then $A^{-1}(A(\mathcal{E}_D, \varnothing)) = A^{-1}(\mathcal{A})$ we obtain that $A^{-1}(A(\mathcal{E}_D, \varnothing)) = (\mathcal{E}_D, \varnothing)$ by the Theorem 2.2 and then $A^{-1}(\mathcal{A}) = (\mathcal{E}_D, \varnothing)$ from Lemma lem:Amensimpl. □

Theorem 2.3. *There is an isomorphism between the database and the TAM with no correlational rows (that is, we chose $\forall T \in TAM.rows(T) \subseteq \mathcal{E}_D$).*

Proof. Via the two previous lemmas. □

At this point we could not use the TAM representation, and see the dbA as a different view over the database data. If we evaluate the $dbA(db)$ function we could simplify the notation and then obtain the following result:

$$dbA_{jt}(db) = A_{jt}(\mathcal{H}DB(db)) = \begin{cases} v & \exists v.v \in U(j) \wedge v : t \wedge e \in \tau \wedge \tau \in db \\ \varepsilon & oth. \end{cases}$$

This also implies that the cost of accessing the data via database is the same of accessing it on a matrix form that is often used for representing data in hypergraphs. At this point we'll choose to provide a database representation for the $\mathcal{E}_D$ relations, in order to apply albebric operations over data. We'll also use the dbA function when we need to provide a read-only view over the data. This theorem will bring us to the Definition 5.1.3 on page 67.

3

RELATIONAL DATA MINING

Contents

Many researchers committed in KDD have published some guidelines in order to automate the process of knowledge discovery and to make it resemble to a software engineering process - an example of such framework is proposed in [Deb+99]. In this section we suggest another way [Cal+06] to automate the KDD discovery that is data driven and uses operations over data that, by the way, hasn't been examined or implemented yet. At the end of the chapter we'll also explain some other statistical techniques that are already used for data mining purposes.

3.1 PRELIMINARY RELATIONAL ALGEBRAIC OPERATIONS OVER DATA TABLES WITH UNCERTAIN DATA

- **Why do we have to extend the record definition with the indexing function, that is $e \equiv \langle \mathbb{S}(e), \mathbb{T}(e), \mathbb{D}(e), w(e), \varphi(e) \rangle$? This choice gets the data model more complicated.** The main benefit of the indexing function φ will be later evident in the "tensor-like representation" of the correlational relations over the data $\mathcal{E}_E$, by which we could store on each result entry via dovetailing which entities took part in generating it and we'll only add a scanning linear time of the previous database representation to update the tensor, and hence we could lower the time complexity. Another advantage is that this indexing will permit to define this latter tensor-like representation in a more easy way,

since it allows to implement the operations on the database and on tensor in separate steps.

- **Apparently, there are still some problem with your indexing update definition, as you have also to check if the indices are consistent or not.** The only real index-collision problem could be observed in the union function, where we could merge databases with a different indexing structure. At this point we shall assume that each hypergraph, and hence its database-plus-tensorial representation, have different index values, as we assume that each hypergraph is "nestled" in the same work environment. In order to show that each algebric operation has consistent indices, for each algebric operation we have to prove that is "index-consistent".

Definition 3.1.1 (Index-consistency). *A database unary operation $\dot{\triangleright}$ is said to be **index-consistent** iff. for all the tables of the current database the indices among the tables are kept distinct.*

*A database binary operation $\dot{\bowtie}$ is said to be **index-consistent** iff. it generates a database where the indices among the tables are kept distinct.*

We recall Definition 2.1.14 on page 30 where we showed the concept of database with uncertain data. [MM12] defined the Join operation for these datasets through some SQL queries: now we want to define them inside the relational algebra. We could see that some operations, like π, σ, $Calc$ and ρ don't change the weight of the relations, since they don't "structurally modify" the data. At this point we have only to modify the probabilities of some other operations, like $\cup$, $\bowtie$ and Γ (since in this case the data is aggregated and hence transformed) drawing on the previous considerations concerning the index update of the transformed tables.

This measure is necessary for reasons that will be clear in Subsection 5.1.2 on page 68.

Definition 3.1.2 (Union). *The **union** operator over a set of tables T with the same schema and size $|T| = n$ $(\forall t \neq t' \in T.\mathbb{T}(t) = \mathbb{T}(t'))$ is defined as follows:*

$$\bigcup T = \{ \left\langle \mathbb{S}(e), \mathbb{T}(T_1) \cdot '' \vee \cdots \vee '' \cdot \mathbb{T}(T_n), e, \operatorname{avg} Map(E, w), \overrightarrow{dt}(Map(E, \varphi)) \right\rangle | \\ E = \{e, e' | t, t' \in T, e \in t, e' \in t' \wedge \mathbb{D}(e) = \mathbb{D}(e')\}\}$$

In the following examples, $\mathbb{S}$ and $\mathbb{T}$ representations are omitted, in order to focus more on the changes occurring on the data.

Example

4. *Given two tables T_1 and T_2 respectively defined as follows:*

Num1	*Num2*	*Num3*	*w*	φ
1	2	3	1	1
1	2	6	1	2
1	4	3	1	3

Num1	*Num2*	*Num3*	*w*	φ
1	2	3	1	4
1	2	6	0.8	5
1	2	1	0.8	6

the result of $\cup T_1, T_2$ *is the following one:*

Num1	*Num2*	*Num3*	w	φ
1	2	3	1	250
1	2	6	0.9	663
1	4	3	0.1	13
1	2	1	0.8	34

Definition 3.1.3 (Selection)**.** *The* ***selection*** *operator over a table* t *is defined as follows:*

$$\sigma_P(t) = \{ \langle \mathbb{S}(e), \mathbb{T}(e), \mathbb{D}(e), w(e), \varphi(e) \rangle \mid e \in t \wedge P(e) \}$$

Definition 3.1.4 (θ-Join)**.** *The* ***theta join*** *operator over two tables* t *and* t' *is defined as follows:*

$$t \bowtie_\theta t' = \left\{ \left\langle \mathbb{S}(e) \cup \mathbb{S}(e'), \mathbb{T}(e) \cdot \mathbb{T}(e'), e \cup e', w(e)w(e'), dt(\varphi(e), \varphi(e')) \right\rangle \;\middle|\; e \in t, e' \in t', \theta(e, e') \right\}$$

Example

5. *Given two tables* T_1 *and* T_2 *respectively of schema* A *and* B *defined as follows:*

Num1	*Num2*	w	φ
1	2	1	1
1	4	1	2

Num2	*Num3*	w	φ
2	3	1	3
2	6	0.8	4
2	1	0.8	5

the result of $T_1 \bowtie_{A.Num2=B.Num2}$ *is the following one:*

Num1	*Num2*	*Num3*	w	φ
1	2	3	1	13
1	2	6	0.8	19
1	2	1	0.8	26

Definition 3.1.5 (Projection)**.** *The* ***projection*** *operator over a table* t *with schema* $\mathbb{S}(t)$ *over the data types* $\vec{L}$ *is defined as follows:*

$$\pi_{\vec{L}}(t) = \{ \left\langle \vec{L} \cap \mathbb{S}(e), \vec{L} \cdot "in" \cdot \mathbb{T}(e), \left\{ x_i \;\middle|\; x_i \in e \wedge x_i : t \wedge t \in \vec{L} \right\}, p, i \right\rangle |$$
$$e \in t \wedge P = Filter(t, x \mapsto \pi_{\vec{L}}(\{e\}) = \pi_{\vec{L}}(\{x\})) \wedge w = \text{avg } Map(P, w) \wedge i = \overrightarrow{dt}(Map(P, \varphi))\}$$

Example

6. *Given a table T_1 defined as:*

Num1	*Num2*	*Num3*	w	φ
1	2	3	0.4	1
1	2	6	0.6	2
1	4	3	1	3

the result of $\pi_{Num1,Num2}(T_1)$ is the following one:

Num1	*Num2*	w	φ
1	2	0.5	63
1	4	1	13

Definition 3.1.6 (Rename)**.** *The **rename** operator over a table t with schema $\mathbb{S}(t)$ is defined as follows:*

$$\rho_{R \leftarrow X}(t) = \pi_{\mathbb{S}(t) \setminus \{X\}}(Calc_{x \mapsto x.X \text{ as } R}(t))$$

where Calc is made explicit in Definition 3.2.1.

Definition 3.1.7 (Embedding)**.** *The **embedding** operator over the records t of a table T as defined in [MM06] extends a value v as a X field:*

$$\varepsilon_{X \leftarrow v}(T) = \{ \langle \mathbb{S}(e) \cup \{ X \}, X \cdot \mathbb{T}(e), t \cup \{ v : X \}, w(t), \varphi(t) \rangle \mid t \in T \}$$

3.2 PREFACE ON THREEFOLDED DATA MINING (TDM)

Let's use an algebra for data mining as expressed in [Cal+06]. This model, introduced in [JLNoo], defines a multistep process where each output of a mining operation should be used as a input for the following, in order to acheive Knowledge Discovery with a one-shot activity that involves also decision trees, data partitioning, aggregation and transformation.

This could be acheived throug a division of the data in three communicating worlds [Cal+06], named Data-World (**D-World**), Intensional-World (**I-World**) and Extensional-World (**E-World**). The *Data World* is defined exactly as the traditional Relational Algebra over tuples t-s and relational schemas $\mathcal{R}(B_1, \ldots, B_n)$ where each attribute $B_1 \in \mathcal{A}$ has a domain $dom(B_1)$. By

Col1	Col2	Col3	w	φ
A	2	4	0.1	1
B	8	10	0.2	2
C	15	20	0.3	3
A	28	3	0.4	4
C	1	3	0.5	5

(a) $\mathcal{T}$

Col1	Col2	Col3	S	w	φ
A	2	4	6	0.1	1
B	8	10	18	0.2	2
C	15	20	35	0.3	3
A	28	3	31	0.4	4
C	1	3	4	0.5	5

(b) $Calc_{Col2+Col3 \text{ as } S}(\mathcal{T})$

Col1	R	w	φ
A	30	0.25	44548
B	8	0.2	53
C	16	0.4	327642

(c) $\Gamma^{\text{Sum}(Col2) \text{ as } R}_{\langle Col1 \rangle}(\pi_{\{Col1,Col2\}}(\mathcal{T}))$

Figure 3.1: Examples of *Calc* and Γ operations over a table $\mathcal{T}$.

detailing the *D-World* definition, we can see that it could be extended with the functions given in the following definition:

Definition 3.2.1 (D-World Functions)**.** *Inside the D-World algebra, [Cal+06] defines some arithmetic operations over the tuples are defined as:*

$$Calc_{Op(\vec{L}) \text{ as } S}(T) = \left\{ \left\langle \mathbb{S}(e) \cup \{S\}, S \cdot \mathbb{T}(e), e \cup \{Op(e[\vec{L}]) : S\}, w(e), \varphi(e) \right\rangle \;\middle|\; e \in T \right\}$$

An aggregation operation similar to SQL's `group by` *could be defined as:*

$$\begin{aligned}\Gamma^{\oplus(X_i) \text{ as } S}_{\langle \vec{L} \setminus \{X_i\} \rangle}(D) = \{ \Big\langle & \vec{L} \setminus \{X_i\}, "Aggr" \cdot X_i \cdot "over" \cdot \vec{L} \setminus \{X_i\} \cdot "in" \cdot \mathbb{T}(e), x[\vec{L} \setminus \{X_i\}] \cup \{s : S\}, w, i \Big\rangle \mid \\ & e \in t \wedge P = Filter(t, x \mapsto x[\vec{L}] = e[\vec{L}]) \wedge s = \oplus Map(P, x \mapsto x[X_i]) \\ & i = \overrightarrow{dt}(Map(P, \varphi)) \wedge w = \text{avg}\, Map(P, w) \\ & \}\end{aligned}$$

where $X_3 \notin \{ X_i, \ldots, X_j \}$*. We have to remark that the* $\langle X_1, \ldots, X_n \rangle = \langle\rangle$ *means that the aggregation is performed over all the values with type* X_i*.*

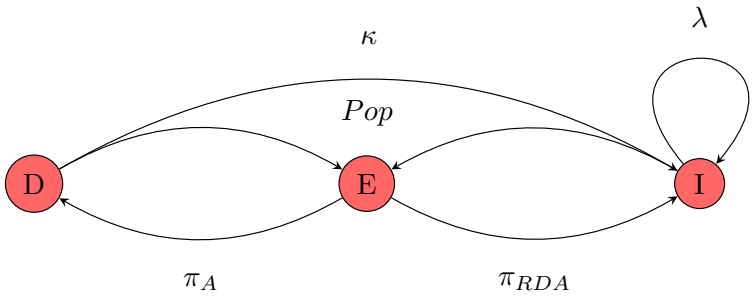


Figure 3.2: *Original Data Mining algebra in 3W representation ([Cal+06]).*

Example

7. *Let's take a look at Figure 3.1: given table $\mathcal{T}$ provided in Subfigure (a), we want to express the previous non-relational operations by providing some examples; $Calc_{Col2+Col3\ \texttt{as}\ s}(\mathcal{T})$ returns the table (b), while (c) provides an example of the grouping operation.*

[Cal+06] provides a tabular representation of a decision tree, where the region defines the terms under which the classification applies. Each region is organized as a set of clauses $r = \{ p_i \}_{i \in \mathbb{N}}$ where p_i are predicates over the relational data.

For the tables defined in this world, they define the same traditional relational operations, with some *grouping* and *ungrouping* operations through which could also define more complex operators, as showed in [Cal+06] for the set difference.

We could also define the classification as a couple of k-ary functions (p^k, c^k) where if $p^k(\vec{a})$, then there is a classification function $c^k(\vec{a})$ that returns a classification value. This definition maps $r \mapsto p$ and vice versa.

[Woo12] points out how «*aggregation functions play an essential role in network analysis, while the ability to transform network by creating new nodes based on aggregations of set of existing nodes is also crucial*». The mined data could be aggregated through the Γ operator given in Definition 3.2.1 on the previous page.

Inside the *Extensional-World*, we could want to link the clauses given in the previous world to the relations given in the first. Concerning this world, they define for intra-world operators the Γ aggregator and the σ selection as in the D-World.

Definition 3.2.2 (Clause classification over tuples). *A tuple t satisfies every clause in the extended region $r = \{ c_1, \ldots, c_n, c \}$ (expressed like $[\![t]\!]^r$) iff. given a* ***classification*** *(p, c) over a*

common classification function c and a predicate $p = \bigwedge_{i \leqslant n} p_i$, then $p(t)$. Finally we could define that t is classified as $c(t)$ towards p as follows:

$$[\![t]\!]^p \Rightarrow c(t)$$

The mining loop operator is defined in order to obtain interesting properties from the rules that couldn't be directly extracted over the data. This operator provides a general `while-condition` scheme, that could be implemented in any kind of algorithm over inferenced rules.

Definition 3.2.3. *A **mining loop** $\lambda_{init,\Delta R,\mathcal{A}}$ operator over clauses [Cal+06] is a general instantiation of the following scheme:*

```
R ← init
while (ΔR) {
    R ← A(R)
}
return R
```

Algorithm 3.1: Mining Loop operator schema

Example

8. *Frequent Itemset as λ.* *[Cal+06] provides an example of how we could use the mining loop operators with other one previously discussed in order to provide an algebric implementation of the frequent-itemset technique.*

```
FIinit(Items) {
  return (∅, κ_{Product∋memberOf}(Unnest_{SetOf<Product>}(Items)))
}
FIΔR ← C ≠ ∅ in (F,C)

FIA_θ(F,C) {
  F ← F ∪ σ_{supp≥θ} ( Γ^{COUNT(SetOf<Product>) as supp}_{<Set>} (Pop(C,D)) )
  C ← getCan(F)
  return (F,C)
}

MFSG_θ ≡ λ_{FIinit,FIΔR,FIA_θ}
```

where $genCan$ is an algebric function as depicted in the same quoted article to generate the next candidates.

Some other operations could be defined in order to obtain clauses from the data (κ), to join the claueses with data that satisfy the first (Pop), for then splitting data (π_A) from intensional

A	B	C	w	φ
2	2	4	0.4	1
5	1	3	0.2	2

(a) $\mathcal{T}$

R	w	φ
$B + C \leqslant 6$	0.4	4
$B + C \leqslant 4$	0.2	8

(b) $\kappa_{B+C \leqslant val(B)+val(C) \text{ as } R}(\mathcal{T})$

Figure 3.3: An example of the application of the regionizing operator κ for mining relations over the data. Weights and indices are omitted

data (π_{RDA}). A general overview of these [Cal+06] Data-Mining operations could be viewed in Figure 3.2 on page 40 and in the following definition:

Definition 3.2.4 (DM Extraworld operations). *[Cal+06] defines a **populating function** $Pop(D, R)$, where D is in the D-World and R in I-World tables, gives an E-Relation with a domain $dom(D) \cup dom(R)$ and:*

$$Pop(D, R) = D \bowtie_{t_d, t_r \mapsto [\![t_d]\!]^{t_r}} R$$

We shall also define a π_{RDA} projection towards E-World attributes and a π_A towards D-World attributes.

*We define a **regionizing operator** $\kappa_{f_{A \ldots Z} \text{ as } R}$ over Ds where, given a D-World relation, we define a I-World region over the function described in $\circledast$ and f is defined as a relation between some operations over labels $A \ldots Z$ and some values $a \ldots z$associated to the previously given labels:*

$$f_{A \ldots Z}(a, \ldots z) = \Re(A *_1 \cdots *_n Z, a *_{n+1} \cdots *_{n+m} z) \equiv \Re(Op_{\circledast}(A \ldots Z), op_{\circledast}(a \ldots z))$$

in order to obtain a set of properties:

$$\begin{aligned} \kappa_{f_{\vec{L}} \text{ as } R}(D) = \{ \langle \{R\}, R, r, 1, i \rangle \mid \\ t \in D \wedge P = Filter(D, x \mapsto f_{\vec{L}}(x[\vec{L}]) = r) \wedge r = f_{\vec{L}}(t[\vec{L}]) \wedge \\ \wedge\, i = \overrightarrow{dt}(Map(P, \varphi)) \wedge w = \text{avg}(Map(P, w)) \wedge \} \end{aligned}$$

where to each property p_i in the given set could be associated a common classification function c over $A \ldots Z$.

We could easily extend the given definition by association to each property of the given set a same classification function c.

Example

Region	Class
{shp='bell',odor='none'}	{pois='f'}
{shp='bell',odor<>'none'}	{pois='t'}
{shp<>'bell',col='red'}	{pois='t'}
{shp<>'bell',col<>'red'}	{pois='f'}

(a) $\mathcal{T}$

shp	col	odor	pois
bell	red	none	f
bell	yellow	none	f
flat	yellow	ansie	t
flat	green	none	f

(b) *Testing*

Region	Class	shp	col	odor	pois
{shp='bell',odor='none'}	{pois='f'}	bell	red	none	f
{shp='bell',odor='none'}	{pois='f'}	bell	yellow	none	f
{shp<>'bell',col<>'red'}	{pois='f'}	flat	green	none	f

(c) $Pop(\mathcal{T}, Testing)$

Figure 3.4: An example of the *Pop* operator between the data table *Testing* and the intensional table $\mathcal{T}$.

9. *[Cal+06] provides an example of the application of the previously defined operators: Figure 3.3 provides an example for the regionizing operator, while Figure 3.4 shows why the Pop operator could be interpreted as a join over properties ($\mathcal{T}$) and some data (Testing).*

3.3 DATABASE OPERATIONS

We would like to extent all these operations to databases. Since all the previous $\triangleright$ operations were table-based, we'll have naturally that any database operation $\dot{\triangleright}$ could be seen as an application of $\triangleright$ to all its tables:

$$\dot{\triangleright}(DB) = \{\, \triangleright(t) \mid t \in DB \,\}$$

If we have a general binary operator $\bowtie$ among data tables, this could be generalised with a $\dot{\bowtie}$ defined as follows:

$$DB_1 \dot{\bowtie} DB_2 = \{\, d_1 \bowtie d_2 \mid d_1 \in DB_1 \wedge d_2 \in DB_2 \,\}$$

In some cases, we have also to check if we could perform the $\triangleright$ to all databases' entries. For instance, we must redefine some operations that are schema dependant as follows.

Definition 3.3.1 (Reindexing). *Given a table t, the **reindexing** operator changes each record's index $\varphi(r)$ in t with a function $f(\varphi(r))$:*

$$\Phi_f(t) = \{ \langle \mathbb{S}(e), \mathbb{T}(e), \mathbb{D}(e), w(e), f(\varphi(e)) \rangle \mid e \in t \}$$

$$\dot{\bigcup} DB = \left\{ \bigcup T \;\middle|\; T = \{ t, t' \mid db, db' \in DB, t \in db, t' \in db', \mathbb{T}(t) = \mathbb{T}(t) \} \right\}$$

$$\dot{\pi}_{\vec{L}}(DB) = \left\{ \pi_{\vec{L}}(t) \;\middle|\; t \in DB \wedge \vec{L} \cap \mathbb{S}(t) \neq \varnothing \right\}$$

$$\dot{Calc}_{Op(\vec{L})\texttt{as}S}(DB) = \left\{ Calc_{Op(\vec{L})\texttt{as}S}(t) \;\middle|\; t \in DB \wedge \vec{L} \subseteq \mathbb{S}(t) \right\} \cup \left\{ \Phi_{\overrightarrow{dt}}(t) \;\middle|\; t \in DB \wedge \vec{L} \nsubseteq \mathbb{S}(t) \right\}$$

$$\dot{\kappa}_{f_{\vec{L}} \texttt{ as } R}(DB) = \left\{ \kappa_{f_{\vec{L}} \texttt{ as } R}(t) \;\middle|\; t \in DB \wedge \vec{L} \subseteq \mathbb{S}(t) \right\}$$

$$\dot{\Gamma}^{Op(X_i) \texttt{ as } S}_{X \setminus \{X_i\}}(DB) = \left\{ \Gamma^{Op(X_i) \texttt{ as } S}_{X \setminus \{X_i\}}(t) \;\middle|\; t \in DB \wedge X \subseteq \mathbb{S}(t) \right\} \cup \left\{ \Phi_{\overrightarrow{dt}}(t) \;\middle|\; t \in DB \wedge X \nsubseteq \mathbb{S}(t) \right\}$$

3.4 OTHER RELATIONAL STATISTICAL TECHNIQUES

3.4.1 *Classification: Naïve Bayes*

Bayesian classifiers [HKP12; Wu+07] are statistical classifiers that can predict a class membership of a tuple over a particular class. This classification technique uses the Bayes theorem:

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

where $P(H|X)$ could be read as "the probability that event H holds, given that X has appened". For example, we would like to test that a given tuple $t = \langle x_1 \colon T_1, \ldots, x_n \colon T_n \rangle$ belongs to a class c_i of type `Class`, and hence:

$$P(t|c_i) \equiv P(t|\texttt{Class} = c_i) = \prod_{i \leqslant n} P(T_1 = x_1|\texttt{Class} = c_i)$$

Supposed that the c_i is already defined over a dataset d, we could interpret the conditional probability $P(T_i = x_i|\texttt{Class} = c_i)$ for not-continuous data as:

$$P(T_i = x_i|\texttt{Class} = c_i) = \frac{|\sigma_{\texttt{Class}=c_i \wedge T_i = x_i}(d)|}{|\sigma_{\texttt{Class}=c_i}(d)|}$$

If T_i is otherwise a continuous attribute, then probability could be defined with a Gaussian distribution:

$$P(T_i = x_i|\texttt{Class} = c_i) = \frac{1}{\sqrt{2\pi}\sigma_{T_i,c_i}} \exp\left[-\frac{(x_i - \mu_{T_i,c_i:\texttt{Class}})^2}{2\sigma^2_{T_i,c_i:\texttt{Class}}} \right]$$

where the sample mean is defined as $\mu_{T_i,c_i:\texttt{Class}} = \Gamma^{\texttt{avg}(T_i) \texttt{ as } \mu}_{\diamondsuit}(\pi_{T_i}(\sigma_{\texttt{Class}=c_i}(d)))$, and the sample variance is $\sigma^2_{T_i,c_i:\texttt{Class}} = {}^1\!/\!_{|\sigma_{\texttt{Class}=C_i}(d))|} \sum_{t \colon T_i \in \pi_{T_i}(\sigma_{\texttt{Class}=C_i}(d))} (t - \mu_{T_i,c_i:\texttt{Class}})^2$

Owner	Status	Income	Defaulted Borrower
T	Single	125K	F
F	Married	100K	F
F	Single	70K	F
T	Married	120K	F
F	Divorced	95K	T
F	Married	60K	F
T	Divorced	220K	F
F	Single	85K	T
F	Married	75K	F
F	Single	90K	T

Table 3.1: *A small database for a Naïve Bayes example.*

Example

10. *[TSK05] provides some examples concerning the use of this classifier. Given the Table 3.1, we could have that:*

$$P(Income = 120K|\texttt{Defaulted Borrower} = F) = \frac{1}{\sqrt{2\pi}54.54}\exp\left[-\frac{(120-110)^2}{2\cdot 2975}\right] = 0.0072$$

where $\mu_{\texttt{Income},F:\texttt{DefaultedBorrower}} = 110$ *is the mean of all the income values where* $\texttt{Defaulted Borrower} = F$ *and* $\sigma^2_{\texttt{Income},F:\texttt{DefaultedBorrower}} = 2975$*. In order to classify* $X \triangleq \langle \texttt{Owner} = F, \texttt{Status} = Married, \texttt{Income} = 120K\rangle$*, we have to find:*

$$\max_{c\in\texttt{DefaultedBorrower}} \{\, P(X \mid c)\,\}$$

hence we have that:

$$\begin{aligned} P(X|\texttt{DefaultedBorrower} = F) = {} & P(\texttt{Owner} = F|\texttt{DefaultedBorrower} = F)\cdot \\ & P(\texttt{Status} = Married|\texttt{DefaultedBorrower} = F)\cdot \\ & P(\texttt{Income} = 120K|\texttt{DefaultedBorrower} = F) \\ = {} & 4/7 \cdot 4/7 \cdot 0.00072 = 0.0024 \end{aligned}$$

3.5 INDEXING CONSISTENCY OF RELATIONAL OPERATIONS

3.5.1 *Preliminar lemmas*

Before starting to prove the index-consistency property for relational operations , we must prove some lemmas over the dt definition $\overrightarrow{dt}$.

Lemma 3.1.

$$\forall i, j, k, n \in \mathbb{N}. i \neq j \wedge k \neq h \Rightarrow dt(i,k) \neq dt(j,h)$$

Proof. We prove by contradiction: suppose that $dt(i,k) = dt(j,h)$, then $k + S_{i+k} = h + S_{j+h}$. Since $k \neq h$ iff. $k > h \vee h > k$, hence $\exists m, n \neq 0. k = h + m \vee h = k + n$. Similarly $i \neq j$ iff. $\exists a, b \neq 0. i = j + a \vee j = i + b$.

Suppose to prove the case where $k = h + m \wedge i = j + a$. In this case we'll have that:

$$\begin{aligned} k + S_{i+k} = h + S_{j+h} &\Leftrightarrow h + m + S_{j+h+a+m} = h + S_{j+h} \\ &\Leftrightarrow m + S_{j+h+a+m} = +S_{j+h} \\ &\Leftrightarrow m + \sum_{u=j+h+1}^{j+h+a+m} u = 0 \end{aligned}$$

that is impossible because the obtained value is always positive.

Suppose to prove the case where $k = h + m \wedge j = i + b$. Now we have that:

$$\begin{aligned} k + S_{i+k} = h + S_{j+h} &\Leftrightarrow h + m + S_{i+h+m} = h + S_{i+b+h} \\ &\Leftrightarrow m + \sum_{u=i+h+1}^{i+h+m} = \sum_{u=i+h+1}^{i+h+b} u \end{aligned}$$

Now we have to prove by cases:

- If $m > b$ then we have that $\sum_{u=i+h+1}^{i+h+m} - \sum_{u=i+h+1}^{i+h+b} u > 0$ and hence the absurd is given that a positive quantity couldn't be zero.
- If $m < b$ then $m = \sum_{u=i+h+m+1}^{i+h+b} u = i + h + m + 1 + \ldots i + h + b$ and then even in this case we have an always positive quantity that couldn't be zero.
- If $m = b$ then it is impossible that $m = 0$.

The other cases are symmetric. □

Lemma 3.2.

$$\forall x, y, z, t \in \mathbb{N}. x \neq z \vee y \neq t \Rightarrow dt(x,y) \neq dt(z,t)$$

Proof. Suppose to have $x \neq z$. If $y = t$, then we should have that:

$$dt(x,y) \neq dt(z,y) \Leftrightarrow y + S_{x+y} \neq y + S_{z+y}u \Leftrightarrow \sum_{u=y+1}^{y+x} u \neq \sum_{u=y+1}^{y+z} u$$

Since it is obvious that $x < z \Leftrightarrow \sum_{u=y+1}^{y+x} u < \sum_{u=y+1}^{y+z} u$ and $x > z \Leftrightarrow \sum_{u=y+1}^{y+x} u > \sum_{u=y+1}^{y+z} u$, the $y = t$ case is proved. If $y \neq t$, we use the previous lemma (Lemma 3.1).

Suppose to have $x = z$. If $y = t$, then the case is proved by symmetry of the $=_{\mathbb{N}}$ relation. If $y \neq t$, then:

$$dt(x,y) \neq dt(x,t) \Leftrightarrow y + S_{x+y} \neq t + S_{x+t}$$

where it's clear that if $y > t$ then $y + S_{x+y} > t + S_{x+t}$ and similarly if $t < y$. □

Inter alia this last lemma guarantees that any two list of naturals of different size have a different vector dovetailing representation: that could be easily seen since given any two lists l, l' with $|l| \neq |l'|$, we have that $\overrightarrow{dt}(l) = dt(|l|, Rl)$ and $\overrightarrow{dt}(l') = dt(|l'|, Rl')$ by definition.

Corollary 3.1.

$$\forall x, y, z, t \in \mathbb{N}. dt(x,y) = dt(z,t) \Rightarrow x = z \vee y = t$$

Proof. By previous lemma. □

Lemma 3.3.

$$\forall x, y, z, t. x = z \wedge y = t \Rightarrow dt(x,y) \neq dt(z,t)$$

Proof. Obvious by rewriting and symmetry. □

Theorem 3.1.

$$\forall x, y, z, t. x = z \wedge y = t \Leftrightarrow dt(x,y) \neq dt(z,t)$$

A formal demonstration of the following theorem is given via Matita script in Algorithm C.15 on page 167, where we also provide the formal definition of the double induction strategy for lists, that could be defined as:

$$\begin{aligned}\forall R : list \to list \to Prop. (\forall n. R\ []\ n) \to\\ (\forall n, m. R\ n :: m\ []) \to\\ (\forall m, n, o, p. R\ m\ p \to R\ (n :: m)\ (o :: p)) \to\\ (\forall n, m. R\ n\ m)\end{aligned}$$

Now, we could provide a description of that lemma as follows:

Lemma 3.4. *Two lists of the same length have the same vector dovetailing representation iff. they're the same:*

$$\forall l, m. \overrightarrow{dt}(l) = \overrightarrow{dt}(m) \Rightarrow l = m$$

Proof. We could rewrite the main hypothesis by Theorem 3.1 on the preceding page:

$$\overrightarrow{dt}(l) = \overrightarrow{dt}(m) \Leftrightarrow dt(|l|, Rl) = dt(|m|, Rm) \Leftrightarrow |l| = |m| \wedge Rl = Rm$$

This means that we have to prove that, for two chosen lists l and m:

$$|l| = |m| \Rightarrow Rl = Rm \Rightarrow l = m$$

We could treat this property that we want to prove as the one over which apply the double induction strategy for lists. In this case we have to prove:

- $\forall n.|[]| = |n| \Rightarrow R[] = Rn \Rightarrow [] = n$

  After choosing l as a particular n list, we have that from the first hypothesis that $0 = |L|$ iff. $L = []$, and then it is immediate to prove the final goal $[] = L$.

- $\forall n, m'.|n :: m'| = |[]| \Rightarrow R(n :: m') = R[] \Rightarrow n :: m' = []$

  Given that from the first hypothesis $|n :: m'| = 1 + |m'| = 0 = |[]|$, we could prove $n :: m' = []$ by absurd since it is clear that $1 + |m'| \neq 0$.

- At this point, we gain for lists $B \subset l$ and $D \subset m$ and naturals $A, B \in \mathbb{N}$ that:

  $$(|B| = |D| \Rightarrow RB = RD \Rightarrow B = D) \Rightarrow |A :: B| = |C :: D| \Rightarrow R(A :: B) = R(C :: D) \Rightarrow A :: B = C ::$$

  where the first gained hypothesis is the inductive hypothesis. At this point, given the $\overrightarrow{dt}$ provided in Algorithm 1.1 on page 16, we must apply the double induction strategy even over B and D, and hence we have to prove this other following sub-cases:

  - For $B = []$ we observe that, :

    $$|[A]| = |C :: D| \Leftrightarrow 1 = 1 + |D| \Leftrightarrow 0 = |D| \Leftrightarrow D = []$$

    We could rewrite D as an empty list in our goal, and then achieve the following result:

    $$R[A] = R[C] \Rightarrow A = C$$

    where $R[A] = R[C] \Rightarrow A = C$ by R definition, and hence we could prove the $A = C$ final goal.

  - Even for $B \equiv E :: F \wedge D = []$ we don't use the inductive hypothesis, and hence we have:

    $$|A :: E :: F| = |[C]| \Rightarrow R(A :: E :: F) = R[C] \Rightarrow A :: E :: F = [C]$$

    The goal is proved by absurd since it is clear that it is not possible that:

    $$|A :: E :: F| = 2 + |F| = 1 = |[C]| \Rightarrow 1 + |F| = 0$$

– For $B \equiv E :: F \wedge D \equiv G :: H$, we obtain two inductive hypothesis; we use only:

$$|E :: F| = |G :: H| \Rightarrow R(E :: F) = R(G :: H) \Rightarrow E :: F = G :: H$$

Given that the goal with the non-inductive hypothesis is:

$$|A :: E :: F| = |C :: G :: H| \Rightarrow R(A :: E :: F) = R(C :: G :: H) \Rightarrow A :: E :: F = C :: G :: H$$

we have that $|A :: E :: F| = |C :: G :: H| \Leftrightarrow |E :: F| = |G :: H|$, and hence we could use this in our inductive hypothesis in order to reduce it to:

$$R(E :: F) = R(G :: H) \Rightarrow E :: F = G :: H$$

Since we have by R's definition and previous lemmas that:

$$\begin{aligned} R(A :: E :: F) = R(C :: G :: H) &\Leftrightarrow dt(A, R(E :: F)) = dt(C, R(G :: H)) \\ &\Leftrightarrow A = C \wedge R(E :: F) = R(G :: H) \end{aligned}$$

we can use $R(E :: F) = R(G :: H)$ for reduce the remaining part of the inductive hypothesis into $E :: F = G :: H$ and hence rewrite $A :: E :: F$ into $C :: G :: H$, and then apply the symmetry of the equivalence.

□

Lemma 3.5.

$$\forall l, m. l = m \Rightarrow \overrightarrow{dt}(l) = \overrightarrow{dt}(m)$$

Proof. Trivial. □

Theorem 3.2.

$$\forall l, m. l = m \Leftrightarrow \overrightarrow{dt}(l) = \overrightarrow{dt}(m)$$

Proof. By the two previous lemmas. □

3.5.2 *Proofs for algebraic operations*

Theorem 3.3 (Algebraic Union's Consistency)**.** *Given a set of databases DB where each database have different indices for different data entries, we have that $\bigcup DB$ generates a new database with consistent indices.*

Proof. The proof could be easily carried out from the Theorem 3.2 on the preceding page, where the following corollary could be easily obtained:

$$\forall l, m. l \neq m \Leftrightarrow \overrightarrow{dt}(l) \neq \overrightarrow{dt}(m)$$

Given that the $\bigcup TS$ definition guarantees that, if there is only one occurrence of a given record, this will be mapped as a list with a single element; it also checks that multiple occurrences with the same data value are mapped into a single entry: this guarantees that the generated database will have consistent indices, too. □

Similar considerations could be carried out for projection and Γ.

Theorem 3.4 (Algebraic Join's Consistency)**.** *Given two databases db and db' and two tables $t \in db$ and $t \in db'$ and four indices $\forall i, j, k, h \in \mathbb{N}$ such that $\varphi^{-1}(i), \varphi^{-1}(j) \in t$ and $\varphi^{-1}(k), \varphi^{-1}(h) \in t'$, given that through $db \dot{\bowtie}_\theta db'$ even $t \bowtie_\theta t'$ is computed, if $\theta(i,k)$ and $\theta(j,k)$ then $dt(i,k) \neq dt(j,k)$*

Proof. It is immediate by Lemma 3.1. □

Some other operations, like ε or $Calc$ only extend the data entry with some other values or change their definition like ρ, and so it is trivial to prove that they transform each data table into an index-consistent data table. Since σ_P removes the data entries that don't satisfy P, if the database is index-consistent with all its indices, it will remain so even after removing some indices.

4

A SURVEY ON (HYPER)GRAPH MINING TECHNIQUES

Contents

4.1 GRAPH MINING

Why studying graph mining techniques if this thesis is about an hypergraph mining algebra? [ABBo6] shows that «*while hypergraphs may be an intuitive representation of higher order similarities, it seems (anecdotally at least) that graphs lie at the heart of this problem*». In fact it is there showed that the problems of spectral hypergraph analysis is reducible to the graph spectral analysis via *star expansion* and *clique expansion*, that transform hyperedges in graph edges.

4.1.1 *Graph Clustering*

At this point, we suppose to manage graph data and hence, in order to provide a smooth transition to the hypergraph representation, we introduce the graph data model, that will be used only for a preliminary review.

Definition 4.1.1 (Graph Data Model)**.** *The **graph data model** of a graph $\mathcal{G} = (V, E)$ is described by the following properties and functions:*

- *V is the vertex/data set, that could be extracted through tables' tuples from databases.*
- *E is the set of the edges $E \subseteq V^2$*
- *Each vertex $v \in V$, as an instance of a relation $\Re$ over attributes, has a schema $\mathbb{S}(v) = \Re$.*

- $\forall(u,v) \in E.w(u,v) \neq 0$, *where* w *is the weight function that is strictly liked with the graph structure and to the informations that are shared. If* $w(u,v)$ *we could assume that the link between* a *and* b *is missing.*
- $\mathbb{T}: E \to$ `String` *is a function that retreives the kind of relation that intercurs between two nodes.*
- $\lambda: V \to \mathbb{R}$ *is a vector that maps each graph vertex into its weights.*

Moreover each graph vertex is represented as a single record of a given relation's instance; the set of all the possible graphs is indicated with $\mathbb{G}$.

Definition 4.1.2 (Graph Database Model)**.** *The* ***graph-database model*** *is defined by a list of graphs* $DB = \{ g_i \}_{i \leqslant l}$ *where each* g_i *is a graph (data model), where each graph index is a DFS code as defined in [YH02]. Each database's graph is described by the following functions:*

- $\Lambda: \mathbb{G} \to$ `String` *is the function that returns the graph's associated name or label.*
- $\Delta: \mathbb{G} \to$ `String` *returns the graph's DFS code.*
- $\mathcal{C}: \mathbb{G} \to$ `List<String>` *returns the graph's assigned cluster (or list of clusters).*

4.1.1.1 *Intra-Graph clustering*

With intra-graph clustering we want to detect all the subgraphs of a given graph, which could detect all the similar nodes inasmuch as they are related by vertex-linking edges: this interconnection could be also weighted in order to express the strength of such relationship.

Definition 4.1.3 (Subgraph Isomorphism)**.** *Given two graphs* $G = (V_1, E_1)$ *and* $H = (V_2, E_2)$, H *is a subgraph of* G $(H \leqslant G)$ *[GJ90] iff.*[1]*:*

$$\exists V, E. \exists f: V_2 \to V. V \subseteq V_1 \wedge E \subseteq E_1 \wedge |V| = |V_2| \wedge |E| = |E_2| \wedge \\ (\forall (u,v) \in V_2 \times V_2. (u,v) \in E_2 \Leftrightarrow (f(u), f(v)) \in E)$$

Given $\mathcal{P}_G(G) = \{g | g \leqslant G \wedge g \neq (\varnothing, \varnothing)\}$ *the set of all the possible subgraphs,* $\mathcal{P}(\mathcal{P}_G(G))$ *returns the set of all the possible graph clusters. Hence any* ***intra-graph clustering algorithm*** $\mathcal{A}$ *with an input* $G \in \mathbb{G}$ *returns a set of tuples* $\{ (g_i, \Lambda(g_i), \Delta(g_i), C(g_i)) \}_{i \leqslant l}$, $\forall i \leqslant l. g_i \leqslant G$ *and* $\bigcup_{i \leqslant l} g_i < G$.

Both these problems could be resolved with the Markov Clustering Algorithm over a square matrix M [Donoo], which could be normalized, and then we could alternate the calculation of the matrix power e with the *inflation* operation, which amplifies the contrast between the areas with greater flow and the ones with a lower one by a given parameter r.

1 Note that every book provides different definitions of the same problem. More definitions are provided in [Sam+13; WNK10].

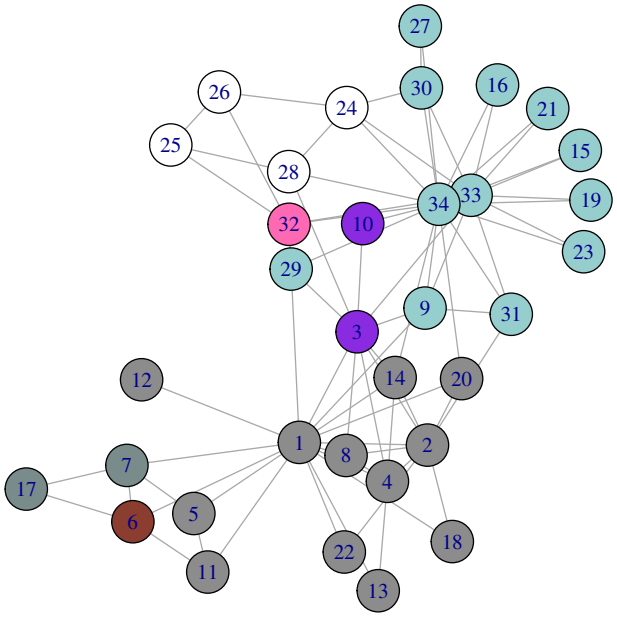


Figure 4.1: Graphic resolution of the [Zac77] problem with the Algorithm C.8 on page 156 using a 6 factor for both inflation and number of iterations. Given a initial uncolored graph, such algorithm identifies the clusters as idetified by the different colours.

```
M ← Normalize[M + I]
do {
  prev ← M
  M ← M^e
  M ← Inflate[M, r]
} while (prev ≉ M)
```

Algorithm 4.1: MCL Algorithm

Exercise 4.1. *Given a graph representation $G = (V, E)$ and an edge weight function w, we could extract from the graph some rules by $\texttt{expr} \equiv E$, $\Re \equiv =$ and $v_{expr} \equiv w$. Our relation table could be defined as $M = \kappa_{E,=,w}(\mathcal{G})$. At this point, we could easily express all the iterative matrix-based algorithms in a mining-loop form.*

MCL's time complexity could be reduced to $O(n)$ with the following heuristics, that remove the less probable walks in sparse matrices:

- **Exact pruning**, where only the largest k escape probability are computed in the Markov matrix associated to the weighted graph.
- **Theresold pruning** where, for each stochastic column c of a matrix and the threshold $\theta \propto ctr(c)$ where ctr is the *mass center* of the vector, only the values greater that θ are kept.

It was also proved in [BH09] that this algorithm with the heuristics introduced in [Dono0] in order to reduce the computational complexity, outperformed on the other clustering algorithms over some undirected protein networks, by using a sofisticated systematical comparation. Event if this version doesn't handle directed graph, [YKSP13] proposed an extension of Markov Clustering that could also handle directed graphs.

4.1.1.2 *Graph Descriptors*

In order to define the intra-database algorithms, we must now define some functions and show some freqeunt subgraphs algorithms in order to be able to characterize the database's graphs G_i via each mined subgraphs g_j, and hence establish the correlation $s(G_i, g_j)$ via a support function s.

Definition 4.1.4 (Support Function)**.** *A **support function** $\vartheta : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ [YH02]returns the likeness that the first graph contains the second as a subgraph:*

$$\vartheta(g, G) = \begin{cases} 1 & g \leqslant G \\ 0 & oth. \end{cases}$$

*The **frequency support function** denotes how frequent is a subgraph g in a graph database DB:*

$$GS(g, DB) = \sum_{G \in DB} \vartheta(g, G)$$

Definition 4.1.5. *Given a graph database* $DB = \{ G_i \}_{i \leqslant k} \subseteq \mathcal{P}(\mathbb{G})$ *the* ***most frequent subgraphs*** $\{ g_i \}_{i \leqslant l}$ *are those which appear more frequently in* DB*. Given a minimum support threshold* θ*, this algorithm could be generalized as a* $MFSG^{\mathcal{G}}_{\theta}$ *where* $MFSG^{\mathcal{G}}_{\theta} : \mathcal{P}(\mathbb{G}) \to \mathbb{D}_{\mathcal{P}(\mathbb{G})}$.

Example 8 provided an algorithm for mining the most frequent itemsets from transactions: we choose to show here only gSpan [YH02; CH06a; Sam+13], as far as other solutions require substantial changes in order to extend the computation in order to include directed graphs [W¨05]. This algorithm uses the depth-first approach in order to produce a minimum DFS code, which is composed from the vertices' and edges' values; moreover, these codes could be sorted with the *DFS lexicographic order* which has the aim to enumerate the frequent rightmost extensions $g \diamond_r e$ of a graph code g with an edge e. The algorithm hereby returns the S set of the frequent substructures in the graph database DB.

> By the way, [YH02] doesn't explicitly define the support function, that is still hard to compute, since the subgraph isomorphism is still considered as an NP-Complete problem [GJ90]. [Fan+11] shows how it is possible to reformulate such subgraph isomorphism as "graph pattern queries" with a $O(|V|^3)$ of the queryied graph, with both `SplitMatch` and `JoinMatch`. Moreover, it is also interesting to observe how a given graph pattern query could be translated into a HML formula[Ace+07], in order to check if it exists another graph that could be satisfy that formula, and hence to test if it could simulate the subgraph-query.

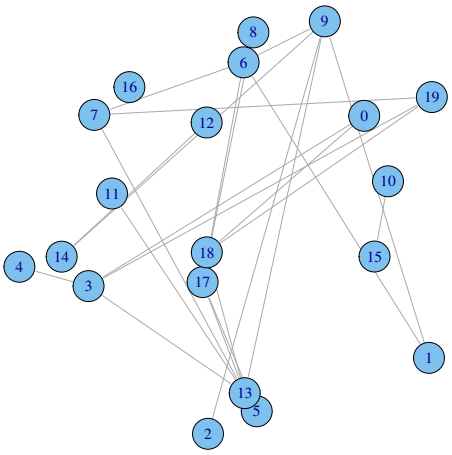
```
gSpan(DB, minsupp, Solution) {
  Ξ ←sort(FrequentEdges(DB,minsupp))
  Solution ← Ξ;
  NStack ← S
  while (g ← NStack.pop()) {
    if g ≠ minDfsCode(g) continue;
    Solution ← g
    ∀e ∈ Ξ. if (e ◇re g)⇒ { // if e ◇re g is a rightmost expansion of g by e
      if GS(e ◇re g, DB) ⩾ minsupp
        NStack.push(e ◇re g)
    }
  }
}
```

Algorithm 4.2: gSpan

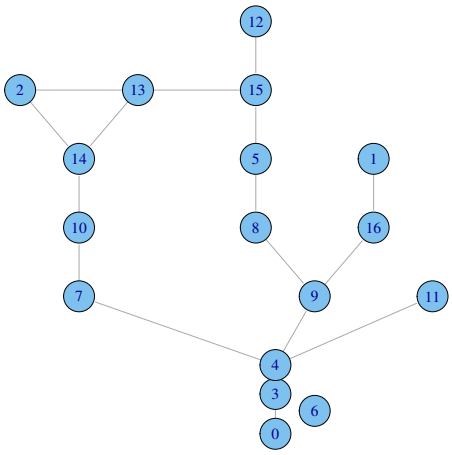
Exercise 4.2. *Given the algorithm implementation in R given in [Sam+13], Figure 4.2 provides an example of the execution of the algorithm over a minimal* $DB = \{ G_1, G_2 \}$ *graph database.*

Problem 4.1. *We could try to express the latter algorithm with a while-loop, since recursion could be easily expressed as an indefinite loop and the rules could be expressed over* $\Re \equiv \diamond_r$.

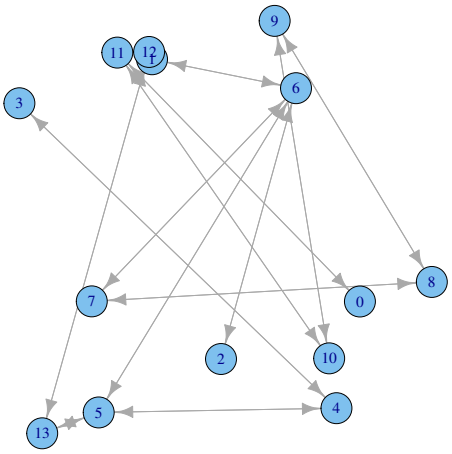
Now we'll introduce the **average distance** which is the base of geometrical descriptors, inasmuch as this measure could be relevant to identify a "geometric signature" of a specific subgraph that is translational and rotation invariant:



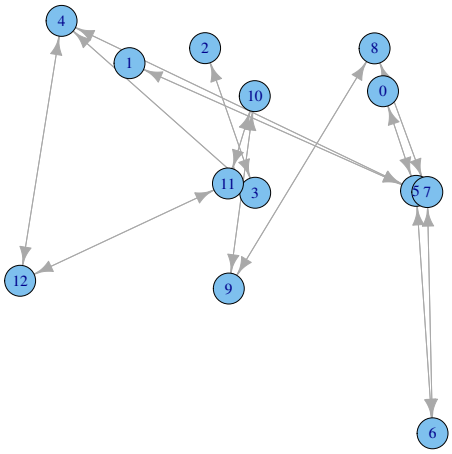
(a) $G_1 \in DB$



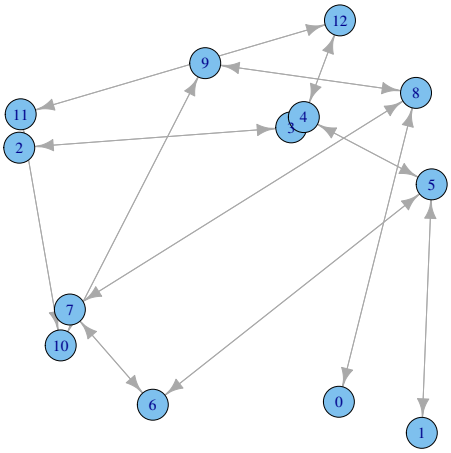
(b) $G_2 \in DB$



(c) $g_2 \in Res$



(d) $g_2 \in Res$



(e) $g_3 \in Res$

Figure 4.2: Given a graph database $DB = \{ G_1, G_2 \}$, the result of the gSpan algorithm with support 0.95 provided in [Sam+13] returns 335 graphs which g_1, g_2, g_3 are the first three.

Definition 4.1.6. *Given a grpah $G = (V, E)$, the matrix D of the **geometrical vertices' distances** (gvd) is:*

$$D_G = \{ e(pos(u), pos(v)) \}_{u,v \in V}$$

*where $pos : V \to \mathbb{R}^n$ gives the space location of the vertex and e is the euclidean distance. The **average distance** $avg(D)$ is defined over the average over the values obtained in D.*

Definition 4.1.7. *A graph $g = (\mathcal{V}, \mathcal{E})$ is an **embedding** of g_j in g_i iif. $V_i \equiv \mathcal{V} \subseteq V_j \wedge E_i \equiv \mathcal{E} \subseteq E_j$. The set of the embeddings is consequently defined as follows:*

$$Embed(g_i, g_j) = \{ (\mathcal{V}, \mathcal{E}) \mid V_i \equiv \mathcal{V} \subseteq V_j \wedge E_i \equiv \mathcal{E} \subseteq E_j \}$$

Definition 4.1.8. *A **geometric graph** $c_i = (g_i, avg(D[V_{g_i}; V_{g_i}]))$ is defined by a (topological) graph g_i and its average distance. A geometric graph $C = (G, D)$ contains a geometric subgraph $c = (g, d)$ ($c \preceq C$) iif.:*

$$g \subseteq G \wedge \exists e \in Embed(g, G). \min(D_G) \leqslant avg(D_e) \leqslant \max(D_G)$$

In order to mine the most frequent geometric subgraphs, we could think to adapt any previously given algorithm for mining topological descriptors (as gSpan) as depicted in [Des+05], in a way that also improves the classification accuracy of the algorithm:

- Collect all the frequent geometrical subgraphs $\{ c_i \}_{i \leqslant l}$ in DB with a variant of the most frequent subgraphs algorithm depicted above, wich instead returns the most frequent topological subgraphs $\{ g_i \}_{i \leqslant l}$.
- For each $G_i \in DB$ (and hereby $C_j \in DB$) and c_i where $c_i \preceq C_j$, set $v_{c_i, C_j} = \{ avg(g) \}$
- Given a bound r to the interatomic distance, we could perform a one-dimensional clustering cl over the values collected in v_{c_i, C_j} ($cl \in \mathcal{P}(v_{c_i, C_j})$) where $\forall \nu \in cl. |\max(\nu) - \min(\nu)| > r$, in order to remove the possible outliers.
- For each $\nu \in cl$ we could obtain the final $c_\nu = (c_i, avg(\nu))$ geometrical frequent subgraphs.

4.1.1.3 *Hierarchical Intra-Database clustering*

The hierarchical clustering over topological graph is very simple, as we could directly use the support function in order to classify the graphs $\{ G_i \}_{i \leqslant k} = DB$ against the most frequent subgraphs $\{ g_i \}_{i \leqslant l}$ previously mined, as introduced before:

Definition 4.1.9. *Given a set of topological subgraphs $\{ g_i \}_{i \leqslant l}$ obtained from a graph database $\{ G_i \}_{i \leqslant k} = DB$, for each $G_i \in DB$ we want to define a **(topological) description vector** v_i as follows:*

$$\forall j \leqslant l. v_i[j] = \begin{cases} supp(G_i, g_j) & supp(G_i, g_j) > minSupp \\ 0 & oth. \end{cases}$$

These database graph's vectors are produced by the following function:

$$TDV(DB, \{ g_i \}_{i \leqslant l}) = \{v_i \mid G_i \in DB \wedge Dom(v_i) = \{ g_i \}_{i \leqslant l}\}$$

Definition 4.1.10. *Given a generic distance function δ between two elements, we could define the $\mathcal{K}$ function, which creates the similarity matrix as follows:*

$$\mathcal{K}_\delta(V) = \{ \delta(x, y) \}_{x,y \in V}$$

At this point, as suggested in [WNK10], we could use the **Tanimoto coefficient** in order to define the distance between two vectors X and Y:

$$T(X, Y) = \frac{\sum_{i=1}^{M} X[i] \cdot Y[i]}{\sum_{i=1}^{M} X[i]^2 + Y[i]^2 - X[i] \cdot Y[i]}$$

Define a distance matrix between the mined vectors:

$$DV_T^\theta(DB) = \mathcal{K}_T(TDV(DB, MFSG_\theta(DB)))$$

and then use the UPGMA algorithm (firstly described in [SM58]) to define the associated dendogram that provides the hierarchical clustering algorithm. UPGMA returns a $T(DB)$ tree (which is induced by the representation of the dendogram) where the leaves represent all the graphs in DB:

```
while (nrows(K_T) > 1) {
  Dendogram[A,B] ← (1/2) min_{A,B} K_T[A,B]
  ∀i ∉ { A, B }
    M[i, { A, B }] ← M[{ A, B }, i] ← (M[A,i]+M[i,B])/2
  M[{ A, B }, { A, B }] ← 0
  delete in M rows and columns whose index
    is in { A, B }
}
return Dendogram;
```

Algorithm 4.3: UPGMA Algorithm

Exercise 4.3. *Given the Laurasiatherian RNA sequence data provided in [LY10], we could use the UPGMA implementation given in the same package in order to obtain the result showed in Figure 4.3.*

Exercise 4.4. *As observed before, even this algorithm could be expressed as a mining loop because it is based on a matrix iteration and manipulation.*

We could hereby cluster the DB graphs via geometrical subgraphs the same way as for topological ones. In this case, an hierarchical clustering algorithm $\mathcal{A}$ could be generalized with the following operator:

$$HC^{\mathcal{A}} \colon M(|DB|, \mathbb{R}) \to T(DB)$$

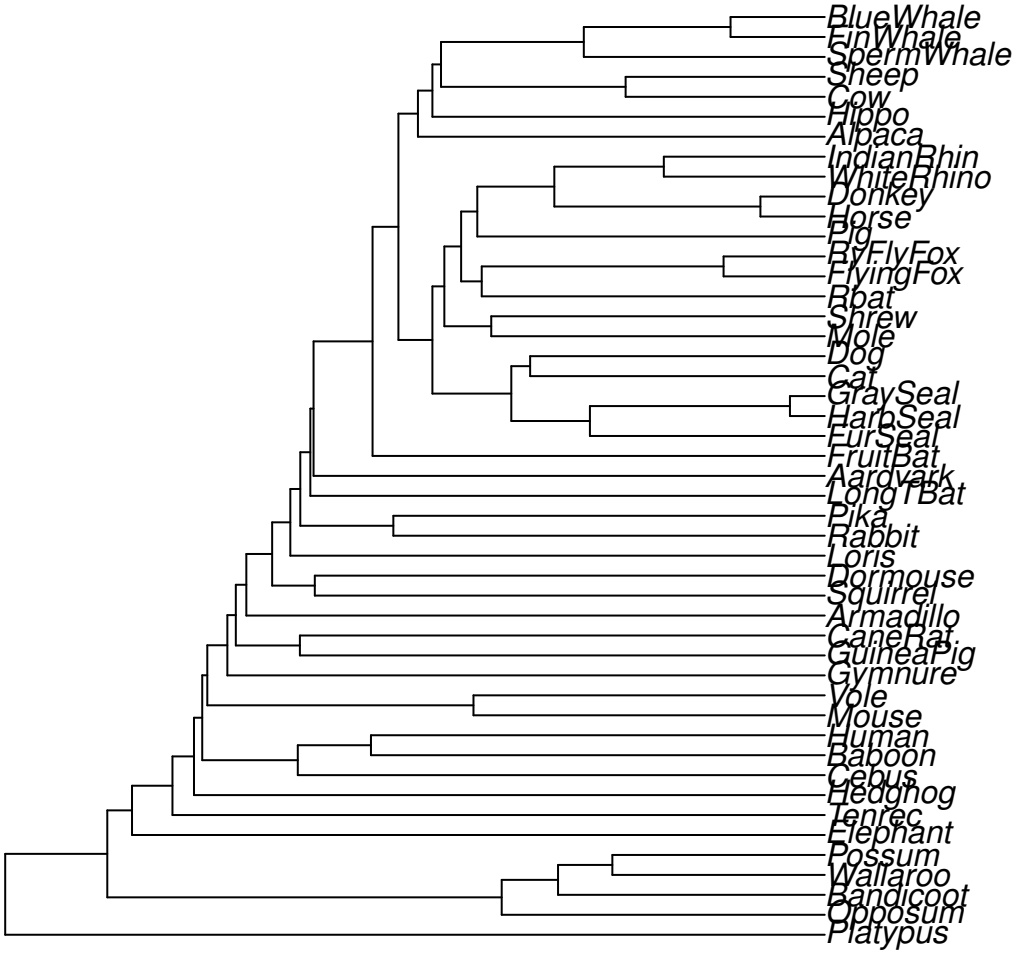


Figure 4.3: UPGMA algorithm example over RNA sequence classification.

where $M(n, K)$ is the set of the K-valued matrices with $n \times n$ size. Moreover, we could also think to cut the tree from a given distance from the root with a $Prune(T, h)$ operator, in order to remove the most infrequent clusters and to obtain some cluster $C \subseteq \mathcal{P}(DB)$ where each cluster $c \in C$ is at least h-far from the root.

4.1.2 *Association Analysis*

In order to solve this problem, [TTM11] exstimate the support witout extimating the *confidence* of the relation between subcomponents, which measures the reliability of the proposed interaction; by the way, they used the likelihood-ratio test with logistic regression as a replacement [Sha13]. In particular, suppose to have already performed some hierarchical clustering over $\mathcal{G}$ and $\mathcal{T}$ in order to perform the interaction analysis only over clusters achieved through graph descriptors for both drugs ($\{ g_i \}_{i \leqslant m}$) and protein ($\{ t_i \}_{i \leqslant n}$). We could obtain some relevant information from this particular problem as follows:

Definition 4.1.11. *The **support function** which determines the how a rule is common inside a graph, is defined as follows:*

$$supp(g \leftrightarrow t) = \frac{\sigma(g, t)}{|E|}$$

where $\sigma(g, t) = |\left\{ (\mathcal{G}_i, \mathcal{T}_j) \;\middle|\; g \subseteq \mathcal{G}_i, t \subseteq \mathcal{T}_j, (\mathcal{G}_i, \mathcal{T}_j) \in E \right\}|$. *The **confidence function** is instead defined as:*

$$conf(g \leftrightarrow t) = \frac{\sigma(g, t)}{\sigma(g)}$$

where $\sigma(g) = |\{ \mathcal{G}_i \mid g \subseteq \mathcal{G}_i, \mathcal{G}_i \in \mathcal{G} \}|$.

A generic association algorithm for binary relations with minimum support θ and minimum confidence ρ could be defined as follows:

$$AA^2_{\theta,\rho}(E) = \{ e \mid e \in E, supp(e) > \theta, conf(e) > \rho \}$$

We could consequently obtain another graph, which shows the relations between the substructures of both chemical molecules and protein as follows:

$\{ g_i \}_{i \leqslant M} \leftarrow MFSG_\theta(\mathcal{G})$
$\{ t_i \}_{i \leqslant N} \leftarrow MFSG_\theta(\mathcal{T})$
$\{ u_i \}_{i \leqslant M} \leftarrow TDV(\mathcal{G}, \{ g_i \}_{i \leqslant M})$
$\{ v_i \}_{i \leqslant N} \leftarrow TDV(\mathcal{T}, \{ t_i \}_{i \leqslant N})$
$\{ G_i \}_{i \leqslant m} \leftarrow leaves(Prune(HC^{\mathcal{A}}(DV^\theta_T(\mathcal{G})), h_1)), m < M$
$\{ T_i \}_{i \leqslant n} \leftarrow leaves(Prune(HC^{\mathcal{A}}(DV^\theta_T(\mathcal{T})), h_1)), n < N$
$E_f \leftarrow \left\{ (g_k, t_l) \;\middle|\; (G_i, T_j) \in E, u_i[k] \neq 0, v_j[l] \neq 0 \right\}$
$G_f \leftarrow (\{ g_i \}_{i \leqslant M} \cup \{ t_i \}_{i \leqslant N}, AA^2_{\theta,\rho}(E_f))$

Algorithm 4.4: Problem 5 solution

A more trivial DB graphs' classification could be induced by the their TDVs in order to hierarchically correlate the graphs in a lattice as follows [CHo6a]:

$$v_i \sqsubseteq v_j \Leftrightarrow \forall k \leqslant l . v_i[k] \leqslant v_j[k]$$

where l is the common vector size.

4.1.3 *Graph Classification Rules*

Since the TDV provides a method to map each graph $G_i \in DB$ into a description vector v_i, the v_i extended with the belonging class mined through the hierarchical clustering could be seen as a relation $\Re(g_1, \ldots, g_l, \mathcal{C})$, where $\{ g_i \}_{i \leqslant l}$ are the mined most frequent subgraph, and $\mathcal{C}$ expresses to which hierarchical cluster $c \in C$ each graph G_i belongs to. Given this tabular form, we could use Naïve Bayes over continue attributes (see Subsection 3.4.1 on page 44) in order to predict from the previously mined frequent subgraphs $\{ g_i \}_{i \in \mathbb{N}}$ the classification of a chemical compound G_{test}. For instance, we could define a bayesian classifier as follows:

$$Class(G_{test}, \{ g_i \}_{i \leqslant l}) = \max_{c \in \mathcal{C}(G)} \left\{ \prod_{i \leqslant l} P(g_i = v_{test}[i] | C = c) \right\}$$

$$\mathcal{C}(G_{test}) = Class(G_{test}, \{ g_i \}_{i \leqslant l})$$

where $\{ g_i \}_{i \leqslant l}$, as already said, was previously mined through most frequent subgraphs and v_{test} is the description vector of G_{test}.

4.2 HYPERGRAPH MINING

Well, but didn't you say that all mining hypergraph problems were reducible to graph problems? Yes, and we're always of the same opinion. By the way, it is clear that the Zoo database used in [Vazo9] (where each hyperedge-row represents a different animal and each column represent an attribute that characterises and distinguishes it from the others) has a natural hypergraph representation, as it doesn't provide any obvious correlational relationship over the data. However it is clear that such correlations could be mined by manipulating such hypergraph.

4.2.1 *Vertex Clustering via Regularized Laplacian*

Given an hypergraph data model $\mathcal{H}$, we could try to point out the correlation between the data inside each hyperedge («*the relationship among different labels can then be captured by their interactions with the data points*» [SJYo8]), by computing $A(\mathcal{H})A^T(\mathcal{H})$. By the way [Sam+13] points out how this similarity matrix's weights (in fact such «*transition probability between the nodes associated with two [hyperedges-entities] captures their similarity* » [SJYo8]) are strictly

related to the weighted degree of each vertex, and hence we have to normalize such information:

Definition 4.2.1 (Target Normalized Laplacian). *A **target normalized laplacian** of a hypergraph [Sam+13] $\mathcal{H} = (\mathcal{V}, \mathcal{E}_D, \mathcal{E}_E)$ is defined as follows:*

$$\forall v_i, v_j \in \mathcal{V}. L_{i,j}(A(\mathcal{H})A^T(\mathcal{H})) = \begin{cases} 1 & v_i = v_j \wedge \deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{\deg(v_i)\deg(v_j)}} & v_i \neq v_j \wedge A(\mathcal{H})A^T(\mathcal{H})[i,j] \neq 0 \\ 0 & oth. \end{cases}$$

where the degree could be defined as:

$$deg(v_i) = \sum_{v_j \in \mathcal{V}} A(\mathcal{H})A^T(\mathcal{H})[i,j]$$

The following code defines the final matrix that could be used to infer the similarity relations over the data:

```
VertexClust <- function (BTB) {
  BTB <- as.matrix(unname(BTB));
  BTB <- BTB %*% t(BTB);
  rows <- dim(BTB)[1];
  cols <- dim(BTB)[2];

  D <- diag(rowSums(BTB));
  I <- diag(1,rows,rows);

  Lap <- matrix(0,rows,rows);
  for (i in 1:rows) {
    for (j in 1:rows) {
      Lap[i,j] <- (D[i,j] - BTB[i,j])/sqrt(D[i,i]*D[j,j]);
    }
  }

  return(Lap);
}
```

Algorithm 4.5: Regularized Laplacian Matrix

Since graph and hypergraph laplacians are usually defined in order to simulate random walk models over them [SJY08], we decided to use the MCL over such matrix, instead of applying other algorithms that have been proposed as [ABB06; SJY08]. Using the Iris data set [Fis36] with Markov Clustering over graphs adjacency matrices, we made out to separate all the three flowers clusters as follows:

```
# Setosa+Virginica Clusters
SV <- iris[c("Petal.Width","Petal.Length")];
```

```
Lap <- VertexClust(SV);
res = collect.mcl.clusters(mcl(Lap,2,600));


# The second cluster is made of Virginicas
SV <- iris[c("Petal.Width","Petal.Length")];
Lap <- VertexClust(SV);
res = collect.mcl.clusters(mcl(Lap,2,8));
```

Algorithm 4.6: Iris Database Example

5

HYPERGRAPH OPERATORS

Contents

In this chapter, We define the operators that will be used in the Data Mining Algebra: these are useful since they allow to define some data mining operators as a composition of different algebra operators over hypergraphs.

In the light of new knowledge provided by the Threefolded Data Mining, we'll try to refine our first basic data model over hypergraphs, in order to suggest a structure that makes possible to differentiate data from mined relations.

5.1 HYPERGRAPH DEFINITION OVER THE THREE WORLDS

At this point, we want to distinguish two hypergraph worlds, one of the correlated data and the other of the mined properties. An example of the first kind of an hypergraph was already depicted on Figure 2.2 on page 25, where we could define as data the world of both binary relations and pure data entities. We could provide a preliminary definition of D/I-Hypergraph in order to map entities and correlational relations provided by the data.

Definition 5.1.1 (D/I-Hypergraph)**.** *An hypergraph data model (Definition 2.1.3 on page 23) where the property (HG2) on page 28 holds, is called a* ***D/I-Hypergraph*** *D/I-Hypergraph.*

For the D/I-Hypergraph we could use the Algorithm 2.3 on page 29 as a visit procedure. By the way, this data-hyperedge $\mathcal{E}_D$ definition doesn't take into account all the considerations that could be carried out from lemmas in Section 2.2 on page 31, and hence we would like to specialize the latter definition for:

- **Providing an algebraic representation of non-correlational data, in order to easily express such algebraic operators.**

  Since we've seen in Subsection 2.2.3 on page 33 the data relations $\mathcal{E}_D$ could be represented directly as databases and that the transformation of a database to a TAM could have a linear computational cost in the size of the database, we could map all the $\mathcal{E}_D$ into a database.

- **Producing (when needed) relation modifications (hyperedges in $\mathcal{E}_E$) only as a side effect of data manipulations.**

  In order to make that possible, we introduce a tensor-like notation for binary relations and use the indexing structure presented in the previous chapters, in order to easily keep track of data modifications.

- **We would like to abstract binary $\mathcal{E}_E$ to more general ones.**

  For the moment, we won't abstract the binary relations into "more general ones", and we suppose (in order to ease our definitions) to keep binary relations among the data elements.

5.1.1 *Tensors for binary relations*

[NTK11] suggests a "*tensor model for relational data*", where each binary relation between two entities i and j in a relation $\Re$ could be expressed as $(i, j) \in \Re$, where $\Re$ should be interpreted as a binary matrix. Hence, we could define our tensor for binary relations as follows:

Definition 5.1.2 (D/I-Hypergraph tensor)**.** *A **D/I-Hypergraph tensor** representation of the $\mathcal{E}_E$ hyperedges of any D/I-Hypergraph $\mathcal{H}$ is indicated as a $TDI(\mathcal{H})$ of size*[1] *$|\mathcal{E}_D| \times |\mathcal{E}_D| \times |\mathbb{T}(\mathcal{H})|$ where $\forall i, j \in Idx.\forall k \in \mathbb{T}(\mathcal{H}).TDI[i, j, k] \neq 0 \wedge \varphi^{-1}(i), \varphi^{-1}(j) \in \mathcal{E}_D$ iff. there is a relation $\Re_k$ where $\varphi^{-1}(i)\ \Re_k\ \varphi^{-1}(j)$. In other words:*

$$\forall i, j, k.TDI[i, j, k] \neq 0 \Leftrightarrow \exists e \in \mathcal{E}_E.\mathbb{T}(e) = k \wedge \{\varphi^{-1}(i)\} = prev(e) \wedge \{\varphi^{-1}(j)\} = next(e)$$

From now on we'll use $TDI[i, j, k] \triangleq TDI[\varphi(i), \varphi(j), k]$ and $TDI_k[i, j] \triangleq TDI[i, j, k]$ as shorthands.

As a first step, we'll only analyze tensors for binary relations, even if we know that it is also possible to define tensors for n-ary relations via rectangular matrices.

Example

1 $\mathbb{T}(\mathcal{H})$ is used as a shorthand for $\{\ \mathbb{T}(e) \mid e \in \mathcal{E}_E\ \}$.

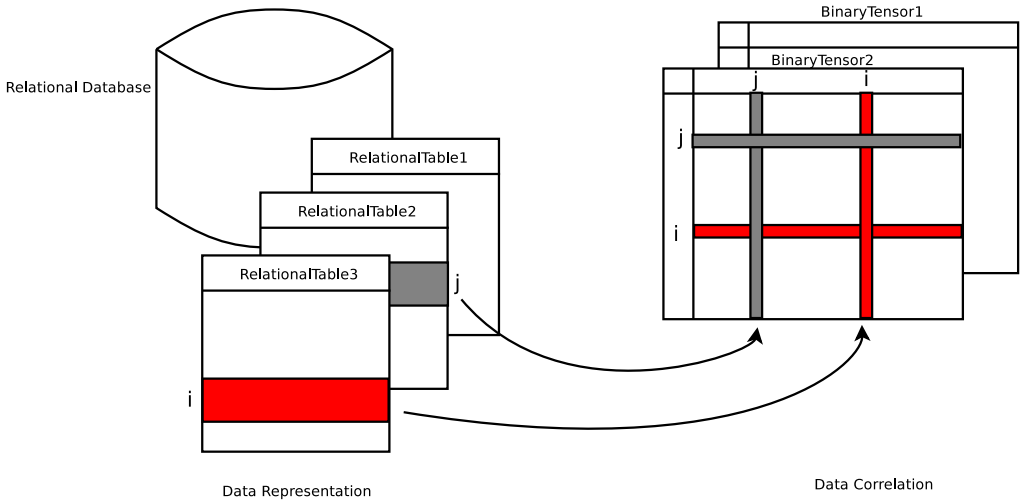


Figure 5.1: *This picture clarifies the DHImpl representation: each Relational Database table entry $\varphi^{-1}(i)$ has an index i, which is also be used in the Tensor representation in order to identify that particular instance..*

11. *Suppose to have a social network hypergraph where the vertices types are posts, users and topics, where we have a friendship relation among users and another relation that links each user to a topic of interest (e.g. each Diaspora user could describe her-/him-self by using 5 hashtags, and hence hashtags are used as a peculiar instance of the topic type). We could suppose to treat the "post-belongs-to" relation as data driven, and hence we won't use that relation in the tensor representation.*

Suppose that now we have to detect the most popular user by selecting the nodes with the greatest degree in the cluster mined with MCL: if we want to consider the influence only by the following-"followee"' relation, we would like to apply the MCL only over this layer without involving the other data relations.

We could think to redefine D/I-Hypergaphs in order to map data hyperedges into databases and binary relations into tensors as follows:

Definition 5.1.3 (D/I-Hypergraph implementation)**.** *A **D/I-Hypergraph implementation** (DHImp) could be seen as a couple (db, T) where db is a Database for Uncertain Data and T is a D/I-Hypergraph tensor.*

Example

12. *Figure 5.1 explains how Relational Databases and Tensors are linked together in $DHImp$-s: given that the database structure is indexed, we could use such indices to point out which entries inside the tensor are related together. While different data tables express different types of entities, different tensors express different kind of relations among the data. This also implies that any change to the database structure should be reflected also on the Tensor representation. The other way around update is not necessary because we could have data that is not related to other entities but that still appears in our dataset.*

5.1.2 *D/I-Hypergraphs algebraic operations - Data operations*

We want to define algebraic operations (inspired by the relational ones) over hypergraphs. At first, we shall discuss how to manipulate $\mathcal{E}_D$ as relational operators, and how these operations could reflect the way $\mathcal{E}_E$ are manipulated. In order to achieve this goal, we have to define some preliminary functions or relations, in order to better define the algebraic operations.

Definition 5.1.4 (Binary $\mathcal{E}_E$ hyperedge constructor)**.** *An **hyperedge constructor** $d_T(i,j)$ takes two data hyperedges i and j, a D/I-Hypergraph tensor T and a type $k \in \mathcal{T}$ and then recreates the original binary $\mathcal{E}_E$ hyperedge as follows:*

$$d_T(i,j,k) = \langle S(j) \cup S(i), k, U(j) :: U(i), T[i,j,k] \rangle$$

Definition 5.1.5 (Elicitation predicate)**.** *The **elicitation predicate** $E \colon \mathcal{E}_D \to \mathcal{E}_D \to \mathcal{T} \to TDI \to Prop \to Prop$ is true iff. it exists an hyperedge with a desired property.*

$$E[i,j,k,T,P] \Leftrightarrow T[i,j,k] \neq 0 \wedge P(d_T(i,j,k))$$

Definition 5.1.6 (Data Selection)**.** *The **selection** operation over a DHImp $h = (db, T)$ is defined as:*

$$\sigma_P^D(h) = (\dot{\sigma}_P(db), \tilde{\sigma}_P(T))$$

where the $\tilde{\sigma}_P$ selection function for the tensor is defined as:

$$\forall i,j,k.(\tilde{\sigma}_P(T))[i,j,k] = \begin{cases} T[i,j,k] & P(i) \wedge P(j) \wedge E[i,j,k,T,P] \\ 0 & oth. \end{cases}$$

In order to make the explanation clearer, I will only treat examples where a single relational table and tensor layer are showed.

Example

13. *Given a DHImp h defined with a database $db = \{t_1\}$ where t_1 is defined as follows:*

Num1	*Num2*	*Num3*	*w*	φ
A	*2*	*4*	*0.1*	*1*
B	*8*	*10*	*0.2*	*2*
C	*15*	*20*	*0.3*	*3*
A	*28*	*3*	*0.4*	*4*
C	*1*	*3*	*0.5*	*5*

and a tensor T with only one layer k:

	1	*2*	*3*	*4*	*5*
1	*0.1*	*0*	*0.3*	*0*	*0*
2	*0*	*0.2*	*0*	*0.5*	*0.1*
3	*0*	*0*	*0.5*	*0.6*	*0*
4	*0.1*	*0*	*0*	*0.2*	*0*
5	*0*	*0.3*	*0*	*0*	*0.5*

the selection $\sigma^{D}_{h.DB.T.Num1="A"}(h)$ provides the following result:

Num1	*Num2*	*Num3*	*w*	φ
A	*2*	*4*	*0.1*	*1*
A	*28*	*3*	*0.4*	*4*

	1	*4*
1	*0.1*	*0*
4	*0.1*	*0.2*

Where indices come useful

At this stage we have to define the join operation. Since that $\forall t \in db.\forall t' \in db'.t \bowtie_\theta t'$ produces entity indices which are the result of the dovetailing function, $\forall r \in t \bowtie_\theta t'.dt^{-1}(\varphi(r))$

provides the two original indices $\langle i, j \rangle$ such that $\varphi^{-1}(i) \in t$ and $\varphi^{-1}(j) \in t'$ by definition. Moreover all the resulting structure has binary indices, and hence:

$$(T \bowtie_\theta T')[i,j,k] \neq 0 \Leftrightarrow \exists \Re_k . \exists a,b,c,d . dt^{-1}(i) = \langle a,b \rangle \wedge dt^{-1}(j) = \langle c,d \rangle \wedge (\varphi^{-1}(a) \Re_k \varphi^{-1}(c) \vee \varphi^{-1}(a) \Re_k \varphi^{-1}(d) \vee \varphi^{-1}(b) \Re_k \varphi^{-1}(c) \vee \varphi^{-1}(b) \Re_k \varphi^{-1}(d))$$

That means that two data-joined relations are linked together by a k relation iff. at least one of the first previous relations were linked to one on the second. In this case the reverse dovetailing function simplifies the definition of $(T \bowtie_\theta T')[i,j,k]$.

Definition 5.1.7 (Data θ-Join)**.** *The θ-**join** operation over two DHImp $h = (db, T)$ and $h' = (db', T')$ is defined as follows:*

$$h \bowtie_\theta^D h' = (db \dot{\bowtie}_\theta db', T \tilde{\bowtie}_\theta T')$$

where the $\tilde{\bowtie}_\theta$ function for the two tensors is defined as:

$$\forall i,j,k . (T \tilde{\bowtie}_\theta T')[i,j,k] = \text{avg} \left\{ i \;\middle|\; i = \text{avg}\{T_i[a,b,k] | a \in dt^{-1}(i) \wedge b \in dt^{-1}(j)\} 0 \wedge T_i \in \{T, T'\} \right\}$$

Example

14. *The join $DHImp_1 \bowtie^D_{DHImpl_1.DB.t_1.Num1 = DHImpl_1.DB.t_1.Num2} DHImpl_2$ where the DHImpl $DHImp_1 = (\{t_1\}, T^1)$ and $DHImp_2 = (\{t_2\}, T^2)$ are defined with the following data tables (t_1 and t_2 respectively):*

Num1	*Num2*	*Num3*	*w*	φ
A	*2*	*4*	*0.1*	*1*
B	*8*	*10*	*0.2*	*2*
C	*15*	*20*	*0.3*	*3*
A	*28*	*3*	*0.4*	*4*
C	*1*	*3*	*0.5*	*5*

Num1	*Num2*	*w*	φ
A	*7*	*0.2*	*6*
B	*5*	*0.3*	*7*

and the following tensors (T^1 and T^2 respectively):

	1	2	3	4	5
1	0.1	0	0.3	0	0
2	0	0.2	0	0.5	0.1
3	0	0	0.5	0.6	0
4	0.1	0	0	0.2	0
5	0	0.3	0	0	0.5

	6	7
6	0	1
7	1	0

gives us the following result:

t_1.Num1	t_1.Num2	t_1.Num3	t_2.Num1	t_2.Num2	w	φ
A	2	4	A	7	0.2	34
B	8	10	B	5	0.6	52
A	28	3	A	7	0.8	61

	34	52	61
34	0.0125	0.15	0
52	0.125	0.025	0.1875
61	0.0125	0.125	0.025

This definition could be generalized in order to express the union of the weights over the data:

Definition 5.1.8 (Data Union). *The **union** operation over a set of DHImp, $H = \{ (db_i, T_i) \}_{i \leqslant k}$ is defined as:*

$$\bigcup^{D} H = (\dot{\bigcup_i} db_i, \tilde{\bigcup_i} T_i)$$

where the $\tilde{\bigcup}$ union function for the set of tensors is defined as:

$$\forall i,j,k.(\tilde{\bigcup} T)[i,j,k] = \text{avg}\left\{ i \;\middle|\; i = \text{avg}\{T_i[a,b,k] | T_i[a,b,k] \neq 0, a \in \overrightarrow{dt^{-1}}(i), b \in \overrightarrow{dt^{-1}}(j)\}, i \neq 0, T_i \in T \right\}$$

Example

15. *The union between the following* $DHImp_1 = (\{t_1\}, T^1)$ *and* $DHImp_2 = (\{t_2\}, T^2)$*, where* t_1 *and* t_2 *are respectively defined as follows:*

Num1	*Num2*	*Num3*	*w*	φ
A	*2*	*4*	*0.1*	*1*
B	*8*	*10*	*0.2*	*2*
C	*15*	*20*	*0.3*	*3*
A	*28*	*3*	*0.4*	*4*
C	*1*	*3*	*0.5*	*5*

Num1	*Num2*	*Num3*	*w*	φ
A	*7*	*11*	*0.2*	*6*
B	*2*	*4*	*0.1*	*7*

and tensors T^1 T^2 *have a single layer and are defined as depicted below:*

	1	*2*	*3*	*4*	*5*
1	*0.1*	*0*	*0.3*	*0*	*0*
2	*0*	*0.2*	*0*	*0.5*	*0.1*
3	*0*	*0*	*0.5*	*0.6*	*0*
4	*0.1*	*0*	*0*	*0.2*	*0*
5	*0*	*0.3*	*0*	*0*	*0.5*

	6	*7*
6	*0*	*1*
7	*1*	*0*

gives us the following result:

Num1	*Num2*	*Num3*	*w*	φ
A	*7*	*11*	*0.2*	*34*
A	*28*	*3*	*0.4*	*19*
B	*8*	*10*	*0.2*	*8*
C	*1*	*3*	*0.5*	*26*
C	*15*	*20*	*0.3*	*13*
A	*2*	*4*	*0.1*	*1078*

	34	*19*	*8*	*26*	*13*	*1078*
34	*0*	*0*	*0*	*0*	*0*	*0.25*
19	*0*	*0.1*	*0*	*0*	*0*	*0.025*
8	*0*	*0.25*	*0.1*	*0.05*	*0*	*0*
26	*0*	*0*	*0.15*	*0.25*	*0*	*0*
13	*0*	*0.3*	*0*	*0*	*0.25*	*0*
1078	*0.25*	*0*	*0*	*0*	*0.075*	*0.0125*

Some other operations only involve a single database, and hence could be defined in a more simple way. In this case we could use an unique function for updating the tensors

Definition 5.1.9 (Update Tensors for Data Unary Operations). *Given a DHImp $h = (db, T)$ and an unary operation $\triangleright$ over a database, if we perform $\dot{\triangleright}(db)$ we could update T with the $Update(T)$ function as follows:*

$$Update(T)[i,j,k] = \text{avg}\left\{ T[a,b,k] \;\middle|\; a \in \overrightarrow{dt^{-1}}(i), b \in \overrightarrow{dt^{-1}}(j) \right\}$$

Definition 5.1.10 (Data Projection). *The **projection** operation over a DHImp $h = (db, T)$ is defined as follows:*

$$\pi^D_{\mathcal{L}}(h) = (\dot{\pi}_{\mathcal{L}}(db), Update(T))$$

Example

16. *Given a DHImp h defined with a database $db = \{t_1\}$ where t_1 is defined as follows:*

Num1	*Num2*	*Num3*	w	φ
A	*2*	*4*	*0.1*	*1*
B	*8*	*10*	*0.2*	*2*
C	*15*	*20*	*0.3*	*3*
A	*28*	*3*	*0.4*	*4*
C	*1*	*3*	*0.5*	*5*

and a tensor T with only one layer k:

	1	*2*	*3*	*4*	*5*
1	*0.1*	*0*	*0.3*	*0*	*0*
2	*0*	*0.2*	*0*	*0.5*	*0.1*
3	*0*	*0*	*0.5*	*0.6*	*0*
4	*0.1*	*0*	*0*	*0.2*	*0*
5	*0*	*0.3*	*0*	*0*	*0.5*

the projection operation $\pi_{Num1}(h)$ provides the following result:

Num1	w	φ
A	*2*	*250*
B	*28*	*8*
C	*28*	*987*

	250	*8*	*987*
250	*0.4*	*0*	*0.075*
8	*0.25*	*0.2*	*0.05*
987	*0.2*	*0.15*	*0.25*

Definition 5.1.11 (Data Group by)**.** *The **group by** operation over a DHImp $h = (db, T)$ is defined as follows:*

$$\Gamma^{Op(L_i)\texttt{as } X|D}_{\vec{L}\setminus\{L_i\}}(h) = (\dot{\Gamma}^{Op(L_i)\texttt{as } X}_{\vec{L}\setminus\{L_i\}}(db), Update(T))$$

Concerning the embedding operation, it is not useful to define it as a database operation, since it is more useful to modify only a group of hyperedges at a time. Moreover, the DHImp definition doesn't require to modify the correlation hyperedges when data is extended or removed inside the hyperedge. We also suppose that the following operations won't change the weight of uncertain data.

Definition 5.1.12 (Data Calc)**.** *The **Calc** operation over a DHImp $h = (db, T)$ is defined as an update over the database, and hence: $Calc^D_{Op(\vec{L})\ \texttt{as}\ S}(h) = (Calc_{Op(\vec{L})\ \texttt{as}\ S}(db), T)$.*

Definition 5.1.13 (Data Rename)**.** *The **rename** operation over a DHImp $h = (db, T)$ is defined as the combination of the Calc and Projection operation for relational databases, and hence even in this time:*

$$\rho^D_{R\leftarrow X}(h) = (\rho_{R\leftarrow X}(db), \tilde{\pi}_{S(db)\setminus\{X\}}(T))$$

5.1.3 *D/I-Hypergraphs algebraic operations - Relational operations*

Now we want to perform some other relational operators over binary relations represented in a tensor-like form. We could see that we cant' simply state to merge two $\mathcal{E}_E$ sets from two different databases, since it doesn't make sense to import relations without importing the data, too. In this case the union over $\mathcal{E}_D$ will be the more appropriate decision to follow.

It doesn't make sense either to rename $\mathcal{E}_E$ relations, since we rename only data properties and, as a side effect, we obtain that even the $\mathcal{E}_E$ relations will be renamed, too. Even projection over binary $\mathcal{E}_E$ is not meaningful, since that operation will reduce binary relations to unary relations.

If we want to select only some other properties among the others, then we could redefine the select operation

Definition 5.1.14 (Correlational Selection)**.** *The **correlational selection** operation over a DHImp $h = (db, T)$ is defined as:*

$$\sigma^E_P(h) = (db, \tilde{\sigma}_P(T))$$

where the $\tilde{\sigma}_P$ selection function for the tensor is defined as:

$$\forall i,j,k.(\tilde{\sigma}_P(T))[i,j,k] = \begin{cases} T[i,j,k] & E[i,j,k,T,P] \\ 0 & oth. \end{cases}$$

Discussing joining correlations

We could at this point discuss the meaning of performing the join operation over two table inside a same database. We could think that a $a\Re_k \ltimes \Re_h b$ iff. $\exists c. a\Re_k c \wedge c\Re_h b$, and hence we could think that $\forall a, b. \sum_c T[a, c, k]T[c, b, h]$ solves our problem, that is $T_k \times T_h$.

Similarly we could define $a\Re_k \rtimes \Re_h b \Leftrightarrow b\Re_k \ltimes \Re_h a$ and $a\Re_k \bowtie \Re_h b \Leftrightarrow a\Re_k \ltimes \Re_h b \wedge a\Re_k \rtimes \Re_h b$.

We could see that $\rtimes$ or $\ltimes$ doesn't express in this case semi-joins but only how relations are oriented.

We could see that is not possible to define join operations between two different databases, as different DHImp don't have shared links among them.

Definition 5.1.15 (Correlational Join)**.** *The **correlational join** operation over a DHImp $h = (db, T)$ between two tensor layers k and j for "oriented relations" is defined as follows:*

$$h_{k\ltimes h} = (db, T_k \times T_h) \quad h_{k\rtimes h} = (db, T_{h\rtimes k}) \quad h_{k\bowtie h} = (db, T_{k\bowtie h}) \wedge \forall i, j. T_{k\bowtie h}[i, j] \triangleq T_{k\ltimes h}[i, j]T_{k\rtimes h}[i, j]$$

At this point, let's ponder how to mantain relations among the data types even when some of the data are removed, and then some of the links are removed.

Yet Another Maieutic Dialog

Q: Excuse me if I interrupt your thesis…I expect that the $\mathcal{E}_D$ transformations won't change any correlational relation (the hyperedges in $\mathcal{E}_E$). Let me explain: if I know that:

$$\exists a, b, c, t.\ TDI(\mathcal{H})[a, b, t] \neq 0 \wedge TDI[b, c, t] \neq 0$$

then I could infer that there is some kind of correlation between entities a and c; hence I want to know that $TDI(\mathcal{H})[a, c, t] \neq 0$ if b is removed from my dataset, and hence I would like that all the transformations will be "path-invariant".

A: Well, you haven't explicitly stated if t is a transitive relation or not.

Q: But why that's so important to you?

A: Remember? We want to keep track of which is the type of the relation that links the nodes, and so we want to know "how" my data is connected. So in this case, if t is not a transitive relation, we cannot directly infer that $TDI(\mathcal{H})[a, c, t] \neq 0$…Do you remember that each relation could be represented in a tabular form?

Q: Yes.

A: Well, we could think to use a join relation between the two t relations in order to avoid that, by removing the b edge, the path between a and c will be broken.

Q: But this wouldn't solve a more general problem where is unfeasable to keep track of all the edges involved in such join operations. Isn't there something more general? I want to keep track of this information without necessarily applying the Nuutila algorithm (ed. See [Nuu95].) for transitive closures.

A: Well, there could be two possible ways... The first uses the correlation relation in order to mine general properties over data. At this point we could assume tat these properties are always true, whether the actual data was removed or not. The second choice could involve the seek of correlation among the data or the preexistent operations, by using MCL purely as a matrix multiplication algorithm, in order to detect $TDI(\mathcal{H})^{k+2}$ where $k = \max_g \{ deg(g) \mid g \in \sigma_{\neg P}(\mathcal{H}) \}$ and hence find the correlation among the remaining nodes.

Q: I prefer the second method, is more straightforward.

A: Keep in mind that this second choice we obtain some relations that don't necessarily mantain the same original scheme.

Q: Who cares!

A: We do! Well, we won't use that in my real-world data, since we won't no more know how the data is correlated, and we won't know if the final result is meaningful or not.

Let's first analyze those straightforward methodologies: at first we would like to collapse all the binary relations into a single relation, and then use the MCL algorithm with some parameters, in order to fasten the process of matrix multiplication and detect the correlation between data in more than one step.

Definition 5.1.16 (Summarize). *The **summarize** operator over all the relations of the DHImp $h = (db, T)$ updates T by adding the following relation with a fresh index:*

$$Summarize_{\alpha,\beta}(T) = MCL_{\alpha,\beta}\left(norm\left(\left\{\sum_{l \in \mathbb{T}(\mathcal{E}_E)} T[i,j,l]\right\}_{(i,j)\in T}\right)\right)$$

In this way we have that some data are related with a given uncertainty measure, but we cannot say "how" these data are related: this operations could be useful in order to analyze

how the information could be spread inside a network [Kim+13] and hence useful to detect the active state of a network, but is not useful to mine and detect new relations from our data, and hence to obtain new informations.

We could get an improvement by slightly modifying the data join defined in Definition 5.1.17, in order to join only that data that are related within a link. We could assume that the weight $w(a,b)$ of an arc (a,b) represents the probability that b is true assuming that a is true (with a certain probability). Hence by Bayes's theorem we have that:

$$P(b|a) = \frac{P(a \wedge b)}{P(a)}$$

and hence we obtain that $P(a \wedge b)$ could be the probability that the two data are both true, and that happens when both data are joined together. If we assume that $P(a)$ is the weight $w(a)$, we obtain that:

$$w(a \bowtie b) = w(a,b)w(a)$$

Since the use of $w(a,b)$ implies that we have oriented arcs from a to b, we could point out this specification by adding the "left" term in order to remember the directionality of the definition.

Definition 5.1.17 (Data θ-Join for left-tied data)**.** *The* ***θ-join for tied data*** *operation over a DHImp $h = (db, T)$ is defined as follows:*

$$Tie(\theta, h) = (db \dot{\ltimes}_{x,y \mapsto \theta(x,y) \wedge \exists k.T[x,y,k] \neq 0} db, T \tilde{\bowtie}_\theta^{Tie} T)$$

where the $\tilde{\bowtie}_\theta$ function for the two tensors is defined as:

$$\forall i,j,k.(T \tilde{\bowtie}_\theta^{Tie} T)[i,j,k] = \text{avg}\left\{ i \;\middle|\; i = \text{avg}\{T_i[a,b,k] | a \in dt^{-1}(i) \wedge b \in dt^{-1}(j)\} 0 \wedge T_i \in \{T, T'\} \right\}$$

and the $\dot{\ltimes}$ operator over databases redefines the weight function of $\dot{\bowtie}$ ($DB_1 \dot{\ltimes}_\theta DB_2 = \{ d_1 \ltimes_\theta d_2 \mid d_1 \in DB_1 \wedge d_2 \in DB_2 \}$), and hence the table relational operation is defined as follows:

$$t \bowtie_\theta^{Tie} t = \left\{ \left\langle \mathbb{S}(e) \cup \mathbb{S}(e'), \mathbb{T}(e) \cdot \mathbb{T}(e'), e \cup e', w(e)\, \text{avg}_k\{T[e,e',k]\}, dt(\varphi(e), \varphi(e')) \right\rangle \;\middle|\; e \in t, e' \in t', \theta(e,e') \right\}$$

Example

17. *Drawing on Example 14 on page 70, we could see that this new operator applied to the same DHImp will return an empty DHImp, since no join is possible as for each entity there is no entity linking the two databases via tensors.*

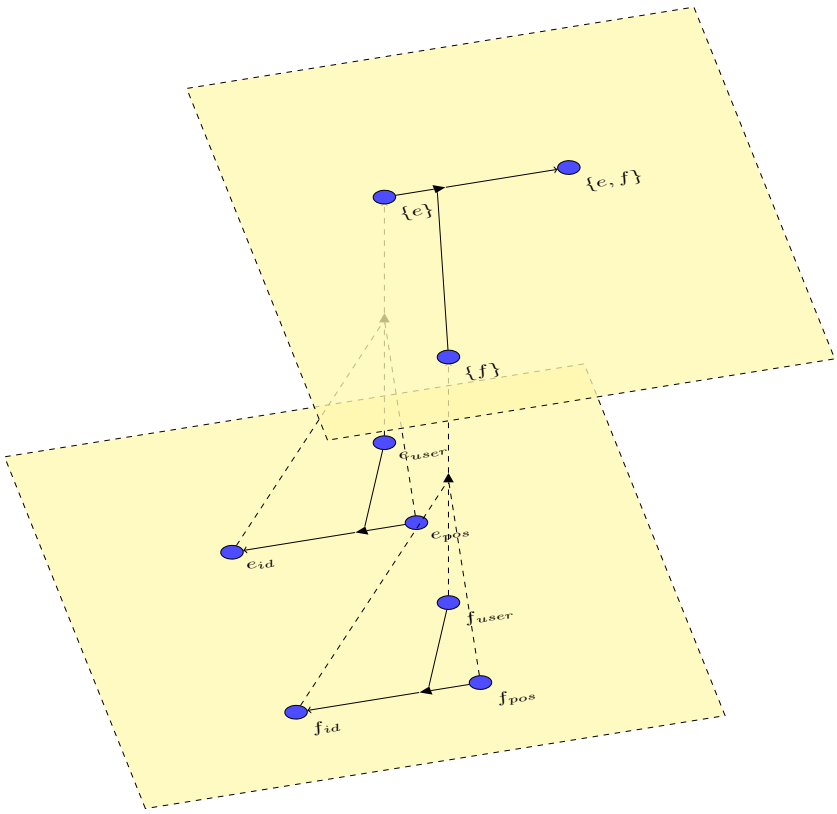


Figure 5.2: *Hypergraph over a data layer (the lower one) and of the mined relations (the upper one). The edges that link the two layers correlate data with its properties.*

5.1.4 *Pure I-Hypergraph for non data-driven relations*

We want to show how in some cases we would like to introduce other relations, that couldn't be necessairly represented as binary relations between databases' entries.

Example

18. *Let's escape from the usual social network domain and let's discuss about an imaginary table of people who survived to the Titanic's shipwreck. Suppose that each entry of this table represent the complete information of the passengers, that is age, gender, cabin's class and if they've survived or not, and to establish that all the people belonging to a certain age interval and that were accommodate in a given class cabin survived.*

In order to do this we have the need to express properties over the data, and hence abstract from the data values, and to collect all the significant values. Moreover, we would like to express a trinary relation that doesn't belong to the data, but that is mined and hence hasn't to be confused with the original correlations.

Let's see now a more abstract example involving some real data hypergraphs.

Example

19. *Suppose that, given a graph data set expressed in a hypergraph form (see Subsection 2.1.2 on page 27), we want to define some clusters over the data hyperedges. If we want to use hierarchical clustering, we would like do be able to do the following things:*

- *We would like to abstract over the data, in order to say that $e \in \mathcal{E}_D$ forms by itself a base sub-cluster that is formed by the only element e, and moreover $\{ e \}$. Hence we would like to:*
  - *place all the data and the data relations in a DHImpl hypergraph layer*
  - *extend our datatypes with datasets*
  - *perform an abstraction relation between e and $\{ e \}$*
- *We want to make clear that the cluster $\{e, f\}$ is made from merging the singlets $\{e\}$ and $\{f\}$ altogether, and that such relation does not belong to our initial data. For this purpose, the second layer will be defined as a **Pure I-Hypergraph**, formed by hyperedges that could*

be interpreted as a set of correlated clauses that define general properties over data. Consequently this layer's hyperedges won't satisfy the property (HG2) on page 28, since it won't contain any $\mathcal{E}_D$ hyperedges.

- *Given that we've defined two distinct layers, the abstraction relation between the data e and $\{ e \}$ will link vertices from different layers: those edges will belong to the so called $\mathcal{E}_L$ set.*

Definition 5.1.18 (Pure I-Hypergraph). *A **Pure I-Hypergraph** is a simple hypergraph $\mathcal{H}$ where all the hyperedges are $\mathcal{E}_E$.*

Definition 5.1.19 (Hypergraph for Data Mining). *An hypergraph for data mining is defined as the following triple:*

$$HDM = (h, \mathcal{H}, \mathcal{E}_L)$$

where $h = (db, T)$ is a DHImpl, $\mathcal{H}_2$ is a Pure I-Hypergraph and $\mathcal{E}_L$ are edges that, given $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and $DB\mathcal{H}(db) = (\mathcal{E}_D, \mathcal{E}_E)$:

$$\forall e \in \mathcal{E}_L.(\forall p \in prev(e).p \in \mathcal{E}_D \cup \mathcal{E}_E) \wedge (\forall p \in next(e).p \in \mathcal{E})$$

5.2 PROMOTING DATA RELATIONS

In this section we want to investigate how we could extract informations from data and "promote" it to relations among the data.

5.2.1 *Expliciting Laplacian data correlation into a matrix form*

In Subsection 4.2.1 on page 61 we've analyzed the Laplacian form for numerical and non categorical data: our aim is to extend such operations even for categorical data: this is possible like showed in [Freo5] whether it is possible to convert it into a index measure that suppose to have a total ordering over the data[2]. That is for any categorical data $T \in \mathcal{T}$ it should be possible to find a "numericizing" function $f : T \to \mathbb{R}$ such that:

$$\forall a, b \in T.f(a) \leqslant f(b) \Leftrightarrow a \preceq b$$

Definition 5.2.1 (Numericize). *Given a list of numericizing functions $\{ f_{t_i} \}_{i \leqslant n}$ where $\{ t_i \}_{i \leqslant n} \subseteq \mathcal{T}$, we could convert all the categorical types $\{ t_i \}_{i \leqslant n}$ into numerical types with the same name in the whole database as follows:*

2 See the example of the "occupation" field that is used with the Duncan's prestige scale.

$$Numericize_{t_1,\dots,t_n}(h) = \rho_{\{t_i\}_{i\leqslant n}\leftarrow\{tmp_i\}_{i\leqslant n}}(\pi^D_{\mathcal{T}\setminus\{t_i\}_{i\leqslant n}}(Calc^D_{f_{t_n}\ \texttt{as}\ tmp_n}(\dots Calc^D_{f_{t_1}\ \texttt{as}\ tmp_1}(h))))$$

Now we could define a correlation function among all the data: we should notice how even in this case this similarity binary hyperedges $\mathcal{E}_E$ will have a $\{\mathbb{E}_D, \mathbb{E}_D\}$ schema, and then use the `VertexClust` function defined in Algorithm 4.5 on page 62:

Definition 5.2.2 (Data correlation layer)**.** *The **data correlation relation** in the tensor T of a DHImpl $h = (db, T)$ could be expressed as follows:*

$$T_{corr}(f_{t_1}, \dots, f_{t_n}) = \texttt{VertexClust}(Numericize_{f_{t_1},\dots,f_{t_n}}(dbA(db)))$$

5.3 HYPERGRAPH DATABASES: A SUPERFLUOUS DEFINITION

At this point, we could be interested in defining some operations over hypergraph databases, as in literature graph-databases are oftenly used (see Definition 4.1.2 on page 52). Given an HYPERGRAPH DATABASE $HDB = \{ db_i, T_i \}_{i\leqslant n}$ described as a collection of *DHImp*s, we would like to define transformations $\mathcal{A}$ that provide another HDB as a result:

$$\mathcal{A}(\{ db_i, T_i \}_{i\leqslant n}) = \{ db_i, T_i \}_{i\leqslant m}$$

Since we would like to avoid the repeated data inside each db_i, we would like to define such database as a $(\dot{\bigcup}_{i\leqslant n} db_i, \{ T_i \}_{i\leqslant n})$ where $\{ T_i \}_{i\leqslant n}$ are collections of tensors for binary relations. At this point, we could collapse all the tensors T_i in a new tensor HT defined as follows:

$$HT[i, j, k] = T_{\texttt{snd}(dt^{-1}(k))}[i, j, \texttt{fst}(dt^{-1}(k))]$$

We could easily see that an HDB could be simply mapped into a DHImp, where the type field $k \in \mathcal{T}$ could be interpreted as a couple $\langle \mathcal{G}, t \rangle$ which points out the type of the relation and the graph to which the relation belongs. As a final remark, we could be not interested in defining hypergraph databases, since we could map all the information in a single hypergraph.

This brings as a primary result that the definition of brand new hypergraph database operations is avoidable. A similar discussion could be carried out also for HDMs.

Part II

HYPERGRAPH APPLICATIONS

6

GSPAN EXPANSION - USING SUBGRAPH MINING FOR OUR GRAPH DATABASE DEFINITION

Contents

We've already seen in Subsubsection 4.1.1.2 on page 54 that gSpan is an algorithm that could detect the frequent subgraphs $\{ \gamma_i \}_{i \leqslant n, n \in \mathbb{N}}$ of a graph database $DB = \{ G_i \}_{i \leqslant m, m \in \mathbb{N}}$. We want to check if we could improve this algorithm by specializing it over our DHImp set of data, since we've already observed that a graph database could be easily mapped to such DHImp.

6.1 TARGETING THE SUBRAGRPH ISOMORPHISM OVER DHIMP

First of all we should consider that we've defined our graph relations, represented as tensors, in order to avoid data replication, and hence we have that all the vertices have different values.

Definition 6.1.1 (Lexicographic Order)**.** *Given two tuples $\vec{x}, \vec{y} : D_1 \times \cdots \times D_n$, we define the $\preceq$ lexicographic order ordering as follows:*

$$\vec{x} \preceq \vec{y} \Leftrightarrow \exists k \in [1, n].\ \left(\forall j \in [1, k). x_j =_j y_j\right) \wedge x_k \prec_k y_k$$

Definition 6.1.2 (ERTriple)**.** *A triple $(a, \lambda(b), c) \colon \mathcal{E}_D \times \mathbb{T}(\mathcal{E}_E) \times \mathcal{E}_D$ is an **ERTriple** for the $DHImp$ $h = (db, T)$ iff. $T_{\lambda(b)}[a, c] \neq 0$. The set of h's ERTriples is noted as $\mathsf{ecode}(h)$.*

Considering that for each ERTriple (a, b, c) each component could be represented as a string, we could define a lexicographic ordering $\leqslant_c$ among two ERTriples that is induced by the string ordering.

We could also see that, given the actual definition of $DHImp$, we cannot have different edges with the same type $k \in \mathcal{T}$, and hence each $DHImp$'s ERTriple is unique. At this point we are no more interested in the subgraph isomorphism formulation given in Definition 4.1.3

on page 52, but we could give the following preliminary definition: *Given two $DHImp$ h and h', we say that h is a subhypergrpah of f' ($h \subseteq h'$) iff.:*

$$\mathcal{E}_D(h) \subseteq \mathcal{E}_D(h') \wedge \texttt{ecode}(h) \subseteq \texttt{ecode}(h')$$

where in this case $\mathcal{E}_D(db, T) \triangleq \bigcup_{t \in db} t$.

Drawing on Definition 4.1.3 on page 52, we would like to map a relation in $\texttt{ecode}(h)$ to its twin one in h': no other isomorphisms are considered interesting, since we are concerned in finding a subgraph of a given graph where, in both representations, all the vertices are unique and where only a single instance of an edge between the same two vertices and with a given type k is allowed. Moreover, this implies that the mapping function f of the subgraph isomorphism in which we are interested is simply defined as an identity function ($f(v) = v$).

Since for each DHImpl h's admissible ERTriple $(a, b, c) \in \texttt{ecode}(h)$ we have that $a, b \in \mathcal{E}_D(h)$, we have that, if $\texttt{ecode}(h) \subseteq \texttt{ecode}(h')$ for some other DHImpl h', we could have by definition that $\mathcal{E}_D(h) \subseteq \mathcal{E}_D(h')$.

Hereby we could reduce the previous narrow definition in this more compact result:

Lemma 6.1 (Narrow DHImpl Subhypergraph isomorphism)**.** *Given two $DHImp$ h and h', we say that h is a subhypergrpah of f' ($h \subseteq h'$) iff. $\texttt{ecode}(h) \subseteq \texttt{ecode}(h')$.*

At this point we could reduce the $\texttt{ecode}(h) \subseteq \texttt{ecode}(h)$ check cost to a string matching if we consider $\texttt{ecode}(h)$ and $\texttt{ecode}(h')$ as ordered sets. We could hereby define the following simple Java algorithm:

```
public boolean subgraphOf(List<ERTriple> sub, List<ERTriple> sup) {
    if (sub == null)
        return true;
    else if (sup == null)
        return false;
    while (!sub.isEmpty()) {
       while ((!sup.isEmpty()) && (!sub.get(0).equals(sup.get(0)))){
            if (sup.size()>1)
                sup = sup.subList(1, sup.size());
            else
                sup = new LinkedList<>();
        }
        if ((sup.isEmpty()) && (!sub.isEmpty()))
            return false;
        //System.out.println(subI.hasNext()+" "+supI.hasNext());
        while ((!sub.isEmpty()) && (!sup.isEmpty()) && sub.get(0).equals(sup.get(0))){
            if (sub.size()>1)
                sub = sub.subList(1, sub.size());
            else
                sub = new LinkedList<>();

            if (sup.size()>1)
                sup = sup.subList(1, sup.size());
            else
```

```
25                  sup = new LinkedList<>();
                }
            }
            return true; // sub == empty is a subgraph
        }
```

This algorithm has optimal complexity, since it could be computed in linear time ($O(|sup|)$). This means that the frequency support function given in Definition 4.1.4 on page 54 could be computed with a time complexity linear in the size of the graph database ($O(|DB|)$) that, in this case, could be directly mapped in a single DHImp. As we are reduced to a particular case of graphs over which perform the subgraph mining, we cut the time complexity of a potentially NP-complete problem to a linear one.

6.2 GSPAN OVER DHIMP

We've already seen that the previous considerations reduced the overall complexity of subgraph isomorphism to a simple ordered string matching problem via the DHImp definition.

Algorithm C.9 on page 157 provides a partial Java implementation of the pseudocode provided in Algorithm 4.2 on page 55: we could see that we didn't implement the $n \neq minDfsCode(n)$ test because we preferred to provide such pruning check in `getRightmostExpansions` (see Algorithm C.10 on page 158). In this method we perform a DFS visit of the rightmost path of the graph ($O(d)$, where d is the maximum depth reachable explorign the graph) and then `filterTriplesForExpansion` removes the edges that could break the rightmost path. This check is necessary as it provides the creation of isomorphic subgraphs, as explained in [Sam+13]. The introduction of the `depth_search` parameter is also useful to define the upper bound of the size of the mined frequent subgraphs.

We could now analyze the theorical time complexity of the proposed solution in the light of new knowledge and of the previous results; drawing on the theorical implementation offered in Algorithm 4.2 on page 55, we have that the most extern cycle iterates over $|\Xi|$ elements, while the rightmost expansion has the computational cost of:

$$in(i) = \begin{cases} |\Xi|(|DB| + in(i-1)) & i > 1 \\ |\Xi||DB| + c & i = 1. \end{cases}$$

that is $O(|\Xi|^i|DB|)$: this provides an overall time complexity of $O(|\Xi|^{i+1}|DB|)$.

We could observe that this result couldn't be applied to all the data structures that could be represented as graphs, since not all the data could be represented with unique labels (i. e. chemical databases, where each atom could be can appear multiple times within the same molecule). By the way, we've developed this solution not for some other general contexts, but for a specific scenario, that is the representation of the OSN data in an informatic context. It is our opinion that other specific contexts could require different data structures where the optimizations here presented are not necessarily possible.

6.3 gSpan extended: an implementation

Our `gSpanExtended` library (`https://github.com/jackbergus/gSpanExtended`) provides an implementation of the previously defined algorithm. An overview of this algorithm is provided by Algorithm C.7 on page 151.

7

HYPERGRAPH FOR DATA MINING IMPLEMENTATIONS

Contents

After having showed the proposed implementation of the hypergraphs for data mining, we want to suggest an implementation in R. We suggest to read Chapter B on page 135, since this implementation uses a simple R extension of our design in order to add a simple type system inside the language.

7.1 DEFINING THE DATA STRUCTURES IN R

7.1.1 *Tables and Table Records*

Let's define the $e = \langle \mathbb{S}(e), \mathbb{T}(e), \mathbb{D}(e), w(e), \varphi(e) \rangle$ structure for a record.

- $\mathbb{S}(e)$: A schema is a list of types, and then it could be simply defined as:

```
Schema <- new.Subtype("Schema",function(ls) prod(sapply(ls,function(x) is.Kind(x)))
    ,List)
```

- $\mathbb{T}(e)$: A type is defined as a string, and hence we could easily extend the definition of the `String` type as follows:

```
DType <- new.Subtype("DType",function(ls) T,String)
```

- $\mathbb{D}(e)$: The data is simply defined as a list of values. In order to check if the data reflects the schema, we have to define the `isInSchema` in order to define the consistency of the record as follows:

```
isInSchema <- function(ls,S) prod(mapply(function(x,y) x$isType(y),S,ls))
is.Record <- function(r) {
  if (!Schema$isType(r$S))
    print("Not a Schema")
  if (!DType$isType(r$Ty))
    print("Not in Type")
  if (!isInSchema(r$D,r$S))
    print("Data not in schema")
  if (!is.numeric(r$w))
    print("Weight is not numeric")
  if (!is.bigz(r$phi))
    print("Phi is not big int")
  return(Schema$isType(r$S) && DType$isType(r$Ty) && isInSchema(r$D,r$S) &&
      is.numeric(r$w) && is.bigz(r$phi))
}

Record <- new.Type("Record",is.Record)

## Creates a new record if the type is correct
initList <- function(S,Ty,D,w,i) list(S=S,Ty=Ty,D=D,w=w,phi=i)
new.Record <- function(S,Ty,D,w,i) {
  ls <- initList(S,Ty,D,w,i);
  if (Record$isType(ls)) {
    return(ls)
  } else {
    return(NULL)
  }
}
```

- $w(e)$: the weight is implemented for the moment as a simple numeric value, that could be extended in the future as a vector storing different kind of values.
- $\varphi(e)$ The indexing value of each table entry is implemented using an R wrapping [Luc+13] for the `gmp` library for defining integers with arbitrary precision and another wrapping [Mae13] over `mpfr` that makes some operations (such as square roots with numbers with arbitrary precision) available.

A table t is here defined as a $\langle \mathbb{S}(t), \mathbb{T}(t), \mathbb{D}(t) \rangle$ where $\mathbb{D}(t)$ is a list of rows with all the same schema $\mathbb{S}(t)$ and type $\mathbb{T}(t)$:

```
Table <- new.Type("Table",function(x) {
  Schema$isType(x$S) &&
    DType$isType(x$Ty) &&
    #Heavy check: problem: i have to do lapply and then unlist... :(
    prod(unlist(lapply(x$dTable,function(row) {
      is.Record(row) && (row$S %==% x$S) && (row$Ty %==% x$Ty)
    }))) }
)
new.Table <- function(S,Ty,lis) {
```

```
  ls <- list(S=S,Ty=Ty,dTable=lis)
  if (Table$isType(ls))
    return(ls)
  else
    return(NULL)
```

In the R algebra implementation we stop at the $DHImp$, that is the tensor-database representation of an hypergraph with only binary relations. The tensor is simply defined as a three-dimensional array, and hence we could provide the final data structure implementation as:

```
new.Tensor <- function(ndata,layers) {
  return(array(0,dim=c(ndata,ndata,layers)))
}

new.DHimpl <- function(db,ten) {
  return(list(db=db,t=ten))
}
```

7.1.2 *Implementing the operators*

Given these first definitions, we could implemented as done in Algorithm C.3 on page 143 the operations for lists of rows: these definition are the base for the table definitions provided next in Algorithm **??** on page ??. By the way R doesn't seem a very good language over which implement complex data structures and algorithms, since its typecast and the collection unlisting utilities are so naïve that became hard to handle for nested structures. Moreover, the resulting code becomes very hard to read and handle.

7.1.3 *Some examples*

Lets define a siple hypergraph database where we use some basic numeric types:

```
Num1 <- new.Subtype("num1",function(x) T,Numeric)
Num2 <- new.Subtype("num2",function(x) T,Numeric)
Num3 <- new.Subtype("num3",function(x) T,Numeric)
Num4 <- new.Subtype("num4",function(x) T,Numeric)
Num5 <- new.Subtype("num5",function(x) T,Numeric)
Num6 <- new.Subtype("num6",function(x) T,Numeric)
```

As a second step, we could define a table named `dtable1`, where the third field of `new.Record` defines the data that should be stored inside the data row. The last field is the row id, where each row is seen as an single entity inside the whole database.

```
##  Defining table dTable1
r1 <- new.Record(list(Num1,Num2,Num3),"Elem",list(1,2,3),1,as.bigz(1))
r2 <- new.Record(list(Num1,Num2,Num3),"Elem",list(1,2,6),1,as.bigz(2))
```

```
r3 <- new.Record(list(Num1,Num2,Num3),"Elem",list(1,4,3),1,as.bigz(3))
dtable1 <- list(r1,r2,r3)
tab1 <- new.Table(list(Num1,Num2,Num3),"Elem",dtable1)
db1 <- list(tab1)
```

that is:

Num1	Num2	Num3
1	2	3
1	2	6
1	4	3

We could also another table which is formed by only one entity:

```
r1b<- new.Record(list(Num4,Num5,Num6),"Ele2",list(1,2,3),1,as.bigz(4))
tab2 <- new.Table(list(Num4,Num5,Num6),"Ele2",list(r1b))
db2 <- list(tab2)
```

that is:

Num4	Num5	Num6
1	2	3
1	2	6
1	4	3

Now we could define two tension layers, `ten` and `nal`, as follows:

```
ten <- new.Tensor(4,1)
ten[1,2,1] <- 0.5
ten[1,1,1] <- 0.8
ten[3,1,1] <- 1
ten[3,1,1] <- 0.6
ten[3,3,1] <- 0.4

nal <- new.Tensor(4,1)
nal[4,4,1] <- 1
```

In this simple R implementation we suppose to provide a single tensor to each *DHImp*, and hence we could provide the following simple implementation:

```
h1 <- new.DHimpl(db1,ten)
h2 <- new.DHimpl(db2,nal)
```

7.2 DEFINING THE WHOLE DHIMP IN JAVA

Our library `hypergraphalgebra` (https://github.com/jackbergus/hypergraphalgebra) provides an implementation of the previously defined algebra, in order to give a proof of concept of how the hypergraph computation could be carried out. In this case each tuple has an associated schema which is defined by the classes of the objects used inside the tuple.

```
public class Tuple {

    private Class classes[];
    private Object elems[];
    double w;
    BigInteger index;

    public Tuple(Class... clazzes) {
        classes = clazzes;
        elems = new Object[clazzes.length];
    }

    public double getWeight() {return w;}
    public void setWeight(double d) { w = d; }
    public BigInteger getIndex() { return index; }
    public void setIndex(BigInteger i) { this.index = i; }
    public Class[] getSchema() { return classes; }

    public Tuple(double weight, BigInteger index, Object... objs) {
        elems = objs;
        classes = new Class[objs.length];
        for (int i=0; i<objs.length; i++) {
            classes[i] = objs[i].getClass();
        }
        w = weight;
        this.index = index;
    }

 /* missing code */

}
```

Each table is defined is viewed from the programmer point of view as a `List<Tuple>` with a same schema `classes`, even if the underlying implementation uses a `Map` to provide a prompt access to each entry.

```
public class Table implements List<Tuple> {

    private Class classes[];
    private String tName;
    private BigInteger countFrom;
    private Map<BigInteger,Tuple> tupz;
    public static BigInteger ERROR = new BigInteger(Integer.toString(-1));

    public Set<BigInteger> getAllKeys() {
        return tupz.keySet();
    }

    public Table(String name, Class ... clazzes) {
        this.classes = clazzes;
```

```
        tupz = new HashMap<>();
        tName = name;
        countFrom = BigInteger.ZERO;
    }

    public Table(String name, BigInteger startIndex, Class ... clazzes) {
        this.classes = clazzes;
        tupz = new HashMap<>();
        tName = name;
        countFrom = startIndex;
    }

  /*missing code*/
}
```

The database is implemented as a simple `Map` between relation names and their tables:

```
public class Database implements Map<String,Table> {

    private Map<String,Table> db;

    public Database() {
        db = new HashMap<>();
    }

    public Database(Map<String,Table> database) {
        db = database;
    }

    /* missing code */
}
```

Even the tensors are implemented with a `Map` between the layer name and the actual `ITensorLayer` layer implementation; in our code we implemeted the binary relations via Guava[1] Tables, that permit a sparse representation of the matrices. $DHImp$ is simply implemented as a couple between a relational database and a tensor:

```
public class DHImp<T extends ITensorLayer> {

    private Database first;
    private final Tensor<T> second;

    public DHImp(Database db, Tensor<T> tensor) {
        first = db;
        second = tensor;
    }

    public Database getDB() {
        return first;
```

1 https://code.google.com/p/guava-libraries/

```
    }
    public Tensor<T> getT() {
        return second;
    }

    public Class<T> getTClass() {
        return second.getLayersClass();
    }

}
```

At this point we could define some `interfaces` in order to define the functional behaviour of our operators:

- `IPhi` interface provides the f function for the reindexing function Φ (see Definition 3.3.1 on page 44).
- `IMapFunction` interface provides the implementation of the $\bigoplus$ function used in Γ (see Definition 3.2.1 on page 39).
- `IJoinProperty` interface provides the implementation of the binary property used in $\bowtie$ operations.
- `ICalc<Return>` interface permits to calculate the value to be nested inside each database entry as required by *Calc* definition (see Definition 3.2.1 on page 39).

Section C.3 on page 146 shows how some relational operators are implemented.

7.2.1 *A brief example*

The code needed for check the result in Example 17 on page 77 could be simply reproduced using this library by executing the following code.

```
        Table one = new Table("t1",Dovetailing.index(1),String.class, Integer.class,Double.
            class);
        one.addRow(0.1, "A",2,4.0);
        one.addRow(0.2, "B",8,10.0);
        one.addRow(0.3, "C",15,20.0);
        one.addRow(0.4, "A",28,3.0);
        one.addRow(0.5, "C",1,3.0);

        Table two = new Table("t1",Dovetailing.index(6),String.class, Integer.class,Double.
            class);
        two.addRow(0.2, "A",7,11.0);
        two.addRow(0.1,"A",2,4.0);

        Database db1 = new Database();
        db1.add(one);
        Database db2 = new Database();
```

```
        db2.add(two);

        BinaryRelationsTensor tensor1 = new BinaryRelationsTensor();
        tensor1.set(1, 1, "A", 0.1);
        tensor1.set(1, 3, "A", 0.3);
        tensor1.set(2, 2, "A", 0.2);
        tensor1.set(2, 4, "A", 0.5);
        tensor1.set(2, 5, "A", 0.1);
        tensor1.set(3, 3, "A", 0.5);
        tensor1.set(3, 4, "A", 0.6);
        tensor1.set(4, 1, "A", 0.1);
        tensor1.set(4, 4, "A", 0.2);
        tensor1.set(5, 2, "A", 0.3);
        tensor1.set(5, 5, "A", 0.5);

        BinaryRelationsTensor tensor2 = new BinaryRelationsTensor();
        tensor2.set(6,7,"A",1);
        tensor2.set(7,6,"A",1);

        DHImp<GuavaBinaryTensorLayer> dh1 = new DHImp<>(db1,tensor1);
        DHImp<GuavaBinaryTensorLayer> dh2 = new DHImp<>(db2,tensor2);
        HypergraphOperations<GuavaBinaryTensorLayer> go = new HypergraphOperations<>();

        result = go.HLeftTiedJoin(dh1, new IJoinProperty() {
            @Override
            public boolean property(Tuple left, Tuple right) {
                return (left.get(0).equals(right.get(0)));
            }
        }, dh2);

        for (com.google.common.collect.Table.Cell<BigInteger, BigInteger, Double> cell:
             result.getT().get("A").getValueRange()) {
            System.out.println(cell);
        }
        for (Tuple x:result.getDB().get("t1|><|t1")) {
            System.out.println(x);
        }
        System.out.println("~~");
```

8

SENTIMENT ANALYSIS OVER TIME

Contents

Sentiment Analysis is " *the computational study of people's opinions, appraisals, attitudes, and emotions toward entities, individuals, issues, events, topics and their attributes* " [LZ12], and more specifically a Data Mining field of study. Its aim is to use text mining techniques, such as **Supervised** [1] or **Unsupervised** [2] **learning** for artificial intelligence techniques, compile a *opinion lexicon* via a dictionary-based (i. e. using an online dictionary) or a corpus-bases (i. e. finding syntactic or cooccurrence patterns over a large corpus of documents), and also some spam detection techniques.

In the current chapter, we'll study how to implement the logical overview provided by [LZ12] in a hypergraph implementation.

8.1 DEFINITIONS

Definition 8.1.1 (Entity). *A sentiment analysis's* ***entity****[LZ12] is the subject of investigation for which it is interesting to gather some opinions. It's defined by a hierarchy of components and subcomponents T and a set of attributes E.*

1 Supervised learning uses "*algorithms that guide the training process of the [neural] network by taking in account the desired answer that the network should provide for a given set of training patterns*" [FM08].

2 Supervised learning uses "*algorithms that allow the network to extract statistically significant information from the distribution of the input patterns or to memorize and reconstruct those input patterns*" [FM08].

Given this, we must express T and E in a hypergraph database. Given the considerations provided in Subsection **??** on page ??, we could observe that the taxonomy T of a given entity should be implemented with hypergraph relations that express the subtyping relation $<:$.

Definition 8.1.2 (Entity Hierarchy over Hypergraphs)**.** *Given an **entity** e based of components c ($c <: e$) and subcomponents ($\exists c.c <: e \Rightarrow s <: c$) expressed with relations $\tau(a <: b) = r$ where:*

- $r = \{ a, b \}$
- $\mathcal{T}(r) =$ *part-of*

The attributes over each entity and its components or subcomponents are expressed with relations $\tau(a.\mu) = r$ where:

- $r = \{ \mu, a \}$
- $\mathcal{T}(r) =$ *attribute-of*

The entity e is collcated at the center of this hierarchy.

From this definition, we must also see that the entities must be implemented as ordered sets, otherwise we could state that both $a <: b$ and $b <: a$ hold. Similar observations could be given for the $a.\mu$ implementation.

At this point, given a generic document (e. g. a tweet), we are interested to implement a opinion relation drawing on the one provided by [LZ12]:

Definition 8.1.3 (Opinion)**.** *An **opinion** is expressed as a quintuple*

$$(e_i, a_{ij}, oo_{ijkl}, h_k, t_l)$$

where e_i is the name of an entity, a_{ij} is an aspect of e_i , oo_{ijkl} is the orientation of the opinion about aspect a_{ij} of entity e_i , h_k is the opinion holder, and t_l is the time when the opinion is expressed by h_k . The opinion orientation oo_{ijkl} can be positive, negative or neutral, or be expressed with different strength/intensity levels. When an opinion is on the entity itself as a whole, we use the special aspect GENERAL to denote it.

Given the Definition 8.1.2, we could see that is not necessary to specify that a_{ij} belongs to e_i, inasmuch as the *part-of* and *attribute-of* carry out this task. We consequently assume that different entities have different attributes or components/subcomponents, otherwise some opinions on certain entities could be wrongly inherited by entities with same subcomponents. *We can therefore establish that two entities can share the same components, sub-components or attributes if and only if these are exactly the same elements (e. g. they are of the same trademark and model), and do not only perform the same task (e. g. they are both camera lens).*

At this point, we could model what a general document instance d is:

Definition 8.1.4 (Document over Hypergraphs). *A **document** hyperedge is an entity d of id $d_{\texttt{id}}$ which contains some information i, it has been produced in a time t and belongs to an user h:*

$$d = (i, t, h, d_{\texttt{id}}) \quad \mathcal{T}(d) = document$$

This relation could be arbitrarly extended with some geolocation information or other attributes. In particular, i could be a common vertex (e. g. an id vertex) that links the document to an contents hyperedge.

Hereby, we could easily provide an opinion implementation over hypergraphs:

Definition 8.1.5 (Opinion relation over Hypergraphs). *An **opinion hyperedge** is a relation:*

$$o = (e, oo, d_{\texttt{id}}) \quad \mathcal{T}(o) = opinion$$

where e is an entity, oo is an opinion value, and $d_{\texttt{id}}$ is the document's id where such opinion is expressed.

Given the latter definition, we could easily accomplish the desired and described implementation of an **opinion summary** in [LZ12], by showing the hypergraph of the entity hierarchy enriched with its related opinions.

8.2 TWITTER DATA EXTRACTION

R is a tool used by data miners[3] in order to obtain statistical reports and analyze the data. Recenlty a book on this subject ([Zha13]) has been published on this subject, which points out how this tool could be also used for dowload Twitter data, and also to process the tweets with Text Mining tools: all this is possible via libraries, that extend the programming language.

Given that R has some libraries that implement relational algebra [MH13] over data frames[4], provides a library (`twitteR` [Gen13]) for download data from Twitter, for carrying out Sentiment Analysis (`sentiment` [LZ12]) and to implement and visualize hypergraphs (`hyperdraw` [Mur13]), we choose to analyze in depth this tool, in order to have subsequently a common language over witch operate benchmarking over operation's execution.

Since the library `twitteR` that download Twitter data uses the `libcurl` native library, in order to install R, we should provide the following shell command:

```
sudo apt-get install R libcurl4-gnutls-dev
```

After that, we could open the R interpreter with superuser privileges, and then install some useful R libraries.

3 See `http://www.rexeranalytics.com/Data-Miner-Survey-Results-2011.html` and `http://java.sys-con.com/node/2288420`.

4 A **data frame** is a tabular representation a collection of instantiations of a same given relation. More precisely, "*is a list, with each component of the list being a vector corresponding to a column in our "matrix" of data*" [Mat11].

```
install.packages("twitteR")    # Twitter Data download
install.packages("tm")         # Text mining tools
install.packages("SnowballC")  # Library needed by tm
```

With a second step, we could visit https://dev.twitter.com/ in order to create a new application, in order to obtain some *consumer Key* and *consumer Secret* useful for the application itself. After that, we could authenticate in R in order to access the Twitter database.

```
library(RCurl)
library(twitteR)
library(ROAuth)

requestURL <- "https://api.twitter.com/oauth/request_token"
accessURL = "http://api.twitter.com/oauth/access_token"
authURL = "http://api.twitter.com/oauth/authorize"
consumerKey = "rMr3dUlgAzD52PWyrVxbZA"
consumerSecret = "U2QNmShCryW51OvAyv0PXA9hWsurXrCejkCq41Ec"
Cred <- OAuthFactory$new(consumerKey=consumerKey,
                         consumerSecret=consumerSecret,
                         requestURL=requestURL,
                         accessURL=accessURL,
                         authURL=authURL)
#The authentication procedure will provide a link, where to authorize the application to
    using your twitter account, and hence provide the application a PIN number requested
    accomplish the authentication.
Cred$handshake(cainfo = system.file("CurlSSL", "cacert.pem", package = "RCurl") )
registerTwitterOAuth(Cred)
```

After inserting the pin, we could query the database, and hence obtain all the tweets that contain the word "Beethoven".

```
tw <- searchTwitter("Beethoven",n=30)
```

At this point, we could export this twitter list in a tabular form like this:

```
db <- twListToDF(tw)
```

This command could be useful to obtain also the data referring to the users, and we could also export this data in a csv file with the following command:

```
write.csv(file="/path/to/file.csv",x=db)
```

From now on, we'll assume that the default DBMS used will be a PostgreSQL one. At this point, we could also define a SQL table to store tweets with the following table:

```
\begin{sql}
Create Table Users(
  R_id INT NOT NULL,
  text TEXT not null,
  favorited boolean NOT NULL,
  favoriteCount INT NOT NULL,
```

```
  replyToSN INT,
  truncated boolean not null,
  replyToSID boolean,
  id INT NOT NULL PRIMARY KEY,
  replyToUID int,
  statusSource TEXT not null,
  screenName TEXT NOT NULL,
  retweetCount INT NOT NULL,
  isRetweet boolean not null,
  retweeted boolean not null,
  longitude float(24),
  latitude float(24)
);
```

All the CSV could be imported in PostgreSQL via the following command:

```
COPY Users FROM '/home/gyankos/testcsv.csv' DELIMITER ',' CSV;
```

In order to do some Text Analysis in R, we need to install the tm library and the SnowballC for the Stemming. We could start to process the twitter data corpus by invoking the following function:

```
db$text <- Corpus(VectorSource(db$text))
```

At this point, before performing some stemming operations over the tweet data, we could be interested to obtain all the links, the hashtags and the referrals that could be subsequently removed. In order to do this, we could define two functions, that could define associations between tweet id and links, hashtags and mentions that are contained.

```
df1 = function(x,y) data.frame(post_id=c(x),has_url=c(y))
df2 = function(x,y) data.frame(post_id=c(x),has_reference=c(y))
df3 = function(x,y) data.frame(post_id=c(x),has_hashtag=c(y))
```

Subsequently, given a *data frame*[5] `x` and a Twitter data frame `db`, we could provide the following function that doesn't update `x`, but provides as a return value an updated version of the previous.

```
has_links_relation <- function(x,db) {
  for (i in 1:length(db$id)) {                    # For each post...
    regex <- gregexpr("http://(\\S+)",db$text) # ... extract the url position
    if (regex[[i]][1] != -1)                   # If a url was found ...
    for (j in 1:length(regex[[i]])) {          # ... For each url found
      #get the string termination index
      getattr <- attr(regex[[i]],"match.length")[j]
      getsubstr <-  substr(db$text[[i]],regex[[i]][j],regex[[i]][j]+getattr-1)
      # Update the x copy with a new data frame, i.e. a new relation between url and tweet
          id
      x <- rbind(x,df1(db$id[[i]],getsubstr))
```

5 A *data frame* "is used for storing data tables" via "a list of vectors of equal length". http://www.r-tutor.com/r-introduction/data-frame

```
    }
  }
  x
}
```

We could define similar functions for retreiving user mentions and hashtags by simply changing the regular expression.

```
has_communication_with <- function(x,db)
{
  for (i in 1:length(db$id)) {
    regex <- gregexpr("@(\\S+)",db$text)
    if (regex[[i]][1] != -1)
    for (j in 1:length(regex[[i]])) {
      getattr <- attr(regex[[i]],"match.length")[j]
      getsubstr <-  substr(db$text[[i]],regex[[i]][j],regex[[i]][j]+getattr-1)
      x <- rbind(x,df2(db$id[[i]],getsubstr))
    }
  }
  x
}

has_hashtag <- function(x,db)
{
  for (i in 1:length(db$id)) {
    regex <- gregexpr("#(\\S+)",db$text)
    if (regex[[i]][1] != -1)
    for (j in 1:length(regex[[i]])) {
      getattr <- attr(regex[[i]],"match.length")[j]
      getsubstr <-  substr(db$text[[i]],regex[[i]][j],regex[[i]][j]+getattr-1)
      x <- rbind(x,df3(db$id[[i]],getsubstr))
    }
  }
  x
}
```

All this data frames are exportable as CSV tables by the previously given command

```
write.csv(file="/path/to/file.csv",x=df)
```

where `df` is the exported data frame.

It is also possible to obtain tweets from a given user id, by invoking the following function:

```
userTimeline("gyankos",n=100)
```

where `n` stands for the number of tweets that should be used.

8.2.1 *Trend Mining*

`twitteR` gives also some facilities to perform some trend mining. For example, we could be interested to find which trends are available in a given area. For example, we could provide some geographic coordinates in order to obtain the **woeid**[6] and next obtain the desired trends:

```
t <- closestTrendLocations(44.49428,11.34677)
print(t$woeid)
getTrends(t$woeid)
```

with the function `availableTrendLocations()`, could provide the locations of some places around the world that have some trends, in order to subsequently use the `getTrends` with the region's woeid.

8.2.2 *Users and Users' timeline*

It is also possible to retreive data from a user's timeline. We could use the function `userTimeline` in order to retreive all the tweets of a given user.

```
t <- userTimeline("gyankos",n=100, includeRts=TRUE)
```

Similarly, all this tweets are collected in data frames, which could be exported in a csv format. By the `getUser` function, we could also retreive some informations about the Twitter users, which SQL table could be represented as follows:

```
Create Table Users(
  R_id INT NOT NULL,
  description TEXT not null,
  statusesCount INT NOT NULL,
  followersCount INT NOT NULL,
  favoritesCount INT NOT NULL,
  friendsCount INT NOT NULL,
  url TEXT not null,
  name TEXT not null,
  created TEXT not null,
  protected boolean not null,
  verified boolean not null,
  screenName TEXT not null,
  location TEXT not null,
  ID INT NOT NULL PRIMARY KEY,
  listedCount INT NOT NULL,
  followRequestSent boolean not null,
  profileImageUrl TEXT not null
);
```

6 *woeid*s are geographical identifications assigned by Yahoo! http://developer.yahoo.com/geo/geoplanet/guide/concepts.html

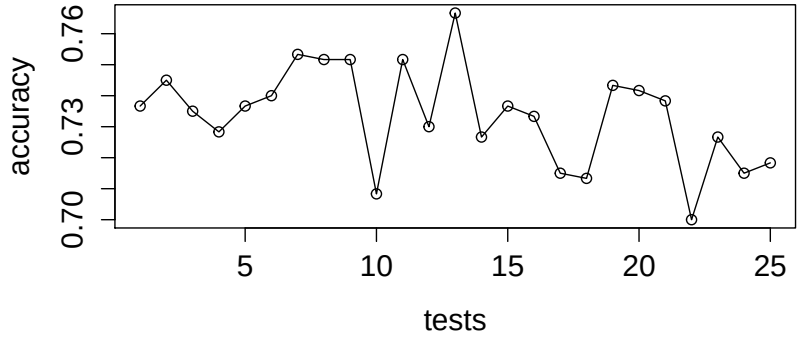


Figure 8.1: *Accuracy testing over Algorithm C.13 on page 163.*

8.3 DATA MANIPULATION

8.3.1 *Analyzing small datasets (movie reviews)*

Let's analyze the following algorythm: it uses a simple algorithm that, after stemming all the document set of positive and negative posts[7], collects the positive and negative lexicon vocabulary[8] and then tries to check if the first positive document has more positive words than negative ones, and vice versa (see Algorithm C.11 on page 160).

By using a slightly different version, that is using a dictionary (`AFINN-111.txt`) with ranked words basing on their negativity or positivity[9], the reliability slightly decreases (see Algorithm C.12 on page 161).

At this point, we could use the *k-nearest neighbor algorithm* to train the corpus, and ignoring the vocabulary set. In this case, the accuracy increases from 62% to 76%, as showed in Figure 8.1.

With this other implementation, where we remove those data columns that don't appear in the `AFINN-96.txt` weighted dictionary, obtaining as a best result an accuracy of 82.7%. If we change the dictionary with the `AFINN-111.txt`, a slight and negligible variation on the accuracy is acheived (82.83%. See Figure 8.2 on the next page).

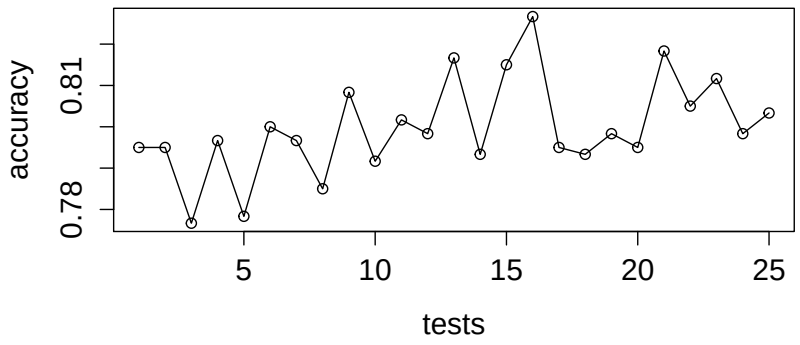
8.3.2 *Data Mining "in the large" (or, Training the algorithm with Twitter datasets)*

In real case data, we want to detect the following situations:

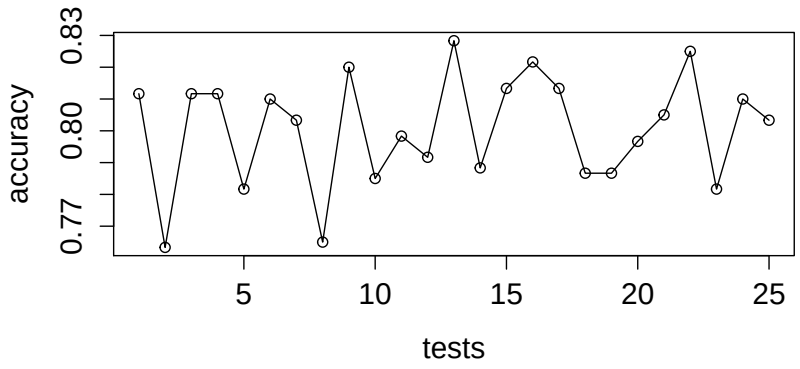
7 The data used is the one provided for movie reviews downloadable at `http://www.cs.cornell.edu/people/pabo/movie-review-data`.

8 `http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar`

9 `http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010`. This Url provides also the `AFINN-96.txt` database.



(a) Results with AFINN-96 dictionary



(b) Results with AFINN-111 dictionary

Figure 8.2: *Accuracy testing over Algorithm C.14 on page 165 with AFINN-96 dictionary and AFINN-111.*

- ◇ We should check if the filtered data becomes a significant training set or not (e. g. ig the resulting data set is formed of all positive or negative valuations)
- ◇ We should detect not also positive or negative thoughts, but also posts

9

SOCIAL NETWORK FORENSICS

Contents

At this point, we could also provide another case study where this relational algebra over hypergraphs could be useful. In particular, we could be interested to analize Social Network data in order to retreive some informations that could be used by investigators in order to solve (e. g.) a crime. In this chapter, we will hereby show how this information could be mined through the R language, by using some already existing libraries.

9.1 DATA LOCALIZATION

Given a generic Tweet database `db`, we could easily plot our data by using the following script, which produces the map showed in Figure 9.1 on the following page:

```
library(maps)                                          #load the libraries
library(mapdata)
svg(filename="svg_plot.svg",width=8,height=8,pointsize=12) #prepares a svg output
map("italy",fill=TRUE,col="green")                     #Italian geolocation
points(db$longitude,db$latitude,col="red")             #plot the points
dev.off()                                              #closes the svg device
```

At this point, it could be possible to cluster all the data using some clustering algorithms that are implemented as libraries for the R language. There could be many possible ways to cluster such data:

- ◇ Identifying regions (i. e. *geographical clusters*) by criteria of proximity.
- ◇ Identifying regions for k clusters.

At this point, it could be more useful to retreive data by invoking a simple query, inasmuch as we could potentially not know how this data is distributed over the space.

Identifying regions by proximity could be used to collect all the servicies that could be provided in a region of radius θ from an user p current location expressed in geographical coordinates $(lat, long)$.

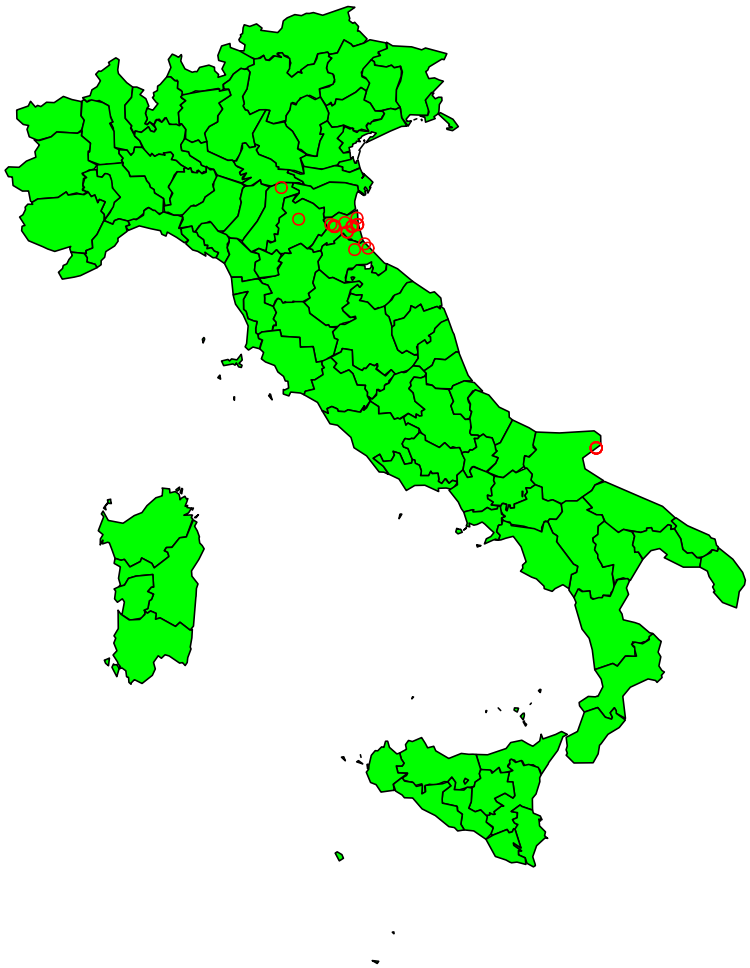


Figure 9.1: *Plotting Twitter geolocations drawing on tweets' geotagging.*

Since the `dbscan` algorithm([Zha13; TSK05]) uses also a μ parameter, which expresses that in a given area of θ radius there must be at least μ points, it would be more interesting to obtain even a small set $< \mu$ points: this request will trivially satisfy the query that an object u must be at most distance θ from p:

$$\sigma^{D}_{u \mapsto \rho(p,u) \leqslant \theta}(dhimp)$$

We could also update the original graph by operating:

$$\sigma^{D}_{u \mapsto \rho(p,u) \leqslant \theta}(dhimp) \bowtie \lceil himp$$

At this point, we would also like to obtain this cluster for a set of P users, in order to find their most θ-near points. In particular, we could see that we could express a more general form of this query in this way:

$$\rho^{T}_{[ulat \leftarrow lat, ulong \leftarrow long]}(\sigma_{e \mapsto \mathbb{T}(e)=\texttt{users}}(dhimp)) \bowtie_{p,u \mapsto \rho((p.ulat,p.ulong),(u.lat,u.long)) \leqslant \theta} \sigma_{e \mapsto \mathbb{T}(e)=\texttt{objects}}(dhimp)$$

This particolar query doesn't need any clustering algorithm, given that we don't need to discover new data, but to find the most proximate values to a given point.

At this point, we could be interested to retreive some data which could be retreived in a given geographical position, ad that could be posted until a given time. For that reason, we could be interested to specify the following arguments with the `searchTwitter` function already discussed on Section 8.2 on page 100:

- ◇ set the `since` and `until` parameters, in order to give a specific interval time;
- ◇ set the `geocode` parameter to choose only the tweets sended in a given region. In particular, as given by the [Gen13] guide:

  > *For the `geocode` argument, the values are given in the format 'latitude,longitude,radius', where the radius can have either `mi` (miles) or `km` (kilometers) as a unit. For example `geocode='37.781157,-122.39720,1mi'`.*

For example, we could want to retreive all the tweets produced from the center of Bologna in a radius of 20km:

```
t <- searchTwitter("",geocode="44.49428,11.34677,20km")
```

Or even some genuine tweets about Ferrara's *Salama da Sugo*:

```
t <- searchTwitter("salama",geocode="44.83802,11.61948,20km")
```

Part III

EPILOGUE

10

POSTFACE

The main aim of this thesis was to provide for the first time a systematic definition of hypergraph algebirc operators, that include querying and data modification operations. There is not a standard definition of those operators, and hence we've tried do give an implementation that is similar to the classical relational operators definitions. At this point we could consider the need of implementing all the datamining operations defined in [Cal+06], and then eventually evaluate the expressive power of our language. It could be also interesting to analyse the time complexity of those operations and to check how such operations could be parallelized.

Drawing on a pre-existing algebra for datamining, we've also firstly introduced an algebra for datamining over hypergraphs, by providing a partial implementation over this new kind of data structure, that is the $DHImp$. We could observe that the definition of such hypergraphs as a couple formed by a database and a set of matrices, makes possible in the near future to implement such structure by using some preexistente technologies (standard relational databases and matrix algorithms) and makes many graph problems embarrassingly parallel.

Although R provides a lot of statistical libraries for data mining and Java provides a system to allow parallel computations via MapReduce (see Appendix A on page 115), we think it is more useful to develop a system that could provide both statistical utilities and parallel computations in a same environment.

Since we have the aim to implement the hypergraph data structure with a framework for querying data, we should also evaluate which is the best language over which develop our future code; functional programming languages allow a more simple implementation of relational operations and queries but are harder development environments: sometimes there is a poor variety of libraries from which to choose the best one. Some functional libraries by the way provide distributed facilities and implement the MapReduce programming model[1].

A possible future expansion of this thesis could be the extension of the tensors for binary relations to tensors for general relations: this has the consequence that we will have to distinguish the square matrices for binary relations from the rectangular ones that express correlations between more than two entities, and that we have to represent via matrices the head and the tail of such relations.

Another significant improvement could be given by the review of the database indexing process: the dovetailing functions could be very useful if we want to keep the information

1 http://plasma.camlcity.org/plasma/index.html

on which data the previous computations were made but, on the other hand, they are not satisfactory for they require too much space after a significant number of computations.

It would also be interesting to see if the result of data mining with relational data differ from the ones obtained by mining the latent correlations that could exist inside the data.

A

R AND HADOOP: AN OVERVIEW

Nowadays MapReduce has become very popular since database queries, relational operations such joins [LD10] or even graph problems [Rod12] could be easily parallelized with this paradigm. In order to solve those problems, Yahoo! has developed with OpenSource licence Google's distribution computation paradigm and HDFS within the Hadoop environment.

R is a statistical tool that it is very supported from the open source community. It has become frequently used in data mining [Zha13; Sam+13] because absence of automatic reasoning tools requires the use of such statistical techniques to measure on the goodness of the data.

By the way Hadoop has been developed in Java and hence we need to make interact the R language, in which most of the statistical analysis is performed, with Java. The aim of this report is to describe which possible adapter architectures could be realized in order to make two systems developed in such different languages interact.

A.1 DATA MINING'S PURPOSES

Nowadays "everything is data": this means that everything is kept in store with the purpose to analyse it later and to discover interesting facts that couldn't be promptly inferred from the raw data; this knowledge discovering process is also driven by the increasing computerisation, that makes possible to collect huge amounts of data. All this data is meant to be analysed in short times, and hence it is needed to parallelize all such computations over huge amount of data:

> «*Powerful and versatile tools are badly needed to automatically uncover valuable information from the tremendous amounts of data and to transform such data into organised knowledge*»[HKP12]

The most important problem of data mining is that all the data must be analysed in a very short time via a performing distrbuted computation, and to obtain meaningful data via accurate statistical techniques.

A.1.1 *R*

R is a free software scripting language for statistical data manipulation and analysis and graphical plotting [Mat11], that is supported in many operative systems and hence it provides the possibility of defining scripts that are cross-platform. In some cases it could be also useful to interface our R script with C or C++ in order to define R wrapping over existing libraries: this is possible since R is designed to call native code functions and because the `Rcpp` library provides an interesting way to make a prompt wrapping between C++ and R data.

By the way R was designed before this "Big Data revolution" [MW11] and hence without any native support for large-scale data analysis and high-performance computing: since R was designed to be used only for statistical purposes, this language was designed to be single-threaded and memory bound, as far as all the data have to be loaded in main memory to be handled.

The interaction of R with the Hadoop system is seen as a desired utility as far as R is used not only by computer scientists, but also by statistician or other users that haven't got much confidence with programming languages. An adapter between the two systems could be very useful in order to use Hadoop without explicitly starting processes via Hadoop direct invocation. In this way we could start the whole distributed computation via R code, and hence automate the whole remote process. These adapters are actually achievable since R permits to develop add-ons and libraries.

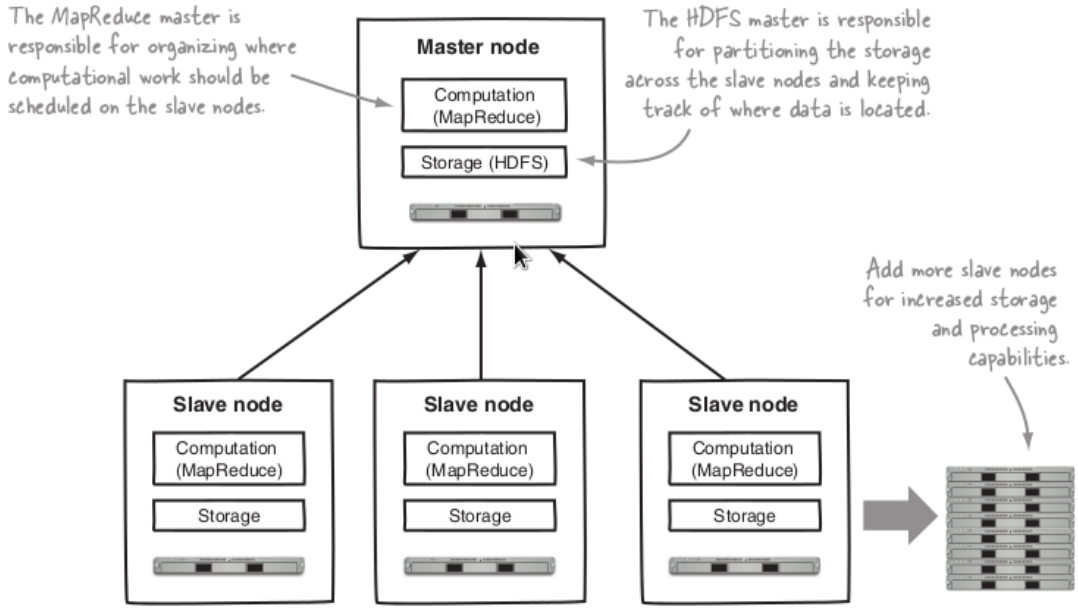
Since R is multiplatform, we need to find a solution that has to be platform independent (some other solutions like `multicore` are not supported in Windows) and it should be easy to make the master node (the one that starts the computation) dialogue with the all workers that need to be orchestrated. Moreover, we would like to analyse data as a whole, and not to discover local properties that often don't match with more general ones that could be retrieved if we check all the data.

A.1.2 *Hadoop*

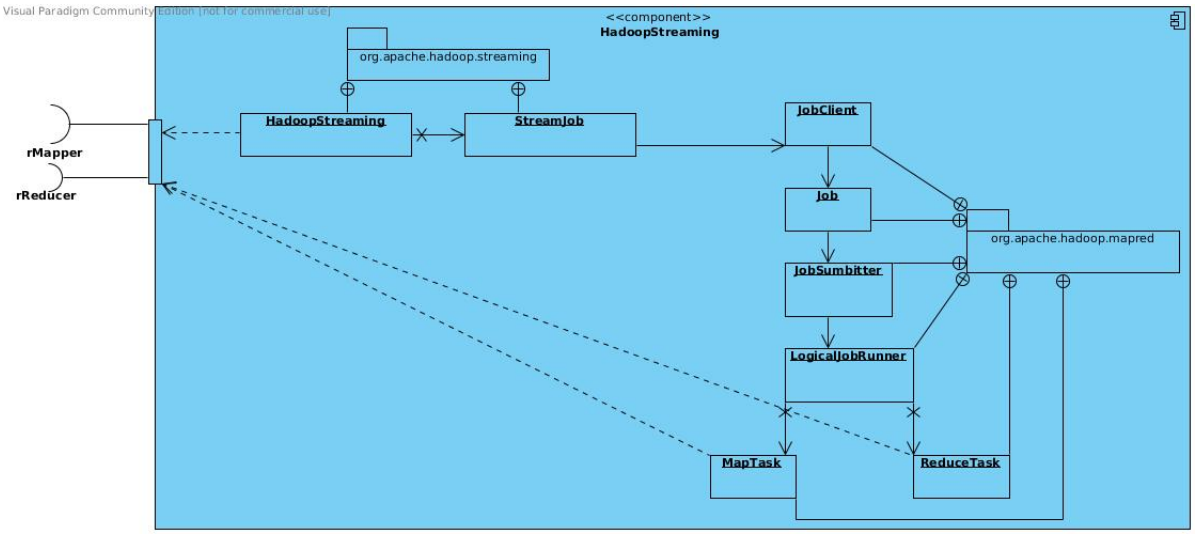
Hadoop (Figure A.1a) is a framework [MW11] which combines the benefits of the HDFS, a free-software implementation of the Google File System (a distributed file system optimized for high throughput and especially designed for managing files that are large for any file-systems commonly available - Figure A.2) and the MapReduce computation framework described by Google, which has the aim of realising a distributed computation by data decomposition in **keys** and **values** and their submission to so-called *map* and *reduce* workers.

> «*Hadoop [...] stores and provides computational capabilities over substantial amounts of data. It's a distributed system made up of a distributed filesystem and it offers a way to parallelize and execute programs on a cluster of machines*» [Hol12]

In this report we'll focus on the Hadoop Streaming 2.4.0 implementation of the MapReduce paradigm, since it is the most recent stable version available. HadoopStreaming (Figure A.1b)



(a) *Hadoop High-Level architecture.* [Hol12]



(b) *HadoopStreaming Component View.*

Figure A.1: Hadoop 2.4.0 Architecure.

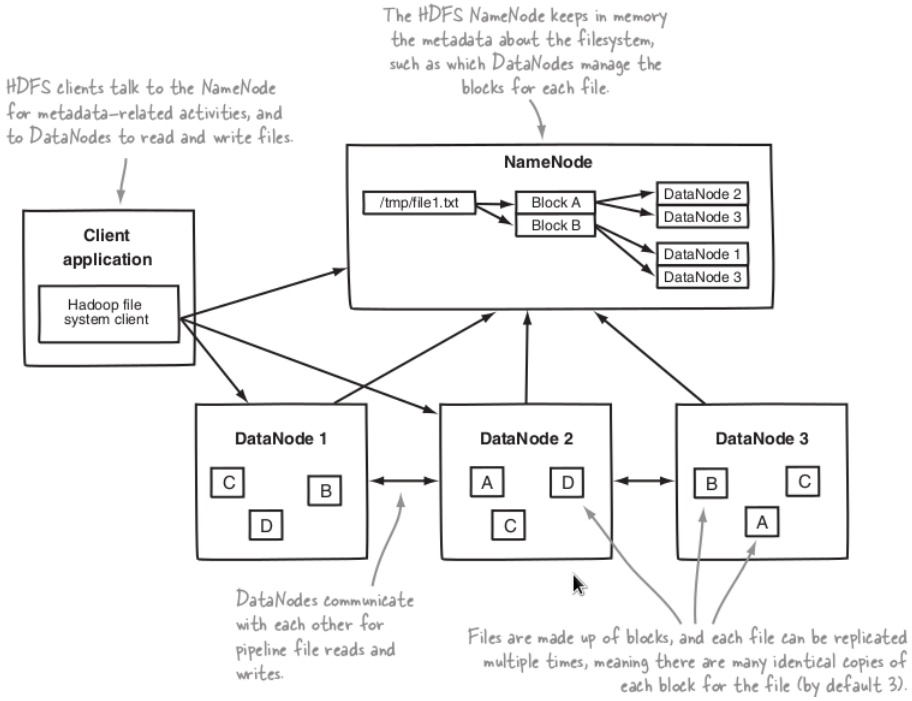


Figure A.2: *Hadoop 0.20.0 HDFS Process View.*[Hol12]

is a **Batch Method** [1] which accepts as parameters the Java classes or generic scripts to be used for both map (`rMapper`) or reduce (`rReducer`) computation via system's command-line arguments (`args`) or method invocation. This configuration prepares the JobClient [Whi09] which submits the `Job` to the `Cluster` class (not showed in the Component Diagram): each slave node with its `LocalJobRunner` will execute the *MapTask* or the *ReduceTask*. From now on the mapper and the reducers will obtain the partial computations via HDFS where all the temporary data is stored (Figure A.4c on page 123), since we suppose that different slaves don't share a primary memory, and hence secondary memory has to be used for backup. More details on how this computation is carried out from its subcomponents will be provided on Section A.3.1 on page 122.

A.2 ARCHITECTURAL TARGETS

*In this section, we'll analyze which is the **architectural context** and which functional and non functional **relevant properties** have to be kept in mind for evaluate the architectural design. From now on we'll call the libraries that permit R's statistical code to interact with Hadoop **adapter solutions**.*

1 See [BHS07] for more details on Pattern Oriented Software Architectures specifications.

A.2.1 *Architectural Context*

In order to describe in which context the adapter solutions are dipped, we've chosen to follow the ISO/IEC/IEEE 42010:2011 standard [ISO11], and hence to describe through the following essential components:

STAKEHOLDERS: The only stakeholder which we'll consider in our generic environment is the user working with R or/and Hadoop at any abstraction level, since we want to focus more on the benefits of the software architecture in its implementation than on the outcome of the data-analysis. We could detect three possible kind of such users:

- The *systems analyst*, which has set only the Hadoop architecture and eventually prepared the interaction of Hadoop software environment with R. This character could be noticed only on big systems or in corporate contexts, or could be assimilated on the developer of the whole adapter solution.
- The *backend user* is the programmer which is not aware of the Hadoop system, and hence has to use the R system without any knowledge of how the interaction between clusters and R is possible. This figure is also very probable on both corporate contexts or accademic ones, as far as we would like that even people that haven't studied computer science (e. g. statisticians) should be able to install and use such libraries via R.
- The *expert user* sums up in a single individual the two previously defined characters. He/she could be found only on small systems, where the Hadoop environment corresponds to a small home cluster.

We choose to describe the architecture from both the *backend user*'s point of view, since we're interested in how the architecture it reflects on the R script code design, and from the *systems analyst*'s, since we want to check how the R code interacts with the Hadoop system. An overall view of the system could be perceived by the *expert user*.

If we want to analyse how our solution behaves in a business environment, we could also mention the *data architect*, which is a backend user with some **systems analyst** skills that defines how the data - that will be analysed - should be stored in order to ease the work of the so-called *data scientist*. Another possible stakeholder is the *business analyst* which could define the company mission through the results that data have been previously carried out.

MISSION AND CONCERNS: The mission of each *adapter solution* is to accomplish distributed computations using the HDFS and the MapReduce paradigm over big quantities of data and to eventually store the resulting computation in a distributed file system. Some more missions and concerns will be described within the Architectural Properties subsection (subsection A.2.2 on page 121) using some stakeholders' point of view.

COMPONENTS: Since a SYSTEM is made of components that exists only to fulfil one or more missions, these could be divided in hardware components and software components: the hardware components will define the infrastructure over which the Hadoop environment

is set up, while the software ones are formed by the Hadoop software framework itself, the R system, the packages for statistical analysis some adapter solutions and the data mining code that interacts with them.

ENVIRONMENT: It is defined as *the set of the stakeholders, the system's components and the external ones with which the system interacts.* In this case an external component could be seen as a remote server from which the pieces of information are retrieved and given to the map-reduce system to manipulate; this could be a possible scenario if the data we want to analyse is not stored on the Hadoop HDFS or if a remote application accesses our analysed data (Figure A.3 on the facing page).

A.2.1.1 *Possible Scenarios*

The analysis of the possible scenarios permits to check how the stakeholders interact with the system, and hence it could be useful to define their possible requirements.

[Ana11] provides some real-case scenarios that imply the use of R with an Hadoop infrastructure describing which are the possible applications where such interaction is required and useful.

LIFE SCIENCES: In a human genomic study context, we could have datasets of 25GB using 3 million variables over 3000 subjects and splitted over 22 files. This model could become more computationally intensive if we want to analyse the genoma of all the people that are visited inside an hospital. This measure could be useful if (e. g.) we want to prevent to hospitalize more people than necessary: in this case the features "storing large amount of data" and "speed and accuracy of data analysis" are important to make diagnosis succeed.

This scenario points out how such information retrieval models should be usable also for doctors that don't have a direct experience of computer science.

NETWORK ANALYSIS: The R packages `igraph` and `statnet` provide some useful features for community detection and statistical analysis over networks: in this case we would like to recreate the networks from data (e. g. from microblogging platform as twitter) that will create semantic network of words and network of users.

Given that the graph properties could be checked via algorithms that are based upon the MapReduce framework [Kat12; LD10], we would like to perform via parallel computations our code by using a distributed version of the former sequential algorithm.

Since R supports many other libraries that permit statistical plotting (`ggplot2`) and that some R code may have already been developed in the past using such libraries, it could very difficult to port all the existing code and libraries to Java.

REUSE AND KNOWLEDGE SHARING: Many nowadays tools make sharing and duplicating data analysis very difficult to perform: R could be used as a universal language for sharing statistical analysis over the data (e. g.) via DeployR, which shares the computing environment in a PLATFORM AS A SERVICE and makes the analysis as realtime as possible (see Figure A.3). This system is more versatile than standard Microsoft Excel [Ana11], as

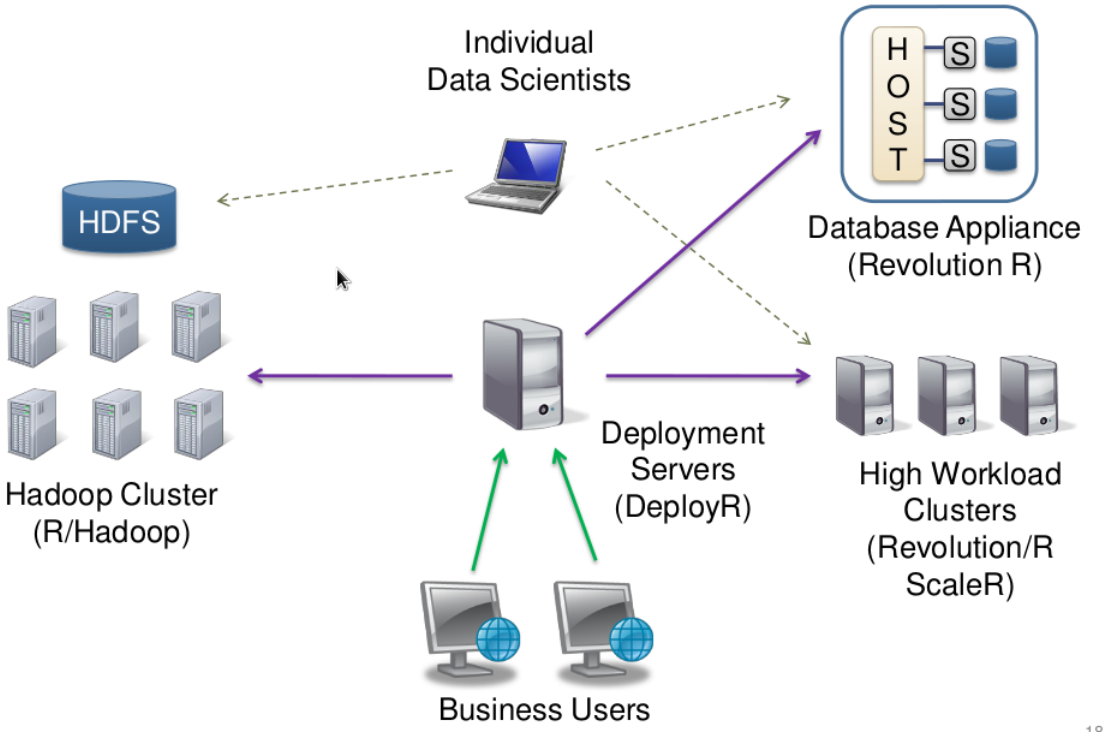


Figure A.3: *Example of an infrastructure for data analysis sharing.* [Ana11]

in R we could already manipulate data and plot result without any direct interaction but via batch computations and procedural computing languages.

A.2.2 *Architectural Properties*

The main requirement is that the R code should directly interact with the Hadoop MapReduce framework. We have now to define different non-functional requirements in order to make this interaction as "natural" and fluid as possible: let's analyse the properties from an *backend user* point of view; since we have a lot of data to analyse we would require that the **speed** of the computation can be completed in a reasonable time, and hence we have that a bad design could potentially slow our computations and the overall performance.

The interaction between R and Hadoop should be as **simple** as possible, that is we would like to develop a code that is easily readable and easy to maintain, and we would like that the R code could be developed without knowing the whole Hadoop infrastructure, in order to focus more on the analysis and less on implementative details. Another concern of simplicity is the *client-side integration*, by which we aspire to use our code on R IDEs such as RStudio.

From a *systems analyst* point of view we would like to solve some **capacity** issues, since Big Data by definition requires lots of data to be stored: we would like to have enough nodes to

store all our data, and to be stored in a way that could be easy to manage. These and other conditions (like some security issues) are all naively solved by the HDFS implementations, and hence doesn't directly affect the requirements of the Adapter design, even if we have to keep in mind that a lot of data could be possibly shared among different users.

We would also like to have a system that is easy to install, so that this character could focus more on the proper functioning of the cluster: those requirements don't influence directly the architecture structure, but could be a primary goal for optimising any business process.

A.3 ADAPTER SOLUTIONS' ARCHITECTURAL ANALYSIS

In this section we'll finally analyse the features of each adapter solution, and hence detect which we could assume to be the best for industrial or smaller environments.

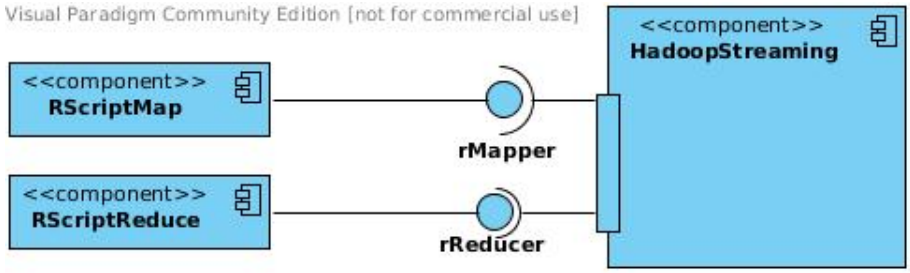
Since Hadoop, as said before, is developed in Java and hence it could be simply unfeasible to write all the existing and the user's R packages into Java, we have to implement an adapter between these two systems, and hence to detect how is this possible. We'll show for each proposed architecture how R will interact with the Java code.

In the following analysis we by the way assume that Hadoop is secure. In this way we could focus more on the possible security flaws that could happen in its interaction with the R script when it submits the jobs to the system.

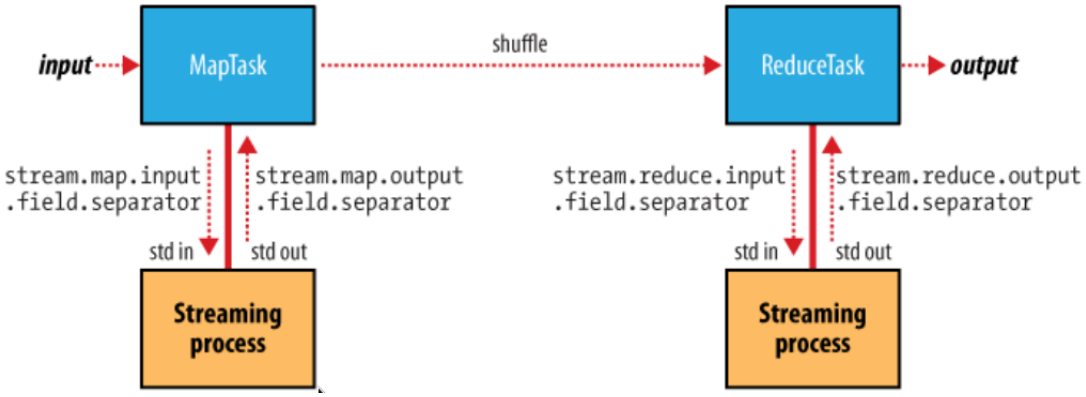
A.3.1 *R+Streaming*

STRUCTURE: The Hadoop components (Figure A.4a) that are used in both MapTask and ReduceTask (Figure A.4b) are the following [Pra13]:

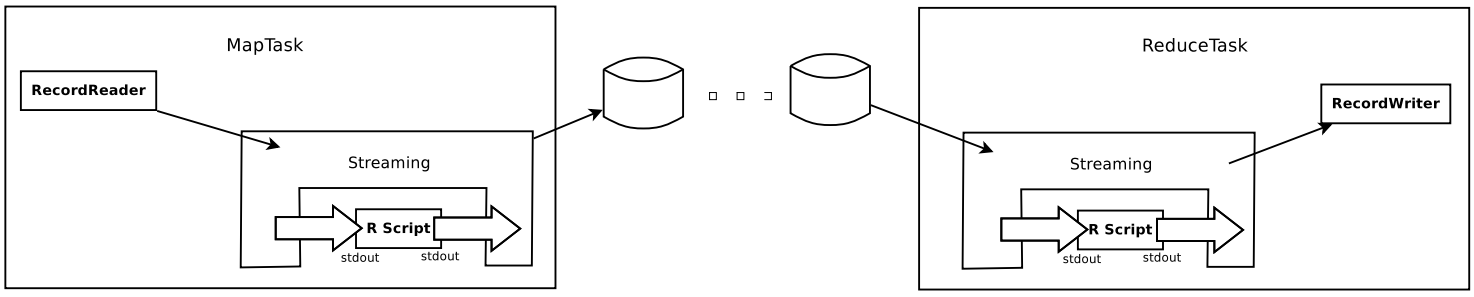
- `R Script`: two R scripts, one on the map side and the other on the reduce side, will be executed by the map and the reduce task that interact with the `Streaming` component via `stdin` and `stdout` connectors in order to receive the data that has to be queried or manipulated.
- `MapTask`: «*starts MapReduce operations by carrying input files and splitting them into several pieces. For each piece, it will emit a key-value data pair as the output value*».
- `RecordReader`: Given a choosen InputFormat which specifies how to transform data into $\langle key, value \rangle$, it acheives the iteration over such data that will be used in the mapper job (that is here represented by the `Streaming` component).
- `Streaming`: This class [Whi09] outputs keys and values (including the map keys and values) as text and, only in this case, it passes the data to a non Java program (e. g. R) via `stdin`. Afterwards it collects the data that the script emits in `stdout`.
- `Reducer`: «*it accepts key-based grouped data from the Mapper output, reduces it by aggregation logic, and emits the* $\langle key, value \rangle$ *pair for the group of values*».
- `RecordWriter`: this class will finally store the computed data with the expected text format as a consequence of the completion of the whole MapReduce process.



(a) *R and Streaming Component View.*



(b) *R and Streaming High-Level Data-Flow.* [Whi09]



(c) *R and Streaming Object-Level Data-Flow.*

Figure A.4: R+Streaming Structure

```
#! /usr/bin/env Rscript
options(warn=-1)
sink("/dev/null")
input <- file("stdin", "r")

while(length(currentLine <- readLines(input, n=1, warn=FALSE)) > 0) {

   fields <- unlist(strsplit(currentLine, ","))

   lowHigh <- c(as.double(fields[3]), as.double(fields[6]))

   stock_mean <- mean(lowHigh)

   sink()

   cat(fields[1], fields[2], stock_mean, "\n", sep="\t")

   sink("/dev/null")

}
close(input)
```

Algorithm A.1 R+Streaming Mapper code

BEHAVIOUR: In the previous description we've already depicted each component's behaviour. Let's take a look once again at Figure A.4: we could see that the first mapper job reads the data, chunks it into $\langle key, value \rangle$ and then passes it to the R script that reads those tuples as strings. After splitting and chunking the data, the R scrpit outputs the filtered (Map, Algorithm A.1) or summed-up data (Reduce, Algorithm A.2) and then process again the handled values. Both kind of tasks are orchestrated by the `Job` itself, as already depicted in the Hadoop Section.

FUNCTIONS: Even if this solution has a very low computational cost, since the Hadoop system is called directly with the following command, it is not very simple for a *backend user* to start the computation:

```
${HADOOP_HOME}/bin/hadoop \
  jar {$HADOOP_HOME}/contrib/streaming/*.jar \
  -inputformat org.apache.hadoop.mapred.TextInputFormat \
  -input inputDB.csv \
  -output outputDB.csv \
  -mapper my/R/mapScript.R \
  -reducer my/R/redScript.R
```

This implementation makes also the non-Hadoop based code less reusable, as it has to be chunked in two single algorithms for mapping and reducing the $\langle key, value \rangle$ couples. Moreover in this way we would have no *client-side integration* since the whole computa-

```r
#! /usr/bin/env Rscript
options(warn=-1)
sink("/dev/null")

outputMean <- function(stock, means) {
  stock_mean <- mean(means)
  sink()
  cat(stock, stock_mean, "\n", sep="\t")
  sink("/dev/null")
}

input <- file("stdin", "r")

prevKey <- ""
means <- numeric(0)

while(length(currentLine <- readLines(input, n=1, warn=FALSE)) > 0) {

  fields <- unlist(strsplit(currentLine, "\t"))

  key <- fields[1]
  mean <- as.double(fields[3])

  if( identical(prevKey, "") || identical(prevKey, key)) {
    prevKey <- key
    means <- c(means, mean)
  } else {
    outputMean(prevKey, means)
    prevKey <- key
    means <- c(means, mean)
  }
}

if(!identical(prevKey, "")) {
  outputMean(prevKey, means)
}

close(input)
```

Algorithm A.2 R+Streaming Reducer code

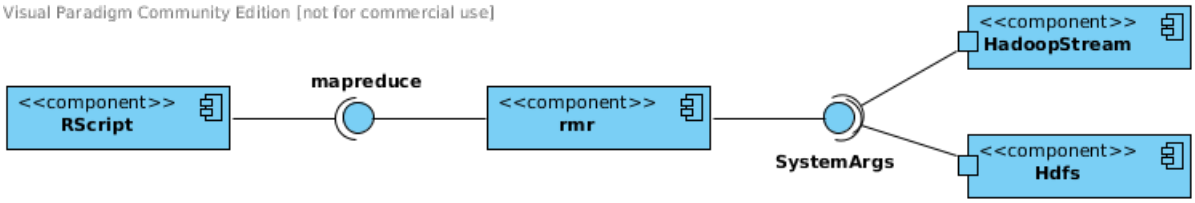


Figure A.5: *RHadoop Component Diagram.*

tion doesn't start in the R environment, but uses such language only to implement the tasks.

On the other hand the system is simple to install, since it is only necessary to have an Hadoop and R support.

There are no security flaws since there is no adapter solution, and hence all the computation is direclty invoked by the user. As a direct consequence this approach computes very quickly (there is no intermediate layout to initialize).

The produced code is not even very easy to develop, as we have always to parse the input values from `stdin` and split them into keys and values and return all the output values as Hadoop requires without using the R native datatypes.

All those considerations make the described solution OS independent.

RATIONAL: This solution couldn't be considered as an adapter between R and the Hadoop system, since Hadoop itself permits to interact with R code via a bash command; as a direct consequence we only globally observe a a **Batch Method** pattern (Figure A.4a on page 123), inside which the computation is carried out with a simple **Pipes and Filter** architecture (Figure A.4c on page 123), where the pipes are the `stdin` and `stdout` of the R script, and the filters are the R scripts.

TESTING: A possible simple testing is depicted in [Hol12], where it is showed how the execution of the R script could be tested with GNU/Linux pipe commands:

```
cat database.csv | mapScript.R | sort --key 1,1 | redScript.R
```

Since in the following solutions we'll show scripts equivalent to the both proposed above, we could perform some testing by comparing the output produced from this solution to the one produced by the following ones, in order to check if the expected results are provided.

```
library(rmr)

map <- function(k,v) {
  fields <- unlist(strsplit(v, ","))
  keyval(fields[1], mean(as.double(c(fields[3], fields[6]))))
}

reduce <- function(k,vv) {
  keyval(k, mean(as.numeric(unlist(vv))))
}

kvtextoutputformat = function(k,v) paste(c(k,v, "\n"), collapse = "\t")

mapreduce(
  input = "stocks.txt",
  output = "output",
  textinputformat = rawtextinputformat,
  textoutputformat = kvtextoutputformat,
  map = map,
  reduce = reduce)
```

Algorithm A.3 RHadoop MapReduce implementation

A.3.2 *RHadoop (rmr2)*

RHadoop is «*a collection of four R packages that allow users to manage and analyze data with Hadoop*»[2] developed by RevolutionAnalytics. We'll focus on `rmr2` which interacts with the Hadoop's MapReduce paradigm's implementation.

STRUCTURE: Let's look at Figure A.5: in this case we could use a simple R script (`RScript`) which implements both mapper and reducer (Algorithm A.3), which are the only two external components that are needed to `HadoopStream` to carry out the computation. After this interaction, the MapReduce computation will be performed as depicted before for the R+Streaming solution.

Since `mapreduce` is a function provided by `rmr2` that starts the distributed computation, accordingly to [Preo8] it could be seen as an interface, and hence as a connector.

The BEHAVIOUR description will now explain why we choose to represent the system's `args` as a communication interface between `rmr` and the back-end component `HadoopStreaming` instead of using `rMapper` and `rReducer`.

BEHAVIOUR: A quick analysis of the `rmr2` source code showed how it solely provides a wrapping over the command line arguments by which configure the whole Hadoop: some native functions are only provided for data conversion reasons (Hadoop MapReduce for-

2 https://github.com/RevolutionAnalytics/RHadoop/wiki

mat into standard R data types), while the provided Java classes are directly used by `HadoopStreaming` for checking out which class to use to parse the input and output stream data. Except for this wrapping features, this solution doesn't show substantially different behaviours from the one depicted in the former solution.

FUNCTIONS: The data conversion from `HadoopStreaming` format to R mapper/reducers and vice versa, jointly with the `args` wrapping functions, are the only and negligible computational costs added to the basic Hadoop MapReduce computation. Even if it is required to only install the `rmr2` package we have to perform this operation on each node that starts and carries out the `Job` computation [Ana07].

The use of System environment's variables for finding and executing the Hadoop's binaries and *jar*s creates a huge security flaw, since it is very simple to change the environment variables in order to perform a MITM attack, or to deliberately exclude the Hadoop system from the whole computation. This security issue could be easily resolved if the R code was designed to invoke Java methods directly, as it will be showed in the following solution.

Given that this Adapter permits to create a single script where to implement both mapper and reducer without initialising or starting the Hadoop cluster directly via command line, we could see how this solution provides a *client-side integration* and provides a whole "functional unit" for encase the whole MapReduce computation specification.

This library requires a *simple* interface for map and reduce arguments, as data is already mapped to keys, values and list of values by a perliminar parsing procedure carried out by the library.

Even in this case we have that `rmr` is trivially fully supported on all the OS.

RATIONAL: In this case the **Batch Method** pattern is needed on the interaction between `rmr2` and `HadoopStream`, where our library is used as a simple **Adapter**.

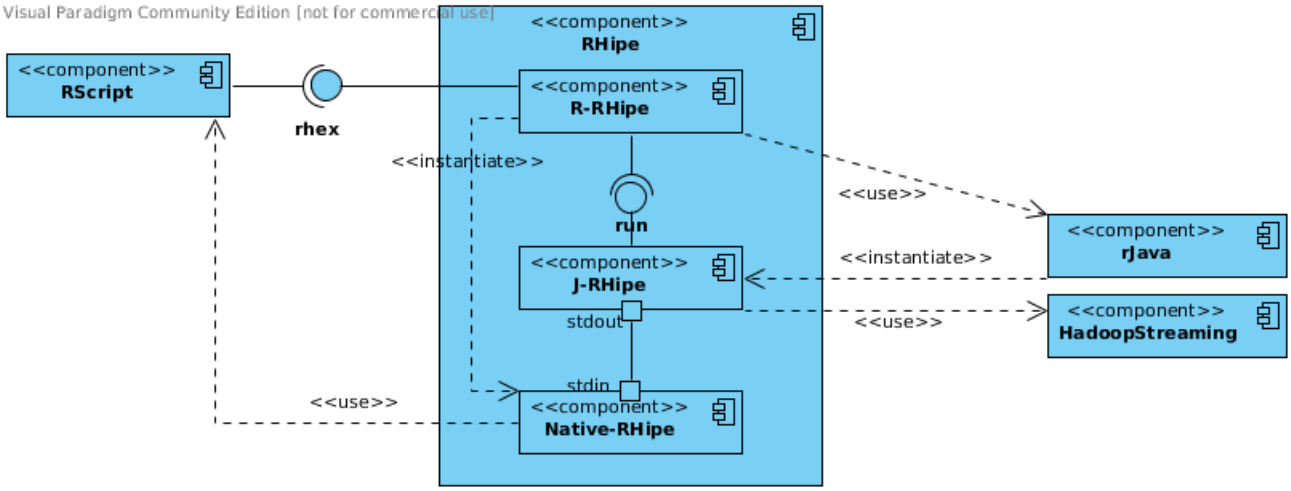
TESTING: A simple test to check to verify the library is working correctly could be carried out using the Algorithm A.3 on the previous page.

If we want to test the sole `rmr2` library, we could think of print as terminal output each command that should be invoked as a `system` syscall: in this way we'll check if the provided arguments are correct.

A.3.3 *Rhipe*

Rhipe is an acronym for R AND HADOOP INTEGRATED PROCESSING ENVIRONMENT; this library provides a more well-designed integration between Java and R code, since it takes full advantage of the possible integration between different languages. Figure A.6a on the facing page shows how these interactions are possible: since R can directly invoke native methods and given that JNI is a Java framework that permits to invoke Java methods from native functions and vice versa, we could invoke Java methods directly from an R environment via native

(a) *Layer view on the interaction between R and Java code.*



(b) *Component Diagram of showing the interaction between R code and the Hadoop system via Rhipe library.*

Figure A.6: Rhipe architectural overview.

code. This mechanism is provided by the `rJava` R library, which is hereby used by Rhipe. Similarly to Java, R provides some utilities for invoking R code from native binaries, and hence it could be also possible to perform the opposite invocation, that is invoking R code from a Java environment.

STRUCTURE: Figure A.6b shows that Rhipe is formed by three components, that are situated in different language environments: `R-Rhipe` is the R language part, `J-Rhipe` is the Java implementation and `Native-Rhipe` is their native counterpart.

Analogously to the former solutions, a single R script (`RScript`) could provide both map and reduce task implementations (Algorithm A.4 on the next page), and moreover even in this case Rhipe provides a `rhex` function via which `R-Rhipe` could start the computa-

```
#! /usr/bin/env Rscript
library(Rhipe)

rhinit(TRUE,TRUE)

map <- expression({
  process_line <- function(currentLine) {
    fields <- unlist(strsplit(currentLine, ","))
    lowHigh <- c(as.double(fields[3]), as.double(fields[6]))
    rhcollect(fields[1], toString(mean(lowHigh)))
  }
  lapply(map.values, process_line)
})

reduce <- expression(
  pre = {
    means <- numeric(0)
  },
  reduce = {
    means <- c(means, as.numeric(unlist(reduce.values)))
  },
  post = {
    rhcollect(reduce.key, toString(mean(means)))
  }
)

input_file <- "/tmp/stocks.txt"
output_dir <- "/tmp/output"

job <- rhmr(
  jobname = "Rhipe CMA",
  map = map,
  reduce = reduce,
  ifolder = input_file,
  ofolder = output_dir,
  inout = c("text", "sequence")
)

rhex(job)
```

Algorithm A.4 Rhipe MapReduce implementation

tion. `R-Rhipe` also initializes `J-Rhipe` via `rJava`, that will later direclty interact with the Hadoop's Java classes in `HadoopStreaming`, and `Native-Rhipe` which will provide the environment for running the R map and reduce computations. The whole computation is started via a `run`-like function.

The mechanism that permits `J-Rhipe` to interact with `Native-Rhipe` is provided by Google's ProtocolBuffer connector, by which we change the standard `stdin` and `stdout` that are used to interact with R scripts.

BEHAVIOUR: Figure A.7 on page 133 deepens the interaction between the Java and Native components which carry out the MapReduce computation orchestrated by `HadoopStreaming`. We could see how a single binary, that is executed in each slave node, could handle all the key-value elements sequentially and hereby handle each request one at a time.

FUNCTIONS: Concerning the computational time, `ProtocolBuffer` permits to quicken the data transfer between the Java orchestrator and the task implementations, since the library provided pipes are expressly designed to carry messages within a large data set:

> «*Protocol Buffers are great for handling individual messages within a large data set. Usually, large data sets are really just a collection of small pieces, where each small piece may be a structured piece of data. Even though Protocol Buffers cannot handle the entire set at once, using Protocol Buffers to encode each piece greatly simplifies your problem: now all you need is to handle a set of byte strings rather than a set of structures.*»[3]

Rhipe reduces the number of possible security flaws from the `rmr2`'s, since in this case only `Native-Rhipe` is directly called via `system` syscall. Even in this case we have a naïve *client-side integration*, even if the incoming MapTask implementation arguments have to be parsed similarly to how it is done in R+Hadoop and the all the map-reduce function interfaces are slightly more complicated than in RHadoop. Some demerits of this architecture are the inability to use it on MacOS and the requirement of installing Rhipe on each slave and client node [MW11].

RATIONAL: Beyond the usual **Pipe and Filter** and **Adapter** POSA4 architectural patterns, we could see that `rJava` acts as a **Builder** for Java objects and as a **Mediator**, since thanks to this library we could create R object that represent Java objects over which invoke the desired methods. `R-Rhipe` serves as a **Builder** for `Native-Rhipe` or also as an **Adapter** between the other Rhipe subcomponents developed in different languages, and hence running on different environments.

TESTING: As already done before, we could think of compare as a test the output of Algorithm A.4's computation with the one produced by the R+Streaming code (Algorithm A.1 on page 124 and A.2).

In this bridge solution is very hard to develop separate testings for each Rhipe subcomponent, since there is a very tight interdependence between those. By the way we could

3 https://developers.google.com/protocol-buffers/docs/techniques

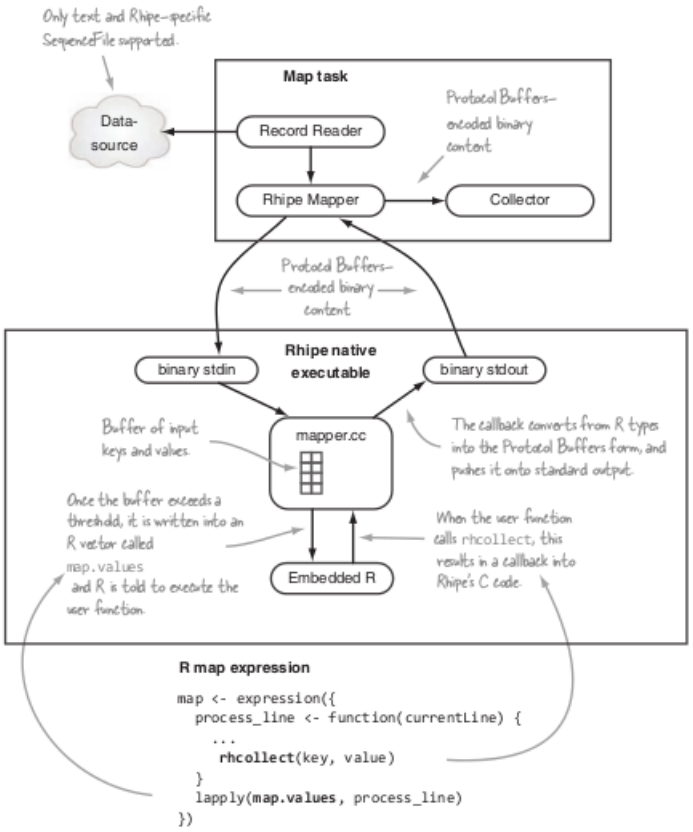
test the interaction between `J-Rhipe` and `Native-Rhipe` and hence check if the data that is passed through the `ProtocolBuffer` is always parsed and used with same semantics from both sides of the connector: since the messages that have to pass inside the connector have to be formatted as the ProtocolBuffer expects, we have to check if the data meaning is preserved in both encoding and decoding phases.

A.4 FINAL REMARKS

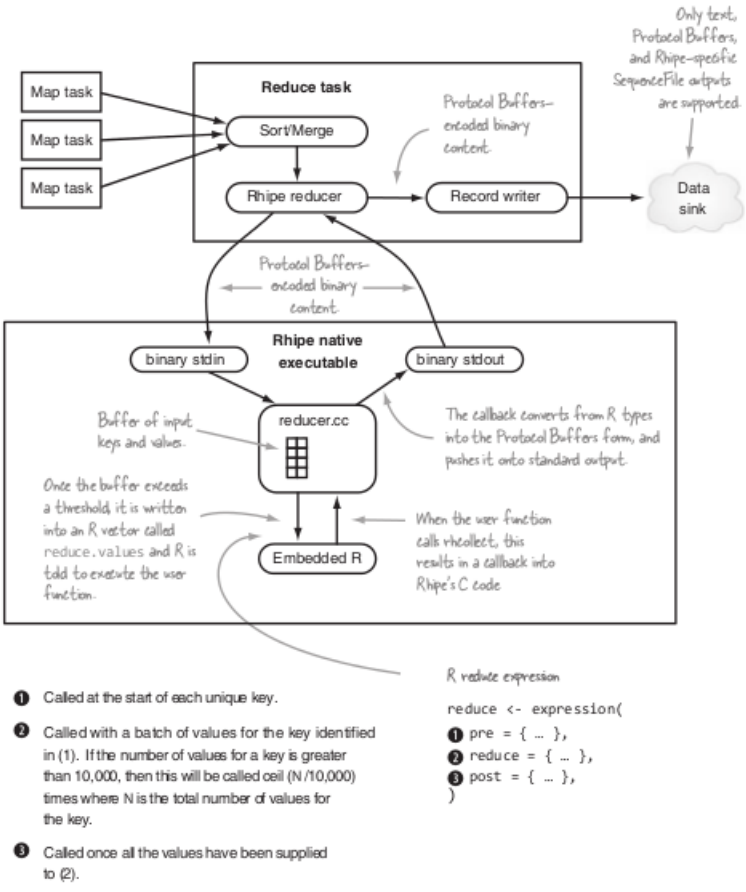
Given the previous architectural analysis, we could exclude from the desired solutions the R+Streaming one, since it doesn't provide client-side integration and because it doesn't facilitate R scripts maintainability and reuse, as they cannot be directly invoked by other R scripts and all the computation is carried out in batch mode. If MacOS support is not required, we could consider to use Rhipe since it is more secure, even if in this case we still have to parse the incoming data like in the R+Streaming approach.

Even if we would have to choose a library in order to extend for meeting our architectural requirements, we'll chose in any case the Rhipe library. If we want to avoid to pass th edata to `Native-Rhipe`, we could use the `JRI` library that permits to call R code from Java. In this way we don't need to use Google's non-free implementation of ProtocolBuffer, and we could keep all the slave tasks in the same process, as already done in standard HadoopStream support. These modifications could also quicken the computation of the map or reduce task, since we won't have the need to transfer data between different processes. We could also make the Rhipe R map-reduce interface more easy to handle by extending `R-Rhipe` similarly as already implemented in RHadoop. If we had choosen to modify RHadoop solution we had to re-implement the communication between R and Java, and hence we would have to rewrite most part of the library.

Since the only advantage of RHadoop is to provide a very simple function interfaces for R mappers and reducers, we finally state that the solution that best meets our expectations that is also best designed is Rhipe, since we prefer to have a more secure library than to have a more easy to use one.



(a) *Layer view on the interaction between R and Java code.*



(b) *Component Diagram of showing the interaction between R code and the Hadoop system via Rhipe library.*

Figure A.7: Rhipe Behaviour and interaction between Java and Native components. Note that in the current implementation `mapper.cc` and `reducer.cc` are replaced by `mapreduce.cc`. [Hol12]

B

IMPLEMENTING A SIMPLE TYPE SYSTEM IN R

In order to make the implementation of R code more easy to check and manage, we've introduced in R a simple type system, in order to check the coehernce of the data schema. It is our personal opinion that type safe systems could also quicken the developing of code.

This code was produced on the first attempt to implementation of the hypergraph mining algebra on R.

The first Listing provides how to define a definition of Type, that is a name with a property that data should satisfy. We have to reimplement the comparison operator as %==%, since it is not possible to do so for basic R types.

```
# I need to redefine the operator == since == couldn't be extended for generic lists
setGeneric("%==%", function (e1,e2) {
  e1==e2;
})


## This is the kind, au vrais the meta-type
Kind <- setRefClass(
  "Type",
  fields=list(name="character",prop="function"),
  methods = list(isType=function(x){
                   prop(x);
                 },
                 valid=function() {
                   #Autogen
                   if (!is.character(name)) {
                     return (FALSE);
                   }
                   if (!is.function(prop)) {
                     return (FALSE);
                   }
                   return (TRUE);
                 }
  )
)

# Unfortunate choice (comparing only by name)
setMethod("==", signature(e1 = "Type", e2 = "Type"), function (e1, e2) {
```

```
  e1$name == e2$name
})
setMethod("%==%", signature(e1 = "Type", e2 = "Type"), function (e1, e2) {
  e1 == e2
})

## Checks if it is a correctly defined Type
is.Kind <- function(t) {
  if (!((attr(t,"class")[[1]]=="Type"))) {
    return(FALSE);
  } else {
    return(t$valid());
  }
}

## Creates a new Type
new.Type <- function(name,test) {
  toret <- Kind$new(name=name,prop=test);
  if (toret$valid()) {
    return(toret);
  } else
    stop("Invalid Type Format")
}

########################################
# Checks if a type is in the current list
Type.in.list <- function (x, table) {
  for (i in table) {
    if (x %==% i) # comparing also lists
      return(TRUE);
  }
  return(FALSE);
}
setMethod("%in%", signature(x = "Type", table = "list"), Type.in.list)
########################################

## Defines a Subtype as a Type that has additional features from the base type
new.Subtype <- function(name,test,Type) {
  toret <- Kind$new(name=name, prop=function(x) { (Type$isType(x))&&(test(x)) });
  if (toret$valid()) {
    return (toret);
  } else
    stop("Invalid Sub-Type Format")
}
```

Algorithm B.1: Implementing Types

We could provide some examples of simple data types, that could be successively extended into our schema types.

```
hd <- function(l) {
  head(l,1)[[1]]
}

tl <- function(l) {
  tail(l,length(l)-1)
}

empty <- function(l) length(l)==0

# Comparing lists
setMethod("%==%", signature(e1 = "list", e2 = "list"), function (e1, e2) {
  if (length(e1)==length(e2)) {
    tmp2 <- e2
    for (elem in e1) {
      if (!elem %in% tmp2)
        return(FALSE)
      else
        tmp2 <- tl(tmp2)
    }
    return(TRUE)
  } else
    return(FALSE)
})

has_not_duplicated <- function(l) {
  if (length(l)<=1)
    return(TRUE)
  else {
    tmp <- tl(l);
    elem <- hd(l);
    if (elem %in% tmp)
      return (FALSE)
    else
      return (has_not_duplicated(tmp))
  }
}


## Defines Numeric DataTypes
Numeric <- new.Type("Numeric",is.numeric)

## Defines Characters
String <- new.Type("String",is.character)

## Defines Vector DataTypes
Vector <- new.Type("Vector",is.vector)

## Defines Lists
List <- new.Type("List",is.list)
```

```
## Define sets as lists of unique elements
Set <- new.Subtype("Set", has_not_duplicated ,List)

# Comparing sets or lists
setMethod("%==%", signature(e1 = "Value", e2 = "Value"), function (e1, e2) {
  if ((e1$typeName() == "List" && e2$typeName() == "List")||
        (e1$typeName() == "Set" && e2$typeName() == "Set")) {
    return(e1$get() %==% e2$get())
  } else
    return(FALSE)
})



## Defines a Typed List
ListX <- function(X) {
  ## Otherwise, i must declare different $in$ and == methods
  new.Type("List",function(l) {
    if (!is.list(l)){
      return(FALSE);
    } else
      return(prod(unlist(lapply(l,function(x) X$isType(x)))));
  })
}

ListString <- ListX(String)


setMethod("%in%", signature(x = "Type", table = "Type"), function (x, table) {
  if (table$type$name=="List" || table$type$name=="Set")
    Type.in.list(x,table$get())
  else
    stop("ERROR: the table is not a List nor a Set")
})
```

Algorithm B.2: Implementing Types

At this point, we could implement our data types: we could define a value as a R object where both values and types are casted. This could be useful for small data types but, in our case, we prefer to use the check system for small data types and implement them as the previously described R object for more relevant structures. We could define an object with a certain type with the `a%:% T` operator, where *a* is `a` standard R value and `T` is a type. with the `a%?% T` operator we could ask if a is of the correct type.

```
Value <- setRefClass(
  "Value",
  fields=list(typeof="Type",ival="ANY"),
  methods = list(valid=function() {
                   #Autogen
```

```
                  if (!is.Kind(typeof)) {
                    return (FALSE);
                  }
                  if (!typeof$isType(ival)) {
                    return (FALSE);
                  }
                  return (TRUE);
                },
                get=function() { ival; },
                set=function(v) {
                 if (!typeof$isType(v)) {
                   stop(paste(c(as.character(v),"has not type",typeof$name),collapse=" ",
                       sep=""));
                   return(NULL);
                 } else
                   ival <<- v;
                },
                type=function() { typeof; },
                typeName = function() { typeof$name; }
  )
)

## Comparing two elements
setMethod("==", signature(e1 = "Value", e2 = "Value"), function (e1, e2) e1$get() %==% e2$
    get() && e1$type() == e2$type())
setMethod("%==%", signature(e1 = "Value", e2 = "Value"), function (e1, e2) e1 == e2 )

flattenValueList <- function(x) {
  if (is.list(x)) {
    lapply(x,function(z) z$get())
  }
}

is.Value <- function(x) class(x)=="Value"

## This definition doesn't allow to define List<Value>
Value.in.list <- function (x, table) {
  for (i in table) {
    if (x %==% i)
      return(TRUE);
  }
  return(FALSE);
}

setMethod("%in%", signature(x = "Value", table = "list"), Value.in.list)

setMethod("%in%", signature(x = "Value", table = "Type"), function (x, table) {
  if (table$type$name=="List" || table$type$name=="Set")
    Value.in.list(x,table$get())
  else
    stop("ERROR: the table is not a List nor a Set")
```

```
})

## Binary Operator that creates an type-guarded object
"%:%" <- function(valu,type) {
  if ((!type$isType(valu))||(!is.Kind(type))) {
    if ((!type$isType(valu$get()))||(!is.Kind(type))) {
      stop(paste(c(as.character(valu),"has not type",type$name),collapse=" ",sep=""));
      return(NULL);
    } else
      return(Value$new(typeof=type,ival=valu$get()));
  } else
    return(Value$new(typeof=type,ival=valu));
}

"%?%" <- function(val,type) {
  type$isType(val$ival)
}
```

Algorithm B.3: Implementing values with Types

C

ALGORITHMS

C.1 HYPERGRAPHS IN R

In order to produce the hypergraph representation on Chapter 2, we used the following code.

```
#Loading the required libraries
library(hyperdraw)
library(hypergraph)

# Changes the color property to all the hyperedges
setProp <- function(gl,dh1,param,val) {
  for (x in dh1@head) {
    for (y in dh1@tail) {
      edgeData(gl,c(x,dh1@label),c(dh1@label,y),param) <- val
    }
  }
}

# Sets the list of the vertices used for the hypergraph representation
vertices <- c("tom","sam","pos1","pos2","id1","id2","id3","id4","content1","content2","
    geopos2","geopos1")
# Defining the hyperedges as header and tail
dh1 <- DirectedHyperedge(c("tom","pos1"),"id1","User2")
dh2 <- DirectedHyperedge(c("sam","pos2"),"id2","User1")
dh3 <- DirectedHyperedge(c("content1","geopos1","id1"),"id3","Post1")
dh4 <- DirectedHyperedge(c("content2","geopos2","id2"),"id4","Post2")
hg <- Hypergraph(vertices,list(dh1,dh2,dh3,dh4))
# Preparing the hypergraph visualization
gl <- graphLayout(graphBPH(hg))
setProp(gl,dh1,"color","blue")
setProp(gl,dh2,"color","blue")

# Saving the hypergraph plotting into a pdf file
pdf("/home/gyankos/EDropbox2/Tesi/src/image/hgs/figure01.pdf")
plot(gl)
dev.off()
```

Algorithm C.1: R code used for generating Figure 2.1.

C.2 DHIMP IMPLEMENTATION IN R

```
require(gmp)
require(Rmpfr)

dsum <- function(x) as.bigz(x*(x+1)/2)
dt <- function(i,j) as.bigz(as.bigz(j)+as.bigz(dsum(i+j)))

lhd <- function(l) unname(unlist(head(l,1)))
ltl <- function(l) tail(l,length(l)-1)
gmp_sqrt <- function(x) .mpfr2bigz(sqrt(.bigz2mpfr(x)))

dmax <- function(c) .mpfr2bigz(floor((.bigz2mpfr(gmp_sqrt(1+8*c)-1)/2))) ##TODO: sqrt
dtInv <- function(co) {
  dmx <- as.bigz(dmax(co))
  sdmax <- as.bigz(dsum(dmx))
  return(c(dmx-(co-sdmax),co-sdmax))
}

dtvec <- function(l) {
  R <- function(ls) {
    if (length(ls)==0)
      return(0)
    else if (length(ls)==1) {
      return(lhd(ls))
    } else {
      return(dt(lhd(ls),R(ltl(ls))))
    }

  }
  return(dt(length(l),R(l)))
}

dtvecInv <- function(l) {
  ij <- dtInv(l)
  n <- ij[[1]]
  cl <- ij[[2]]
  lis <- list()
  while(T) {
    ij <- dtInv(cl)
    if (n==0)
      return(lis)
    else if (n==1)
      return(c(lis,cl))
    else {
      #print(ij[[1]])
      if (length(lis)==0)
        lis <- c(ij[[1]])
      else
        lis <- c(lis,ij[[1]])
```

```
      n <- n-1
      cl <- ij[[2]]
    }
  }
}
```

Algorithm C.2: R code used for implementing the Dovetailing functions.

```
## The selection property is a trinary one which accepts the data, the database
## and the tensor itself
select <- function(db,Tensor,table,prop) {
  return(do.filter(table,function(x) prop(db,Tensor,x)))
}


PHI <- function(table) {
  lapply(table,function(x) new.Record(x$S,x$Ty,x$D,x$w,dtvec(x$phi)))
}

## Extends each record with the data computed
calc <- function(db,Tensor,table,Fun,Over,asR) {
  lapply(table,function(r){
    s<- r$S                                # Gets the schema
    li<- sapply(s,function(x) x %in% Over) # Gets the positions
    d <- r$D
    val <- Fun(db,Tensor,unlist(d[li]))
    ## TODO: rename r
    return(new.Record(c(s,asR),r$Ty,c(d,val),r$w,r$phi))
  })
}

## Embedding the function
embed <- function(table,val,asR) {
  lapply(table,function(r){
    ## TODO: rename r$T
    return(new.Record(c(r$S,asR),r$Ty,c(r$D,val),r$w,r$phi))
  })
}

project <- function(L,Lvec) {
  # This list memorizes the data that has been already seen in the linear scan of L
  lis  <- list()
  # For each similar entry in Lvec, I memorize all the different weights
  pesi <- list()
  # For each similar entry in Lvec, I memorize their indices in a list
  inde <- list()
  s <- list()
  t <- ""
  if (length(L)==0)
      return(L)
```

```
  ## Apply the projection over the data (linear time)
  L <- lapply(L,function(r) {
    d <- r$D
    s <<- r$S
    li<- sapply(s,function(x) x %in% Lvec)
    t <<- r$Ty
    d <- d[li]
    s <<- s[li]
    return(new.Record(s,r$Ty,d,r$w,r$phi))
  })

  ## Collect all the data to be returned
  L <- lapply(L,function(elem) {
    if (elem$D %in% lis) {
      #If I've already passed some data with the same projected values, I append the values
      where <- elem$D %==% lis
      pesi[where] <<- lapply(pesi[where],function(x) c(x,(elem$w)))
      inde[where] <<- lapply(inde[where],function(x) c(x,(elem$phi)))
    } else {
      lis <<- c(lis,  list(elem$D))
      pesi <<- c(pesi,list(elem$w))
      inde <<- c(inde,list(elem$phi))
    }
    return(elem)
  })

  inde <- unique(inde)

  ## Creates the list of records to return, where dtvec is the dovetailing for lists
  LL <- mapply(function(d,w,i){
    list(new.Record(s,t,d,mean(w),dtvec(i)))
  }, lis,pesi,inde)

  return(LL)
}

## L: table (list of records)
## Aggr: list data to aggregate
## Xi: data to compute (Type)
## OpXi: function over a vector of Xi where the Aggr values are equal
## asR: type of the generated data
groupby <- function(L,Aggr,Xi,OpXi,asR) {
  lis  <- list()
  pesi <- list()
  inde <- list()
  valu <- list()
  Lvec <- c(Aggr,Xi)
  s <- list()
  t <- ""
  li <- list()
```

```
liSec <- list()
XiPos <- list()
toRET <- list()

## Apply the projection over the selecting data
L <- lapply(L,function(r) {
  d <- r$D
  s <<- r$S
  li<<- sapply(s,function(x) x %in% Lvec)
  liSec<<- sapply(s,function(x) x %in% Aggr)
  XiPos<<- sapply(s,function(x) x %==% Xi)
  t <<- r$Ty
  d <- d[li]
  s <<- s[li]
  toRET <<- sapply(s,function(x) x %in% Aggr)
  return(new.Record(s,r$Ty,d,r$w,r$phi))
})


## Collect all the data to be returned
L <- lapply(L,function(elem) {

  dAggr <- elem$D
  dAggr <- dAggr[liSec]

  if (dAggr %in% lis) {
    where <- dAggr %==% lis
    pesi[where] <<- lapply(pesi[where],function(x) c(x,(elem$w)))
    inde[where] <<- lapply(inde[where],function(x) c(x,(elem$phi)))
    valu[where] <<- lapply(valu[where],function(x) {
      va <- elem$D
      va <- unlist(va[XiPos])
      return(c(x,(va)))
    })
  } else {
    lis <<- c(lis,  list(dAggr))
    pesi <<- c(pesi,list(elem$w))
    inde <<- c(inde,list(elem$phi))
    va <- elem$D
    va <- unlist(va[XiPos])
    valu <<- c(valu,list(va))
  }
  return(elem)
})
s <- s[toRET]

## TODO: t = t with Xi over Aggr
## Check if Op(v) is asR
LL <- mapply(function(d,w,i,v){
  list(new.Record( c(s,asR),t,c(d[toRET],OpXi(unlist(v))),mean(w),dtvec(i)))
}, lis,pesi,inde,valu)
```

```
  return(LL)
}

join <- function(table1,table2,db,Tensor,theta) {
  toret <- list()
  for (x in table1) {
    for (y in table2) {
      if (theta(db,Tensor,x,y)) {
        s <- c(x$S,y$S)
        t <- paste0(c(x$Ty,y$Ty),collapse="|><|")
        d <- c(x$D,y$D)
        w <- prod(x$w,y$w)
        i <- dt(x$phi,y$phi)
        toret <- c(toret,list(new.Record(s,t,d,w,i)))
      }
    }
  }
  return(toret)
}

## Renames Xfrom into Rto, and Rto will be the last column
rename <- function(table,Xfrom,Rto) {
  if (length(table)==0)
    return(table)
  ##TODO: assuming that all the tables have the same size and that Rto is not in schema
  s <- table[[1]]$S
  sS <- sapply(s,function(x) !(x%==%Xfrom))
  s <- c(s[sS],Rto)
  return(project(calc(NULL,NULL,table,function(a,b,x) x,list(Xfrom),Rto),s))
}
```

Algorithm C.3: R implementation for “list of rows”.

C.3 HYPERGRAPHS AND GSPAN EXTENDED IN JAVA

```
public class Dovetailing {

    public static final BigInteger TWO = BigInteger.ONE.add(BigInteger.ONE);
    public static final BigInteger EIGHT = BigInteger.TEN.subtract(TWO);

    public static BigInteger summate(BigInteger k) {
        return k.multiply(k.add(BigInteger.ONE)).divide(TWO);
    }

    public static BigInteger dt(BigInteger i, BigInteger j) {
        BigInteger ij = i.add(j);
        return summate(ij).add(j);
```

```
    }

    public static BigInteger index(int i) {
        return new BigInteger(Integer.toString(i));
    }

private static final BigDecimal SQRT_DIG = new BigDecimal(150);
private static final BigDecimal SQRT_PRE = new BigDecimal(10).pow(SQRT_DIG.intValue());

/**
 * Private utility method used to compute the square root of a BigDecimal.
 *
 * @author Luciano Culacciatti
 * @url http://www.codeproject.com/Tips/257031/Implementing-SqrtRoot-in-BigDecimal
 */
private static BigDecimal sqrtNewtonRaphson  (BigDecimal c, BigDecimal xn, BigDecimal
    precision){
    BigDecimal fx = xn.pow(2).add(c.negate());
    BigDecimal fpx = xn.multiply(new BigDecimal(2));
    BigDecimal xn1 = fx.divide(fpx,2*SQRT_DIG.intValue(),RoundingMode.HALF_DOWN);
    xn1 = xn.add(xn1.negate());
    BigDecimal currentSquare = xn1.pow(2);
    BigDecimal currentPrecision = currentSquare.subtract(c);
    currentPrecision = currentPrecision.abs();
    if (currentPrecision.compareTo(precision) <= -1){
        return xn1;
    }
    return sqrtNewtonRaphson(c, xn1, precision);
}

/**
 * Uses Newton Raphson to compute the square root of a BigDecimal.
 *
 * @author Luciano Culacciatti
 * @url http://www.codeproject.com/Tips/257031/Implementing-SqrtRoot-in-BigDecimal
 */
public static BigDecimal bigSqrt(BigDecimal c){
    return sqrtNewtonRaphson(c,new BigDecimal(1),new BigDecimal(1).divide(SQRT_PRE));
}

    public static BigInteger[] dtInv(BigInteger couple) {
        BigInteger toret[] = new BigInteger[2];
        BigInteger tosquare = BigInteger.ONE.add(EIGHT.multiply(couple));
        BigDecimal sq = bigSqrt(new BigDecimal(tosquare)).subtract(BigDecimal.ONE).divide(
            BigDecimal.ONE.add(BigDecimal.ONE),RoundingMode.FLOOR);
        BigInteger dmax = sq.toBigInteger();
        toret[1] = couple.subtract(summate(dmax));
        toret[0] = dmax.subtract(toret[1]);
        return toret;
    }
```

```
    private static BigInteger dtvecr(List<BigInteger> r) {
        if (r==null)
            return index(0);
        if (r.isEmpty())
            return index(0);
        if (r.size()==1)
            return r.get(0);
        else {
            Stack<BigInteger> s = new Stack<>();
            for (int j = r.size()-1; j>=0; j--) {
                BigInteger toadd = r.get(j);
                s.push(toadd);
                if (s.size()==2) {
                    BigInteger first = s.pop();
                    BigInteger last = s.pop();
                    s.push(dt(first,last));
                }
            }
            return s.pop();
        }

    }

    public static BigInteger dtVec(List<BigInteger> l) {
        return dt(index(l.size()),dtvecr(l));
    }

    public static List<BigInteger> dtVecInv(BigInteger bi) {
        BigInteger sl[] = dtInv(bi);
        int size = sl[0].intValue();
        List<BigInteger> lbi = new LinkedList<>();
        BigInteger prev = sl[1];
        for (int n=size; n>0; n--) {
            if (n==1)
                lbi.add(prev);
            else {
                BigInteger asl[] = dtInv(prev);
                lbi.add(asl[0]);
                prev = asl[1];
            }
        }
        return lbi;
    }


}
```

Algorithm C.4: Java code used for implementing the Dovetailing functions.

```
public class TableOperations {
```

```
/* missing code */

    public static Table project(Table t,Class... L) {
        Table toret = new Table(t.getName()+" over L", L);
        Map<Integer,List<Double>> wmap = new HashMap<>();
        Map<Integer,List<BigInteger>> imap = new HashMap<>();

        int i = 0;

        for (Tuple y : t) {
            Tuple x = projectTuple(y, L);
            int pos = toret.containsValuesPos(x);
            if (pos==-1) {
                toret.addRow(1,x.get().clone());
                LinkedList<Double> weight = new LinkedList<>();
                weight.add(x.getWeight());
                wmap.put(i,weight);
                LinkedList<BigInteger> index = new LinkedList<>();
                index.add(x.getIndex());
                imap.put(i,index);
                i++;
            } else {
                wmap.get(pos).add(x.getWeight());
                imap.get(pos).add(x.getIndex());
            }
        }

        for (int j=0; j<toret.size(); j++) {
            double size = wmap.get(j).size();
            double avg = 0;
            for (double d : wmap.get(j))
                avg += d;
            avg = avg / size;
            toret.get(j).setWeight(avg);
            //System.out.println("~~~~~");
            /*for (BigInteger x: imap.get(j)) {
                System.out.println(x);
            }*/
            //System.out.println("~~~~~");
            toret.get(j).setIndex( Dovetailing.dtVec(imap.get(j)));
            toret.add(toret.remove(j));
        }

        return toret;
    }

    public static Table select(Table t, IProperty prop){
        Table toret = new Table(t.getName(),t.getSchema());
        Iterator<Tuple> li = t.iterator();
        while (li.hasNext()) {
```

```
            Tuple current = li.next();
            if (prop.prop(current))
                toret.add(current);
        }
        return toret;
    }

    public static Table union(Collection<Table> lt) {
        if (lt==null)
            return null;
        if (lt.isEmpty())
            return new Table("empty table on union"); //with no size
        Class cls[] = null;
        String name = "";
        for (Table t : lt) {
            if (cls==null)
                cls = t.getSchema();
            else
                if (!Arrays.equals(cls, t.getSchema()))
                    throw new RuntimeException("Tables have different schemas");
            name = name + t.getName() + " v ";
        }
        name = name.substring(0,name.length()-2);
        Table toret = new Table(name, cls);
        for (Table t : lt) {
            for (BigInteger i : t.getAllKeys())
                toret.add(t.get(i));
        }
        return project(toret,cls);
    }

/* missing code */

}
```

Algorithm C.5: Java code used for implementing π, σ and $\bigcup$ over tables.

```
public class TensorOperations {

    public static <T extends ITensorLayer> Tensor<T> TUpdate(Database updatedOne, Tensor<T>
         tensor) {
        Tensor<T> toret = new Tensor<>(tensor.getLayersClass());
        Set<BigInteger> allKeys = updatedOne.getAllKeys();
        for (BigInteger x: allKeys) {
            System.out.println(x);
        }
        for (String layer : tensor.keySet()) {
            for (BigInteger dtx : allKeys) {
                List<BigInteger> vx = Dovetailing.dtVecInv(dtx);
                for (BigInteger dty : allKeys) {
```

```
                    List<BigInteger> vy = Dovetailing.dtVecInv(dty);
                    double size = vx.size() * vy.size();
                    if (size == 0)
                        continue;
                    double avg = 0;
                    for (BigInteger x : vx)
                        for (BigInteger y : vy)
                            avg = avg+tensor.get(layer).get(x, y);
                    avg = avg / size;
                    toret.set(dtx, dty, layer, avg);
                }
            }
        }
        return toret;
    }

    public static <T extends ITensorLayer> Tensor<T> TJoin(Database ndb, Collection<String>
         commonLayers, Tensor<T> tLeft, Tensor<T> tRight) {
         Tensor<T> nt = new Tensor<>(tLeft.getLayersClass());
        Set<BigInteger> allKeys = ndb.getAllKeys();
            for (String x : commonLayers) {
            T xtLeft = tLeft.get(x);
            T xtRight = tRight.get(x);
            for (BigInteger idx : allKeys) {
                BigInteger dtx[] = Dovetailing.dtInv(idx);
                for (BigInteger idy : allKeys) {
                    BigInteger dty[] = Dovetailing.dtInv(idy);
                    double avg = 0;
                    for (int i = 0; i < 2; i++) {
                        for (int j = 0; j < 2; j++) {
                            avg = avg + xtLeft.get(dtx[i], dty[j]) + xtRight.get(dtx[i],
                                dty[j]);
                        }
                    }
                    avg = avg / 8.0;
                    nt.set(idx, idy, x, avg);
                }
            }
        }
            return nt;
    }

}
```

Algorithm C.6: Java code used for implementing Update and Union tensor update functions.

```
public abstract class Abstract_gSpan<CodeType extends ICode> {

    //private SerializableList<IgSpanGraphs<CodeType>> database = null;
  private List<MultiLayerGraph> new_database = null;
```

```
    private AbstractSubgraphOf<CodeType> subgraphIsomorphismAlgorithm;

    /**
     * Filters the triples that could be added for rightmost expansion
     * @param mlg
     * @param ert
     * @param right_path
     * @return
     */
    public Collection<ERTriple> filterTriplesForExpansion(final IgSpanGraphs<CodeType> mlg,
         Collection<ERTriple> ert, final List<Entity> right_path) {
        return Collections2.filter(ert, new Predicate<ERTriple>() {
            @Override
            public boolean apply(ERTriple input) {
                //Keep the rightmost path (don't break it)
                if (input == null) {
                    return false;
                }
                //Remove errors ~ belts and suspenders
                if (input.hasNull())
                    return false;

                if (mlg.getRepresentation() != null) {
                    //get the graph
                    MultiLayerGraph tmp = mlg.getRepresentation();
                    //for all the outgoing vertices from input.source();
                    for (Entity x : tmp.getOutSet(input.getSource(), null)) {
                        //remove the triple if this could break the integrity of the
                            rightmost path
                        if (input.getDestination().compareTo(x) > 0) {
                            return false;
                        }
                    }
                }

                return right_path.contains(input.getSource());
            }
        });
    }

    /**
     * Forcing the implementation of an algorithm
     * @param subIso        Subgraph Algorithm Implementation
     */
    protected Abstract_gSpan(AbstractSubgraphOf<CodeType> subIso) {
        subgraphIsomorphismAlgorithm = subIso;
    }

    public abstract IgSpanGraphs<CodeType> createCodeGraph(MultiLayerGraph m);
```

```
    /*public Collection<IgSpanGraphs<CodeType>> getDatabase() {
        return database;
    }*/

    public void setDatabase(List<MultiLayerGraph> mlgdb) {
        new_database = mlgdb;
    }

    public double support(MultiLayerGraph g) {
        double n = 0;
        for (MultiLayerGraph x : new_database) {
            if (subgraphIsomorphismAlgorithm.subgraphOf(g, x)) {
                n++;
            }
        }
        n = n / ((double) new_database.size());
        return n;
    }

    public Collection<ERTriple> getMostFrequentEdges(double thereshold) {

        HashMap<ERTriple, Double> count = new HashMap<>();
        if (thereshold <= 0) {
            thereshold = Double.MIN_VALUE;
        }
        final double tmpt = thereshold;

        int i = 1;
        Iterator<MultiLayerGraph> mlgi = new_database.iterator();
        //Initializing count, that is counting the occurrence of the edges inside the
            database
        while (mlgi.hasNext()) {
          MultiLayerGraph x = mlgi.next();
          System.out.print(i+ ", ");
          i++;
          if (x==null)
            System.out.println("ERROR");
          IgSpanGraphs<CodeType> old = createCodeGraph(x);
            for (ERTriple y : old.getAllEdges()) {
                if (!count.containsKey(y)) {
                    /*
                     //TEST
                     for (ERTriple z:count.keySet())
                     if (z.equals(y))
                     throw new RuntimeException("ERROR: equal but not incremented");
                     */
                    count.put(y, (double) 1);
                } else {
                    count.put(y, count.get(y) + 1);
                }
            }
```

```
        }

        //Normalizing factor
        final double size = new_database.size();

        //Filters the entries by their frequency
        Collection<Map.Entry<ERTriple, Double>> frequentOnes = Collections2.filter(count.
            entrySet(), new Predicate<Map.Entry<ERTriple, Double>>() {
            @Override
            public boolean apply(Map.Entry<ERTriple, Double> t) {
                return ((t.getValue() / size) >= tmpt);
            }
        });

        //Given the entry, returns only the key, that is the triple  ~~~ Vonlenska/
            Hopelandic
        return Collections2.transform(frequentOnes, new Function<Map.Entry<ERTriple, Double
            >, ERTriple>() {
            @Override
            public ERTriple apply(Map.Entry<ERTriple, Double> f) {
                return f.getKey();
            }
        });

    }

    public static MultiLayerGraph toMLG(ERTriple input) {
        LinkedList<ERTriple> e = new LinkedList<>();
        e.add(input);
        return MultiLayerGraph.create(e);
    }


    public abstract List<Entity> getEntities(final IgSpanGraphs<CodeType> mlg);

        /**
     * Given a base graph and a set of possible edge extensions, it returns the
     * set of all the possible rightmost-extensions of such graph
     *
     * @param mlg basic graph
     * @param ert List of possible edge-right extensions
     * @return
     */
    public Collection<IgSpanGraphs<CodeType>> getRightmostExpansions(final IgSpanGraphs<
        CodeType> mlg, Collection<ERTriple> ert) {

        Collection<IgSpanGraphs<CodeType>> toret;
        if (ert.isEmpty()) {
            return new LinkedList<>();
        }

```

```
        ////////////////////////////////////////////////////////////////////
        //Returns the sequence of the entities encountered on the rightmostpath
        AbstractDBVisit<List<Entity>> adbv_rightmost = new SimpleRightmostExpansion();
        DFS<List<Entity>> d = new DFS<List<Entity>>(mlg.getRepresentation(), adbv_rightmost
            );
        final List<Entity> right_path = d.start();
        ////////////////////////////////////////////////////////////////////

        //Returns the possible rightmost extensions
        ert = filterTriplesForExpansion(mlg, ert, right_path);

        //Add the rightmost extension to the graph itself
        toret = Collections2.transform(ert, new Function<ERTriple, IgSpanGraphs<CodeType
            >>() {
            @Override
            public IgSpanGraphs<CodeType> apply(ERTriple input) {
                CodeType ls = mlg.getACode();
                ls.add(input);
                return createCodeGraph(MultiLayerGraph.create(ls));
            }
        });

        return toret;
    }

    public void gSpan(double edge_t, int depth, List<MultiLayerGraph> returned) {
        gSpan(edge_t, depth,Double.MIN_VALUE,returned);
    }

    public void gSpan(double edge_threshold, int depth_search, double support_thereshold,
        List<MultiLayerGraph> returned) {
        //Candidates for rightmost expansion
      System.out.println("getting frequent edges... ");
        LinkedList<ERTriple> N = new LinkedList<>(getMostFrequentEdges(edge_threshold));
        System.out.print("Done.");

        //HashSet<SortedSet<ERTriple>> S = new HashSet<>();

        for (ERTriple n : N) {
          System.out.println("Triple ~ " + n.toString());
            int depth;
            returned.add(toMLG(n)); //adds first element
            LinkedList<IgSpanGraphs<CodeType>> gsgs = new LinkedList<>();
            Stack<Integer> layer = new Stack<>();
            gsgs.push(createCodeGraph(toMLG(n)));
            layer.push(0);
            while (!gsgs.isEmpty()) {
                final IgSpanGraphs<CodeType> candidate = gsgs.get(0); //double-checking
                    staticity of the result
                gsgs.remove();
                depth = layer.pop();
```

```
                if ((depth < depth_search)
                        && (support(candidate.getRepresentation()) >= support_thereshold)
                        ) {

                    List<IgSpanGraphs<CodeType>> gsl = new LinkedList<>(
                        getRightmostExpansions(candidate, N));

                    SortedSet<ERTriple> tmpEdges = candidate.getAllEdges();
                    if (!tmpEdges.isEmpty())
                        returned.add(MultiLayerGraph.create(tmpEdges));
                    int hasNull = 0;

                    for (IgSpanGraphs<CodeType> kt : gsl) {
                        if (kt!=null)
                            gsgs.push(kt);
                        else
                            hasNull += 1;
                    }

                    for (int j = 0; j < gsl.size()-hasNull; j++) {
                        layer.push(depth + 1);
                    }
                }

            }
        }

/* missing code */


}
```

Algorithm C.7: Java abstract code implementing the general version of the gSpan Algorithm over DHImp-s.

C.4 GRAPH MINING ALGORITHMS

```
################################################################################
## MCL

# Inflation step of MCL
mcl.inflate <- function (M,
         inf) {
  M <- M^(inf);
  return (M);
}

# Normalize the matrix by column
```

```
mcl.norm <- function (M) {
  colum.sum <- apply(M,2,sum)
  M <- t(M) / colum.sum
  return (t(M))
}

# MCL procedure
mcl <- function (M,   # Matrix
     inf,   # Inflation value
     iter,  # Number of iterations
     verbose = F
     ) {
  for (i in 1:iter) {
    old.M <- M;
    M.norm <- mcl.norm(M);
    M <- M.norm%*%M.norm;
    M <- mcl.inflate(M, inf);
    M <- mcl.norm(M);
    if (sum(old.M == M) == dim(M)[1]*dim(M)[2]) {
      break;
    }
    if (verbose) {
      print (paste ("iteration", i));
    }
  }
  return (M);
}


collect.mcl.clusters <- function (M   # Matrix (mcl result)
                    ) {
  M.names <- 1:nrow(M);
  l <- list()
  clustered.nodes <- vector(mode = "logical", length = dim(M)[1])
  for (i in 1:dim(M)[1]) {
    nodes <- M.names[which(M[i,] != 0)];
    if (length(nodes) > 0 && !clustered.nodes[which(M[i,] != 0)]) {
      l <- c(l,list(nodes))
      clustered.nodes[which(M[i,] != 0)] = T;
    }
  }
  return(l);
}
```

Algorithm C.8: Markov Clustering Algorithm, slightly modified from http://www.bigre.ulb.ac.be/Users/jvanheld/BMOL-F-501/practicals/r_scripts/mcl.R.

```
public void gSpan(double edge_threshold, int depth_search, double support_thereshold, List<
    MultiLayerGraph> Solution) {
    //Candidates for rightmost expansion
```

```
    LinkedList<ERTriple> N = new LinkedList<>(getMostFrequentEdges(edge_threshold));

    for (ERTriple n : N) {
      System.out.println("Triple ~ " + n.toString()); //Current examined triple
        int depth;                                    //Depth of the DFS Search tree
        Solution.add(toMLG(n));                       //Adds first element as graph
        LinkedList<IgSpanGraphs<CodeType>> gsgs = new LinkedList<>();
        Stack<Integer> layer = new Stack<>(); //Defines a stack for the current DFS tree
            layer depth
        gsgs.push(createCodeGraph(toMLG(n))); //Pushes the triple to the DFS subtree
            departing from n
        layer.push(0);                        //Start layer
        while (!gsgs.isEmpty()) {
            //Pop the first element and the layer information
            final IgSpanGraphs<CodeType> candidate = gsgs.get(0);
            gsgs.remove();
            depth = layer.pop();
            if ((depth < depth_search) //If I've reached the search depth and if it is
                supported
                && (support(candidate.getRepresentation()) >= support_thereshold)
                ) {
                    //Returns the list of the overall possible rightmost expansions
                    List<IgSpanGraphs<CodeType>> gsl = new LinkedList<>(
                        getRightmostExpansions(candidate, N));

                    SortedSet<ERTriple> tmpEdges = candidate.getAllEdges();
                    if (!tmpEdges.isEmpty())
                        Solution.add(MultiLayerGraph.create(tmpEdges));
                    int hasNull = 0;

                    //For each possible next expansion
                    for (IgSpanGraphs<CodeType> kt : gsl) {
                        if (kt!=null)
                            gsgs.push(kt);
                        else
                            hasNull += 1;
                    }

                    for (int j = 0; j < gsl.size()-hasNull; j++) {
                        layer.push(depth + 1);
                    }
                }

        }
    }
```

Algorithm C.9: gSpan Java Implementation

```
/**
 * Given a base graph and a set of possible edge extensions, it returns the
```

```
 * set of all the possible rightmost-extensions of such graph
 *
 * @param mlg basic graph
 * @param ert List of possible edge-right extensions
 * @return
 */
public Collection<IgSpanGraphs<CodeType>> getRightmostExpansions(final IgSpanGraphs<
    CodeType> mlg, Collection<ERTriple> ert) {
    Collection<IgSpanGraphs<CodeType>> toret;
    if (ert.isEmpty()) {
        return new LinkedList<>();
    }

    ////////////////////////////////////////////////////////////////////
    //Returns the sequence of the entities encountered on the rightmostpath
    AbstractDBVisit<List<ERTriple>> adbv_rightmost = new SimpleRightmostExpansion();
    DFS<List<ERTriple>> d = new DFS<List<ERTriple>>(mlg.getRepresentation(), adbv_rightmost
        );
    final List<ERTriple> right_path = d.start();
    ////////////////////////////////////////////////////////////////////

    //Returns the possible rightmost extensions
    ert = filterTriplesForExpansion( ert, right_path);

    //Add the rightmost extension to the graph itself
    toret = Collections2.transform(ert, new Function<ERTriple, IgSpanGraphs<CodeType>>() {
        @Override
        public IgSpanGraphs<CodeType> apply(ERTriple input) {
           CodeType ls = mlg.getACode();
           ls.add(input);
           return createCodeGraph(MultiLayerGraph.create(ls));
        }
    });

    return toret;
}

/**
 * Filters the triples that could be added for rightmost expansion
 */
public Collection<ERTriple> filterTriplesForExpansion(Collection<ERTriple> ert, final List<
    ERTriple> right_path) {
    return Collections2.filter(ert, new Predicate<ERTriple>() {
        @Override
        public boolean apply(ERTriple input) {
            //Keep the rightmost path (don't break it)
            if (input == null) {
                return false;
            }
            //Remove errors ~ belts and suspenders
            if (input.hasNull())
```

```
                return false;
            //for all the outgoing vertices from input.source();
            for (ERTriple x : right_path) {
                //remove the triple if this could break the integrity of the rightmost path
                if (input.compareTo(x) > 0) {
                    return false;
                }
            }

            return right_path.contains(input.getSource());
        }
    });
}
```

Algorithm C.10: getRightmostExpansions implementation

C.5 TEXT MINING ALGORITHMS

```
#install.packages("stringr") (requires)

library(tm)
pos <- Corpus(DirSource("/home/gyankos/Scrivania/sentiment/pos")) #Loading the positive
    documents
neg <- Corpus(DirSource("/home/gyankos/Scrivania/sentiment/neg")) #Loading the negative
    documents

#Modifying the documents
rempunct <- function (x) { gsub("[[:punct:]]", "", x) }
remdigit <- function (x) { gsub("\\d+", "", x) }
pos <- tm_map(pos, rempunct)        #Removes punctuation
neg <- tm_map(neg, rempunct)
pos <- tm_map(pos, stripWhitespace) #Removes white spaces
neg <- tm_map(neg, stripWhitespace)
pos <- tm_map(pos, tolower)         #all the words in lowercase
neg <- tm_map(neg, tolower)
pos <- tm_map(pos, removeWords,stopwords("english")) #removing stop words
neg <- tm_map(neg, removeWords,stopwords("english"))
pos <- tm_map(pos, stemDocument)    #Stemming the documents
neg <- tm_map(neg, stemDocument)
pos <- tm_map(pos, remdigit)        #Removing the numbers
neg <- tm_map(neg, remdigit)
pos <- tm_map(pos, stripWhitespace) #Removing white spaces again
neg <- tm_map(neg, stripWhitespace)

#Loading the positive and negative dictionary
posw <- as.list(readLines("/home/gyankos/Scrivania/sentiment/positivew.txt"))
negw <- readLines("/home/gyankos/Scrivania/sentiment/negativew.txt")
posw <- unique(rapply(list(posw),stemDocument))
```

```
negw <- unique(rapply(list(negw),stemDocument))

#Test
#dtm <- DocumentTermMatrix(pos,posw)
#rst_dtm <- (removeSparseTerms(dtm, 0.4))

library(stringr)
splitwords <- function(x) str_split(x, '\\s+')
pos <- tm_map(pos, splitwords) #Splitting each document in word lists
neg <- tm_map(neg, splitwords)
reify <- function (y) {
  x = list()
  for (i in 1:length(pos)) x[[i]] <- unlist(unique(unlist(y[[i]])))
  x
}
pos <- reify(pos)              #merge all the subslists caused by \n
neg <- reify(neg)


pos_result = list()
for (j in 1:length(pos)) {
  pos_result[j] = (sum(!is.na(match(unlist(pos[j]),posw)))-sum(!is.na(match(unlist(pos[j]),
      negw)))>0)
}
pos_ok <- sum(unlist(pos_result))/length( pos_result)

#And paradoxally, a negative document could acheive a positive score
neg_result = list()
for (j in 1:length(neg)) {
  neg_result[j] = (sum(!is.na(match(unlist(neg[j]),posw)))-sum(!is.na(match(unlist(neg[j]),
      negw)))<0)
}
neg_ok <- sum(unlist(neg_result))/length(neg_result)


reliability <- (sum(unlist(pos_result))+sum(unlist(neg_result)))/(length(pos_result)+length
    (neg_result))

print(reliability) # [1] 0.6275
```

Algorithm C.11: First sentiment analysis algorithm with unweighted vocabulary and film-review corpus.

```
#install.packages("stringr") (requires)

library(tm)
pos <- Corpus(DirSource("/home/gyankos/Scrivania/sentiment/pos")) #Loading the positive
    documents
neg <- Corpus(DirSource("/home/gyankos/Scrivania/sentiment/neg")) #Loading the negative
    documents
```

```
#Modifying the documents
rempunct <- function (x) { gsub("[[:punct:]]", "", x) }
remdigit <- function (x) { gsub("\\d+", "", x) }
pos <- tm_map(pos, rempunct)        #Removes punctuation
neg <- tm_map(neg, rempunct)
pos <- tm_map(pos, stripWhitespace) #Removes white spaces
neg <- tm_map(neg, stripWhitespace)
pos <- tm_map(pos, tolower)         #all the words in lowercase
neg <- tm_map(neg, tolower)
pos <- tm_map(pos, removeWords,stopwords("english")) #removing stop words
neg <- tm_map(neg, removeWords,stopwords("english"))
pos <- tm_map(pos, stemDocument)    #Stemming the documents
neg <- tm_map(neg, stemDocument)
pos <- tm_map(pos, remdigit)        #Removing the numbers
neg <- tm_map(neg, remdigit)
pos <- tm_map(pos, stripWhitespace) #Removing white spaces again
neg <- tm_map(neg, stripWhitespace)

#Loading the positive and negative dictionary
posneg <- read.delim(file='/home/gyankos/Scrivania/sentiment/AFINN-111.txt', header=FALSE,
    stringsAsFactors=FALSE)
names(posneg) <- c('word', 'score')
posneg$word <- tolower(posneg$word)
posneg$word <- stemDocument(posneg$word) #Stemming similar words
posneg <- posneg[!duplicated(posneg),]   #Removing duplicated
getrowscore <- function(x,y) x[i,2]

#Test
#dtm <- DocumentTermMatrix(pos,posw)
#rst_dtm <- (removeSparseTerms(dtm, 0.4))

library(stringr)
splitwords <- function(x) str_split(x, '\\s+')
pos <- tm_map(pos, splitwords) #Splitting each document in word lists
neg <- tm_map(neg, splitwords)
reify <- function (y) {
  x = list()
  for (i in 1:length(pos)) x[[i]] <- unlist(unique(unlist(y[[i]])))
  x
}
pos <- reify(pos)              #merge all the subslists caused by \n
neg <- reify(neg)

#ERROR: in this case I score -1 on a positive document


pos_result = list()
for (j in 1:length(pos)) {
  pos_result[j] = (sum(na.omit(unlist(lapply(match(unlist(pos[j]),posneg$word),function (x)
       posneg[x,2])))) > 0)
```

```
}
pos_ok <- sum(unlist(pos_result))/length( pos_result)

#And paradoxally, a negative document could acheive a positive score
neg_result = list()
for (j in 1:length(neg)) {
  neg_result[j] = (sum(na.omit(unlist(lapply(match(unlist(neg[j]),posneg$word),function (x)
       posneg[x,2])))) < 0)
}
neg_ok <- sum(unlist(neg_result))/length(neg_result)

reliability <- (sum(unlist(pos_result))+sum(unlist(neg_result)))/(length(pos_result)+length
    (neg_result))

print(reliability) # [1] 0.6225
```

Algorithm C.12: Second sentiment analysis algorithm with weighted vocabulary and film-review corpus.

```
# required libraries
### http://www.youtube.com/watch?v=j1V2McKbkLo
library(tm)
library(plyr)
library(class)

remdigit <- function (x) { gsub("\\d+", "", x) }

stemOut <- function (pos) {
  pos <- tm_map(pos, removePunctuation)
  pos <- tm_map(pos, stripWhitespace)
  pos <- tm_map(pos, tolower)
  pos <- tm_map(pos, removeWords,stopwords("english"))
  pos <- tm_map(pos, stemDocument)
  pos <- tm_map(pos, remdigit)
  pos <- tm_map(pos, stripWhitespace)
  pos
}

genTermDocumentMatrix <- function (clas,path,sparsness) {
  apath <- sprintf("%s/%s",path,clas)
  x <- Corpus(DirSource(directory = apath))
  x <- stemOut(x)
  x <- TermDocumentMatrix(x)
  tdm_cor <- removeSparseTerms(x,sparsness)

  #By transposing the matrix, each term will be a columns and the other way around for
      documents
  tdm_cor <- as.data.frame(t(data.matrix(tdm_cor)))
  tdm_cor <- cbind(tdm_cor, rep(clas,nrow(tdm_cor)))
  colnames(tdm_cor)[ncol(tdm_cor)] = "targetClass"
```

```
  return(tdm_cor)
}


allTDM <- function (classvec,path,sparsness) {
    x <- do.call(rbind.fill,lapply(classvec,genTermDocumentMatrix,path,sparsness))
    x[is.na(x)] = 0
    return(x)
}

doDataSet <- function (classvec,path,sparsness,trainperc) {
  tdm <- allTDM(classvec,path,sparsness)

  tdm_train <- sample(nrow(tdm),ceiling(nrow(tdm) * trainperc))
  tdm_test <- (1:nrow(tdm)) [ -tdm_train]

  # Projection over targetClass attrs
  classes <- tdm[, "targetClass"]
  # Projection over the other columns (except targetClass) attrs
  datas   <- tdm[, !colnames(tdm) %in% "targetClass"]

  #KNN algorithm
  pred <- knn(datas[tdm_train,],datas[tdm_test,],classes[tdm_train])
  conf <- table(Prediction=pred,Actual=classes[tdm_test])
  acc <- sum(diag(conf)/ length(tdm_test))
  calc <- function(x) knn(datas[tdm_train,],x,classes[tdm_train])

  return(list(prediction=pred,confusion_matrix=conf,accuracy=acc,calculate=calc))
}

data_class <- c("neg","pos")
cpath <- "/home/gyankos/Scrivania/sentiment"
tspar = 0.7
trainp = 0.7

x = list()
dds <- doDataSet(data_class,cpath,tspar,trainp)
best <- dds$accuracy
x[1] <- best
print(best)
for (i in 1:24) {
  tmp <- doDataSet(data_class,cpath,tspar,trainp)
  if (tmp$accuracy > best) {
    dds <- tmp
    best <- tmp$accuracy
    print("best found")
    print(best)
  }
  x[i+1] <- tmp$accuracy
```

```
  print(i)
}

print(best)
plot(1:25,unlist(x),xlab="tests",ylab="accuracy")
lines(1:25,unlist(x))
```

Algorithm C.13: Second sentiment analysis algorithm with *k-nearest neighbor algorithm* training over film-review corpus.

```
# required libraries
### http://www.youtube.com/watch?v=j1V2McKbkLo
library(tm)
library(plyr)
library(class)

remdigit <- function (x) { gsub("\\d+", "", x) }

stemOut <- function (pos) {
  pos <- tm_map(pos, removePunctuation)
  pos <- tm_map(pos, stripWhitespace)
  pos <- tm_map(pos, tolower)
  pos <- tm_map(pos, removeWords,stopwords("english"))
  pos <- tm_map(pos, stemDocument)
  pos <- tm_map(pos, remdigit)
  pos <- tm_map(pos, stripWhitespace)
  pos
}

genTermDocumentMatrix <- function (clas,path,sparsness) {
  apath <- sprintf("%s/%s",path,clas)
  x <- Corpus(DirSource(directory = apath))
  x <- stemOut(x)
  x <- TermDocumentMatrix(x)
  tdm_cor <- removeSparseTerms(x,sparsness)

  #By transposing the matrix, each term will be a columns and the other way around for
      documents
  tdm_cor <- as.data.frame(t(data.matrix(tdm_cor)))
  tdm_cor <- cbind(tdm_cor, rep(clas,nrow(tdm_cor)))
  colnames(tdm_cor)[ncol(tdm_cor)] = "targetClass"

  return(tdm_cor)
}



allTDM <- function (classvec,path,sparsness) {
    x <- do.call(rbind.fill,lapply(classvec,genTermDocumentMatrix,path,sparsness))
    x[is.na(x)] = 0
```

```
    return(x)
}

doDataSet <- function (classvec,path,sparsness,trainperc) {
  tdm <- allTDM(classvec,path,sparsness)
  tdm_train <- sample(nrow(tdm),ceiling(nrow(tdm) * trainperc))
  tdm_test <- (1:nrow(tdm)) [ -tdm_train]

  #Loading the positive and negative dictionary (with less words)
  posneg <- read.delim(file='/home/gyankos/Scrivania/sentiment/AFINN-111.txt', header=FALSE
      , stringsAsFactors=FALSE)
  names(posneg) <- c('word', 'score')
  posneg$word <- tolower(posneg$word)
  posneg$word <- stemDocument(posneg$word) #Stemming similar words
  posneg <- posneg[!duplicated(posneg),]   #Removing duplicated
  getrowscore <- function(x,y) x[i,2]

  # Removing not semantic words
  words <- match(colnames(tdm),posneg$word)
  # Projection over targetClass attrs
  classes <- tdm[, "targetClass"]

  # Projection over the other columns of names present in words
  tdm <- tdm[!is.na(words)]

  datas   <- tdm[, !colnames(tdm) %in% "targetClass"]

  #KNN algorithm
  pred <- knn(datas[tdm_train,],datas[tdm_test,],classes[tdm_train])
  conf <- table(Prediction=pred,Actual=classes[tdm_test])
  acc <- sum(diag(conf)/ length(tdm_test))
  calc <- function(x) knn(datas[tdm_train,],x,classes[tdm_train])

  return(list(prediction=pred,confusion_matrix=conf,accuracy=acc,calculate=calc))
}

data_class <- c("neg","pos")
cpath <- "/home/gyankos/Scrivania/sentiment"
tspar = 0.7
trainp = 0.7

x = list()
dds <- doDataSet(data_class,cpath,tspar,trainp)
best <- dds$accuracy
x[1] <- best
print(best)
for (i in 1:24) {
  tmp <- doDataSet(data_class,cpath,tspar,trainp)
  if (tmp$accuracy > best) {
    dds <- tmp
    best <- tmp$accuracy
```

```
    print("best found")
    print(best)
  }
  x[i+1] <- tmp$accuracy
  print(i)
}

print(best)
plot(1:25,unlist(x),xlab="tests",ylab="accuracy")
lines(1:25,unlist(x))
```

Algorithm C.14: Second sentiment analysis algorithm with *k-nearest neighbor algorithm* training over film-review corpus.

C.6 PROOFS

```
include "arithmetics/nat.ma".
include "basics/bool.ma".
include "basics/lists/listb.ma".

axiom dt: nat →nat →nat.
axiom dt_equiv: ∀a,b,c,d. dt a b = dt c d → a = c ∧ b = d.

let rec R (l:list nat) := match l with
 [ nil ⇒ O
 | cons a b ⇒ match b with
              [ nil ⇒ a
              | cons c d ⇒ dt a (R b)]].

definition dtl := λl. dt (length nat l) (R l).

lemma rw_dtl: ∀l. dtl l = dt (length nat l) (R l). // qed.
lemma R_0: R [] = O. // qed.
lemma R_cons: ∀a. R (a::[])=a. // qed.
lemma R_ccons: ∀a,b,c. R (a::b::c) = dt a (R (b::c)). // qed.


lemma lis_emp: ∀m:list nat. 0=|m|→m=[]. #m
elim m // #A #B #Abs #A1 @False_ind normalize in A1; @(absurd ... A1 (not_eq_O_S (|B|)))
    qed.


lemma Sx: ∀x,y. (S x)=(S y)→x=y. // qed.
lemma Hdx:∀a,c:nat.∀b,d:list nat. |a::b|=|c::d|→S(|b|)=S(|d|). #H5 #H6 #H7 #H8 #H9
    normalize in H9; @H9 qed.

theorem Lnat_elim2 :
 ∀R:list nat → list nat → Prop.
```

```
  (∀n:list nat. R [] n)
  → (∀n,m. R (n::m) [])
  → (∀n,m,o,p. R m p → R (n::m) (o::p))
  → ∀n,m. R n m.
#R #ROn #RSO #RSS #n (elim n) -n // #n0 #Rn0m #H #m (cases m)
  [ @RSO | #n1 #ls @RSS @H ] qed.


lemma testa: ∀l,m. dtl l = dtl m → l = m.
#l #m >rw_dtl >rw_dtl #H lapply(dt_equiv ... H) -H * @(Lnat_elim2 ... l m)
[ #L normalize #H1 lapply(lis_emp ...H1) -H1 #Finish * @sym_eq @Finish
| #N #L #A1 normalize in A1; #A2 -A2 @False_ind
  lapply(sym_eq nat ...A1) -A1 #A1 @(absurd ... A1 (not_eq_O_S (?)))
| -l -m #A #B #C #D
   @(Lnat_elim2 ... B D)
   [ #N #IH normalize in IH; #Elem normalize in Elem; lapply(Sx ... Elem) -Elem #Elem
     lapply(IH ... Elem) -IH #IH lapply( lis_emp ... Elem) -Elem #H1 >H1 normalize #rw >rw
         //
   | #E #F #A1 -A1 #A2 normalize in A2; lapply(Sx ... A2) -A2 #A2 lapply(sym_eq nat ... A2)
       -A2
     #A2 #A1 -A1 @False_ind @(absurd ... A2 (not_eq_O_S (|F|)))
   | #E #F #G #H
     #IH1 #IH2 #H1 lapply(Hdx ... H1) -H1 #H1 lapply(Sx ... H1) -H1 #H1 lapply(IH2 ... H1) -
         IH2 #IH2
     >R_ccons >R_ccons #H1 lapply(dt_equiv... H1) -H1 * #H1 >H1 in IH1; #IH1 #H2
     lapply(IH2... H2) -IH2 #H3 >H3 //] qed.


lemma coda: ∀l,m. l = m →dtl l = dtl m .
#l #m #rw >rw //
```

Algorithm C.15: Proof of a preliminar lemma.

D

REFERENCES

D.1 BIBLIOGRAPHY

[ABB06] Sameer Agarwal, Kristin Branson, and Serge Belongie. **"Higher Order Learning with Graphs"**. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania: ACM, 2006, pp. 17–24 (cit. on pp. 28, 51, 62).

[Ace+07] Luca Aceto et al. ***Reactive Systems: Modelling, Specification and Verification***. New York, NY, USA: Cambridge University Press, 2007 (cit. on p. 55).

[Ana07] Revolution Analytics. ***RHadoop and MapR: Accessing Enterprise-Grade Hadoop from R***. 2007 (cit. on p. 128).

[Ana11] Revolution Analytics. ***Big Data Analytics in R. Big Opportunity, Big Challenge***. 2011 (cit. on pp. 120, 121).

[Ber+11] Michele Berlingerio et al. **"Foundations of Multidimensional Network Analysis."** In: *ASONAM*. IEEE Computer Society, 2011, pp. 485–489 (cit. on p. 12).

[BH09] Sylvain Brohée and Jacques van Helden. **"Evaluation of clustering algorithms for protein-protein interaction networks."** In: *BMC Bioinformatics* 7 (Nov. 10, 2009), p. 488 (cit. on pp. 14, 54).

[BHS07] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. ***Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing***. Chichester, UK: Wiley, 2007 (cit. on p. 118).

[BM10] Alan Bertossi and Alberto Montresor. ***Algoritmi e Strutture di Dati***. De Agostini, CittàStudi Edizioni, May 2010 (cit. on p. 28).

[Cal+06] Toon Calders et al. **"Expressive power of an algebra for data mining"**. In: *ACM Trans. Database Syst.* 31.4 (Dec. 2006), pp. 1169–1214 (cit. on pp. 35, 38–43, 113).

[CH06a] Diane J. Cook and Lawrence B. Holder. ***Mining Graph Data***. John Wiley & Sons, 2006 (cit. on pp. 55, 61).

[Deb+99] J.C.W. Debuse et al. ***A methodology for knowledge discovery: a KDD roadmap***. Tech. rep. SYS Technical Report SYS-C99-01, 1999 (cit. on p. 35).

[Des+05] Mukund Deshpande et al. **"Frequent Substructure-Based Approaches for Classifying Chemical Compounds"**. In: *IEEE Transactions on Knowledge and Data Engineering* 17 (8 2005). Ed. by Beng Chin Ooi, pp. 1036–1050 (cit. on pp. 14, 57).

[Donoo] Stijn van Dongen. **"Graph Clustering by Flow Simulation"**. PhD thesis. University of Utrecht, 2000 (cit. on pp. 52, 54).

[DVMT13] Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone. **"Converting Relational to Graph Databases"**. In: *First International Workshop on Graph Data Management Experiences and Systems*. GRADES '13. New York, New York: ACM, 2013, 1:1–1:6 (cit. on p. 27).

[ERV05] Ernesto Estrada and Juan A. Rodríguez-Velázquez. ***Complex Networks as Hypergraphs***. 2005 (cit. on p. 12).

[Fag83] Ronald Fagin. **"Degrees of acyclicity for hypergraphs and relational database schemes"**. In: *J. ACM* 30.3 (July 1983), pp. 514–550 (cit. on p. 11).

[Fan+11] Wenfei Fan et al. **"Adding regular expressions to graph reachability and pattern queries."** In: *ICDE*. Ed. by Serge Abiteboul et al. IEEE Computer Society, 2011, pp. 39–50 (cit. on p. 55).

[Fis36] R. A. Fisher. **"The use of multiple measurements in taxonomic problems"**. In: *Annals of Eugenics* 07 (July 1936) (cit. on p. 62).

[FM08] Dario Floreano and Claudio Mattiussi. ***Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies***. The MIT Press, 2008 (cit. on p. 97).

[For10] Santo Fortunato. **"Community detection in graphs"**. In: *Physics Reports* 486.3-5 (2010), pp. 75 –174 (cit. on p. 14).

[Fre05] David Freedman. ***Statistical Models: Theory and Practice***. Cambridge University Press, Aug. 2005 (cit. on pp. 11, 80).

[Gal+93] Giorgio Gallo et al. **"Directed Hypergraphs and Applications"**. In: *Discrete Appl. Math.* 42.2-3 (Apr. 1993), pp. 177–201 (cit. on pp. 11, 21, 29).

[Gen13] Jeff Gentry. ***twitteR: R based Twitter client***. R package version 1.1.7. 2013 (cit. on pp. 99, 109).

[GJ90] Michael R. Garey and David S. Johnson. ***Computers and Intractability; A Guide to the Theory of NP-Completeness***. New York, NY, USA: W. H. Freeman & Co., 1990 (cit. on pp. 52, 55).

[HKP12] Jiawei Han, Micheline Kamber, and Jian Pei. ***Data mining: Concepts and techniques, Third edition***. Waltham, Mass.: Morgan Kaufmann Publishers, 2012 (cit. on pp. 44, 115).

[Hol12] Alex Holmes. ***Hadoop in Practice***. Manning Publications Co., 2012 (cit. on pp. 116–118, 126, 133).

[ISO11] ISO/IEC/IEEE. **"Systems and software engineering – Architecture description"**. In: *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)* (Jan. 2011), pp. 1 –46 (cit. on p. 119).

[JLN00] Theodore Johnson, Laks V. S. Lakshmanan, and Raymond T. Ng. ***The 3W Model and Algebra for Unified Data Mining***. 2000 (cit. on p. 38).

[Kat12] Ilya Katsov. ***MapReduce Patterns, Algorithms, and Use Cases***. Feb. 2012 (cit. on p. 120).

[Kim+13] Chungrim Kim et al. **"Influence Maximization Algorithm Using Markov Clustering"**. In: *Database Systems for Advanced Applications*. Ed. by Bonghee Hong et al. Vol. 7827. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 112–126 (cit. on pp. 13, 14, 77).

[LD10] Jimmy Lin and Chris Dyer. ***Data-Intensive Text Processing with MapReduce***. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2010 (cit. on pp. 115, 120).

[Luc+13] Antoine Lucas et al. ***gmp: Multiple Precision Arithmetic***. R package version 0.5-11. 2013 (cit. on p. 90).

[LY10] Liang Liu and Lili Yu. **"Phybase: an R package for species tree analysis."** In: *Bioinformatics* 26.7 (Apr. 13, 2010), pp. 962–963 (cit. on p. 58).

[LZ12] Bing Liu and Lei Zhang. **"A Survey of Opinion Mining and Sentiment Analysis"**. English. In: *Mining Text Data*. Ed. by Charu C. Aggarwal and ChengXiang Zhai. Springer US, 2012, pp. 415–463 (cit. on pp. 97–99).

[Mae13] Martin Maechler. ***Rmpfr: R MPFR - Multiple Precision Floating-Point Reliable***. R package version 0.5-4. 2013 (cit. on p. 90).

[Mat11] Norman S. Matloff. ***Art of R programming: A Tour of Statistical Software Design***. San Francisco, Calif.: No Starch Press, 2011 (cit. on pp. 14, 99, 116).

[MH13] David Meyer and Kurt Hornik. ***relations: Data Structures and Algorithms for Relations***. R package version 0.6-2. 2013 (cit. on p. 99).

[MM06] Matteo Magnani and Danilo Montesi. **"A unified approach to structured and XML data modeling and manipulation"**. In: *Data Knowl. Eng.* 59.1 (Oct. 2006), pp. 25–62 (cit. on pp. 26, 38).

[MM12] Matteo Magnani and Danilo Montesi. **"Joining relations under discrete uncertainty"**. In: *CoRR* abs/1211.0176 (2012) (cit. on p. 36).

[MR13a] Matteo Magnani and Luca Rossi. **"Formation of Multiple Networks"**. In: *Social Computing, Behavioral-Cultural Modeling and Prediction*. Ed. by ArielM Greenberg, WilliamG Kennedy, and NathanD Bos. Vol. 7812. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 257–264 (cit. on p. 12).

[MR13b] Matteo Magnani and Luca Rossi. **"Pareto Distance for Multi-layer Network Analysis"**. In: *Social Computing, Behavioral-Cultural Modeling and Prediction*. Ed. by ArielM Greenberg, WilliamG Kennedy, and NathanD Bos. Vol. 7812. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 249–256 (cit. on p. 12).

[Mur13] Paul Murrell. ***hyperdraw: Visualizing Hypergaphs***. R package version 1.10.0. 2013 (cit. on p. 99).

[MW11] Q. Ethan McCallum and Stephen Weston. ***Parallel R***. O'Reilly Media, Inc., 2011 (cit. on pp. 116, 131).

[NTK11] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. **"A Three-Way Model for Collective Learning on Multi-Relational Data."** In: *ICML*. Ed. by Lise Getoor and Tobias Scheffer. Omnipress, 2011, pp. 809–816 (cit. on p. 66).

[Nuu95] Esko Nuutila. **"Efficient Transitive Closure Computation in Large Digraphs"**. PhD thesis. Helsinki University of Technology, 1995 (cit. on p. 76).

[Odi92] P. Odifreddi. ***Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers (Studies in Logic and the Foundations of Mathematics)***. New Ed. North Holland, Feb. 1992 (cit. on p. 15).

[Pra13] Vignesh Prajapati. ***Big Data Analytics with R and Hadoop***. Packt Publishing, 2013 (cit. on p. 122).

[Pre08] Roger S. Pressman. ***Principi di Ingegneria del software***. McGraw-Hill, 2008 (cit. on p. 127).

[Rod12] Marko A. Rodriguez. ***Graph Degree Distributions using R over Hadoop***. Feb. 2012 (cit. on p. 115).

[Sam+13] Nagiza F. Samatova et al. ***Practical Graph Mining With R***. CRC Press, 2013 (cit. on pp. 52, 55, 56, 61, 62, 87, 115).

[Sha13] Cosma Rohilla Shalizi. **"Advanced Data Analysis from an Elementary Point of View"**. http://www.stat.cmu.edu/cshalizi/ADAfaEPoV/. 2013 (cit. on p. 60).

[SJY08] Liang Sun, Shuiwang Ji, and Jieping Ye. **"Hypergraph Spectral Learning for Multi-label Classification"**. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '08. Las Vegas, Nevada, USA: ACM, 2008, pp. 668–676 (cit. on pp. 28, 61, 62).

[SM58] R. R. Sokal and C. D. Michener. **"A statistical method for evaluating systematic relationships"**. In: *University of Kansas Scientific Bulletin* 28 (1958), pp. 1409–1438 (cit. on p. 58).

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. ***Introduction to Data Mining***. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005 (cit. on pp. 12, 13, 45, 109).

[TTM11] ICHIGAKU TAKIGAWA, KOJI TSUDA, AND HIROSHI MAMITSUKA. **"Mining Significant Substructure Pairs for Interpreting Polypharmacology in Drug-Target Network"**. In: *PLoS ONE* 6.2 (Feb. 2011), e16999+ (cit. on pp. 14, 60).

[Van09] ASHLEE VANCE. **"Data Analysts Captivated by R's Power"**. In: *New York Times* (Jan. 2009) (cit. on p. 14).

[Vaz09] ALEXEI VAZQUEZ. **"Finding hypergraph communities: a Bayesian approach and variational solution"**. In: *Journal of Statistical Mechanics: Theory and Experiment* 2009.07 (July 2009), P07006+ (cit. on pp. 27, 61).

[W+05] MARC WÖRLEIN ET AL. **"A Quantitative Comparison of the Subgraph Miners Mofa, Gspan, FFSM, and Gaston"**. In: *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*. PKDD'05. Porto, Portugal: Springer-Verlag, 2005, pp. 392–403 (cit. on p. 55).

[Whi09] TOM WHITE. ***Hadoop: The Definitive Guide***. 1st. O'Reilly Media, Inc., 2009 (cit. on pp. 118, 122, 123).

[WNK10] NIKIL WALE, XIA NING, AND GEORGE KARYPIS. **"Trends in Chemical Graph Data Mining."** In: *Managing and Mining Graph Data*. Ed. by Charu C. Aggarwal and Haixun Wang. Vol. 40. Advances in Database Systems. Springer, 2010, pp. 581–606 (cit. on pp. 52, 58).

[Woo12] PETER T. WOOD. **"Query languages for graph databases"**. In: *SIGMOD Rec.* 41.1 (Apr. 2012), pp. 50–60 (cit. on p. 40).

[Wu+07] XINDONG WU ET AL. **"Top 10 Algorithms in Data Mining"**. In: *Knowl. Inf. Syst.* 14.1 (Dec. 2007), pp. 1–37 (cit. on p. 44).

[YH02] XIFENG YAN AND JIAWEI HAN. **"gSpan: Graph-Based Substructure Pattern Mining."** In: *ICDM*. IEEE Computer Society, 2002, pp. 721–724 (cit. on pp. 52, 54, 55).

[YKSP13] TAO SHI YU-KENG SHIH SUNGMIN KIM AND SRINIVASAN PARTHASARATHY. **"Directional Component Detection via Markov Clustering in Directed Networks"**. In: *Eleventh Workshop on Mining and Learning with Graphs*. 2013 (cit. on p. 54).

[Zac77] W.W. ZACHARY. **"An information flow model for conflict and fission in small groups"**. In: *Journal of Anthropological Research* 33 (1977), pp. 452–473 (cit. on pp. 13, 53).

[Zha13] YANCHANG ZHAO. ***R and Data Mining: Examples and Case Studies***. 1st. Elsevier, 2013 (cit. on pp. 14, 99, 109, 115).

D.2 FURTHER READING

[AH08] Dean Allemang and James Hendler. ***Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL***. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[AJ11] Jesús M. Almendros-Jiménez. **"A Prolog-based Query Language for OWL"**. In: *Electron. Notes Theor. Comput. Sci.* 271 (Mar. 2011), pp. 3–22.

[AW10] Charu C. Aggarwal and Haixun Wang. ***Managing and Mining Graph Data***. 1st. Springer Publishing Company, Incorporated, 2010.

[Baa+03] Franz Baader et al., eds. ***The description logic handbook: theory, implementation, and applications***. 2nd. New York, NY, USA: Cambridge University Press, 2003.

[Ben+08] M. Bender et al. **"Exploiting social relations for query expansion and result ranking"**. In: *Data Engineering for Blogs, Social Media, and Web 2.0, ICDE 2008 Workshops*. 2008, pp. 501–506.

[CH06b] Diane J. Cook and Lawrence B. Holder. ***Mining Graph Data***. John Wiley & Sons, 2006.

[Col12] Marco Colombetti. **"The Description Logic SROIQ(D)"**. Lecture Notes for the Knowledge Enginerring Course. 2012.

[CYH10] Hong Cheng, Xifeng Yan, and Jiawei Han. **"Mining Graph Patterns"**. English. In: *Managing and Mining Graph Data*. Ed. by Charu C. Aggarwal and Haixun Wang. Vol. 40. Advances in Database Systems. Springer US, 2010, pp. 365–392.

[DGK09] Mike Dean, Benjamin Grosof, and MIchael Kifer. ***The SILK Language***. Dean, Mike, 2009.

[Dun02] M.H. Dunham. **"Data Mining: Introductory and Advanced Topics"**. In: Prentice Hall, 2002. Chap. 5, pp. 125–145.

[FG] Seth Falcon and Robert Gentleman. ***hypergraph: A package providing hypergraph data structures***. R package version 1.34.0.

[GSP13] Steve H. Garlik, Andy Seaborne, and Eric Prud'hommeaux. ***SPARQL 1.1 Query Language***. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/. 2013.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. **"The even more irresistible SROIQ"**. In: *In KR*. AAAI Press, 2006, pp. 57–67.

[Hua98] Zhexue Huang. **"Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values"**. In: *Data Min. Knowl. Discov.* 2.3 (Sept. 1998), pp. 283–304.

[Jin+11] Ruoming Jin et al. **"Path-tree: An efficient reachability indexing scheme for large directed graphs"**. In: *ACM Trans. Database Syst.* 36.1 (Mar. 2011), 7:1–7:44.

[KJ07] KRYS J. KOCHUT AND MACIEJ JANIK. **"SPARQLeR: Extended SPARQL for Semantic Association Discovery"**. In: *Proc. of the 4th European Semantic Web Conference (ESWC*. 2007, pp. 145–159.

[MM10] MATTEO MAGNANI AND DANILO MONTESI. **"A Survey on Uncertainty Management in Data Integration"**. In: *J. Data and Information Quality* 2.1 (2010).

[NM95] ULF NILSSON AND JAN MALUSZYNSKI. ***Logic, Programming, and PROLOG***. 2nd. New York, NY, USA: John Wiley & Sons, Inc., 1995.

[Rud11] SEBASTIAN RUDOLPH. **"Foundations of Description Logics"**. In: *Reasoning Web. Semantic Technologies for the Web of Data*. Ed. by Axel Polleres et al. Vol. 6848. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 76–136.

[Sce+10] SALVATORE SCELLATO ET AL. **"Distance Matters: Geo-social Metrics for Online Social Networks"**. In: *Proceedings of the 3rd Conference on Online Social Networks*. WOSN'10. Boston, MA: USENIX Association, 2010, pp. 8–8.

[Spi12] DANIEL SPIELMAN. **"Spectral Graph Theory"**. In: *Combinatorial Scientific Computing*. Ed. by Uwe Naumann and Olaf Schenk. CRC Press, 2012. Chap. 18, pp. 495–524.

[SS09] STEFFEN STAAB AND RUDI STUDER. ***Handbook on Ontologies***. 2nd. Springer Publishing Company, Incorporated, 2009.

[TL07] SILKE TRISSL AND ULF LESER. **"Fast and practical indexing and querying of very large graphs"**. In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. SIGMOD '07. Beijing, China: ACM, 2007, pp. 845–856.

[WRF13] YUYI WANG, JAN RAMON, AND THOMAS FANNES. **"An efficiently computable subgraph pattern support measure: counting independent observations."** In: *Data Min. Knowl. Discov.* 27.3 (2013), pp. 444–477.

[YYH04] XIFENG YAN, PHILIP S. YU, AND JIAWEI HAN. **"Graph indexing: a frequent structure-based approach"**. In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. SIGMOD '04. Paris, France: ACM, 2004, pp. 335–346.

[ZA03] MOHAMMED J. ZAKI AND CHARU C. AGGARWAL. **"XRules: an effective structural classifier for XML data"**. In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '03. Washington, D.C.: ACM, 2003, pp. 316–325.

[Has+07] MOHAMMAD HASAN ET AL. **"ORIGAMI: Mining Representative Orthogonal Graph Patterns"**. en. In: *Proc. IEEE Int. Conf. on Data Mining ICDM'07*. Ed. by IEEE Computer Society Press. Oct. 2007, pp. 153–163.

INDEX