DB2SNA: an All-in-one Tool for Extraction and Aggregation of underlying Social Networks from Relational Databases

Rania Soussi[1,3], Etienne Cuvelier[1], Marie-Aude Aufaure[1,2], Amine Louati[2,3], and Yves Lechevallier[2]

[1] Ecole Centrale Paris, MAS Laboratory, Business Intelligence Team, Grande Voie des Vignes 92 295 Chatenay-Malabry
{rania.soussi, etienne.cuvelier, Marie-Aude.Aufaure}@ecp.fr
[2] INRIA Paris-Rocquencourt, Axis Team, Domaine de Voluceau 78150 Rocquencourt
{Yves.Lechevallier, Marie-Aude.Aufaure, Amine.Louati}@inria.fr
[3] ENSI, RIADI-GDL Laboratory, Campus Universitaire de la Manouba, 2010, Tunisia
Amine.louati@riadi.rnu.tn

Abstract. In the enterprise context, People need to visualize different types of interactions between heterogeneous objects (e.g. product and site, customers and product, people interaction (social network)...). The existing approaches focus on social networks extraction using web document. However a considerable amount of information is stored in relational databases. Therefore, relational databases can be seen as rich sources for extracting a social network. The extracted network has in general a huge size which makes it difficult to analyze and visualize. An aggregation step is needed in order to have more understandable graphs. In this chapter, we propose a heterogeneous object graph extraction approach from a relational database and we present its application to extract social network. This step is followed by an aggregation step in order to improve the visualisation and the analyse of the extracted social network. Then, we aggregate the resulting network using the k-SNAP algorithm which produces a summarized graph.

Keywords: relational database, graph database, social network analysis, graph extraction and aggregation

1 Introduction

The data manipulated in an enterprise context are structured data as well as unstructured data such as e-mails, documents, etc. Graphs are a natural way of representing and modeling such data in a unified manner (structured semi-structured and unstructured ones). The main advantage of such structure relies on (or resides in) its dynamic aspect and its capability to represent relations,

even multiple ones, between objects. It also facilitates data query using graph operations. Explicit graphs and graph operations allow a user to express a query at a very high level of abstraction.

People need to visualize different types of interactions between heterogeneous objects (e.g. product and site, customers and product, people interaction like social networks, etc.).

In order to analyze these interactions and facilitate their querying using graph query languages and social network analysis methods, it is relevant to modulate such interaction by using a graph structure.

Indeed, these different graphs can help enterprises sending product recommendations (using the graph of Products and client), finding experts (using social network), etc.

Nevertheless, in a business context, important expertise information is stored in files, databases and especially relational databases. Relational database pervades almost all businesses. Many kinds of data, from e-mails and contact information to financial data and sales records, are stored in databases. Also, databases used in business contain information about all people, objects and processes related to the enterprise. This data source is a rich one to extract object interaction.

However, a relational database is not the most suited data structure to store the "graph-like" information about a social network. By nature, a graph database is more preferable, because its structure is close to the structure of a social network.

Then, this chapter presents a new approach of social network extraction from relational database which allows discovering hidden relationships between entities. This approach has been generalized to extract different kind of heterogeneous objects graphs: such graph contains several kinds of relations and objects. Each object owns a set of characteristics which can be different from object to another. In order to facilitate the visualization and data interpretation, it seems interesting to aggregate the extracted graphs. This aggregation should use not only the relations between nodes but also the characteristics of each one and very few algorithms do that. In this context, we use the aggregation algorithm K-SNAP.

Then, the structure of this chapter is the following. First, we propose in section 2, as a pre-treatment, to migrate the social network information stored in a classical relational database towards a graph database model . Starting from this new representation of the social network information contained in the original databases, we extract the social network structure, but it would be a pity to lose a lot of information using the classical graph representation of a social network. Indeed, in the graph theory used to model such networks, all nodes are of the same type, and all relationships are of the same kind. But, in real life, all people in a social network do not play the same role, and all relationships dot not necessary share the same type. In other words, we are not all friends, or only friends, with our neighbors or colleagues. The enterprise framework is a perfect example for this: accountants do not have the same relationship with

their accountant colleagues, with the salesmen, with the workers and, finally with their manager. And all these people play, of course, different roles in the business of their enterprise. That is why we propose in section 3 to use heterogeneous graphs to model and extract such complex social networks, by working directly on the resulting graph database. The extraction of the social network is made using graph transformations. In a last step, we propose a visualization process because the "raw data" of a network can have, in general, a huge size which makes its difficult to analyze and visualize. In order to ease the latter tasks we propose to use an aggregation process in section 4. After a study of the existing graph aggregation methods, we propose to use an existing technique which takes into account the heterogeneity of our networks. A global view of our all-in-one solution is given in Figure 1. Finally, we conclude and give some perspectives to this work.

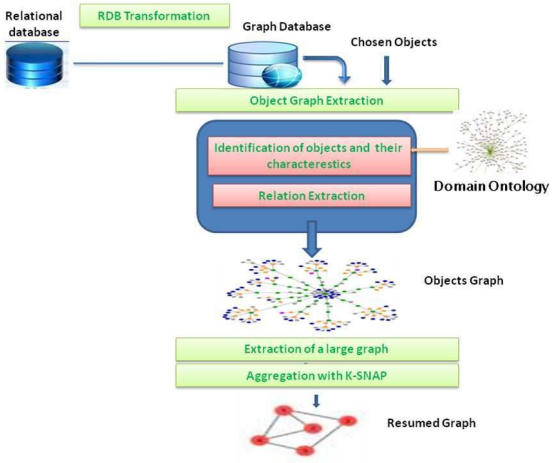


Fig. 1. Graph extraction approach.

2 Graph database Models and Graph Aggregation Algorithm

2.1 Graph database Models

A graph database is defined [13] as a "database where the data structures for the schema and/or instances are modeled as a (labeled) (directed) graph, or generalizations of the graph data structure, where data manipulation is expressed by graph-oriented operations and type constructors, and has integrity constraints appropriate for the graph structure. There is a variety of models for a graph database (for more details see [13]). All these models have their formal foundation

as variations of the basic mathematical definition of a graph. The structure, used for modeling entities and relations, influences the way that data is queried and visualized. In this section, we compare existing models in order to find the one most suitable one for storing and representing a social network. We will focus on the representation of entities and relations in these models. In the following, we present some models classified according to the data structure used to model entities and relations.

Models based on simple node Data are represented in these models by a (directed or undirected) graph with simple nodes and edges. Most of these models (GOOD [14]), GMOD [15], etc.) represent both schema and instance database as a labeled directed graph. Moreover, LDM [16] represents the graph schema as a directed graph where leaves represent data and whose internal nodes represent connections between the data. LDM instances consist of two-column tables, one for each node of the schema. Entities in these model are represented by nodes labeled with type name and also with type value or object identifier (in the case of instance graph). Some models have nodes for explicit representation of tuples and sets (PaMaL[17], GDM [18]), and n-ary relations (GDM). Relations (attributes, relations between entities) are generally represented in these models by means of labeled edges. LDM and PaMaL use tuple nodes to describe a set of attributes which are used to define an entity. GOOD defines edges to distinguish between mono-valued (functional edge) and multi-valued attributes (nonfunctional edge). Nevertheless, these models do not allow the presentation of nested relations and are not very well suited for complex objects modeling.

Models based on complex node. In these models, the basic structure of a graph (node and edge) and the presentation of entities and relations are based on hypernodes (and hypergraphs). Indeed, a hypernode is a directed graph in which nodes themselves can be graphs (or hypernodes). Hypernodes can be used to represent simple (flat) and complex objects (hierarchical, composite, and cyclic) as well as mappings and records. A hypergraph is a generalized notion of graph where the notion of edge is extended to hyperedge, which relates to an arbitrary set of nodes. The Hypernode Model [19] and GGL[20] emphasize the use of hypernodes for representing nested complex objects. GROOVY [21] is focused on the use of hypergraphs. The hypernode model is characterized by using nested graphs at the schema and instance levels. GGL introduces, in addition to its support for hypernodes (called Master-nodes), the notion of Master-edge for the encapsulation of paths. It uses hypernodes as an abstraction mechanism consisting in packaging other graphs as an encapsulated vertex, whereas, the Hypernode model additionally uses hypernodes to represent other abstractions like complex objects and relations. Most models have explicit labels on edges. In the hypernode model and GROOVY, labeling can be obtained by encapsulating edges, that represent the same relation, within one hypernode (or hyperedge) labeled with the relation name.

Discussion The purpose of this review of graph database models is to find the most suited one to model many complex data objects and their relationships, such as social networks. Social Network is an explicit representation of relationships between people, groups, organizations, computers or other entities [22]. As other networks, it can be represented as a complex graph [23].

Indeed, the social network structure can contain one or more types of relations, one or more types or levels of entities and many attributes over the entities. This structure is dynamic due to growth of the volume, change of attributes and relations.

Then, we compare the previous graph database models using some characteristics related to social network: the ability to present dynamic and complex objects, nested and neighborhood relations and the ability to give a good visualization of social network. We resume the comparison on Table 1 where "+" indicates the graph model support, "-" indicates that the graph model does not support and "+/-" partial support. From this comparison, we have concluded that models based on hypernodes can be very appropriate to represent complex and dynamic objects. In particular, the hypernode model with its nested graphs can provide an efficient support to represent every real-world object as a separate database entity. Moreover, models based on a simple graph are unsuitable for complex networks where entities have many attributes and multiple relations.

	Entity		Relation		Visualization
	Complex	Dynamic	Nested	Neighborhood	
Hypernode	+	+	+	+	+
Groovy	+	+	+	+	-
GGL	+	+	+	+	-
GOOD	-	+	-	-	+
GMOD	-	+	-	-	+
PaMaL	+	+	-	+	+/-
GDM	+	+	-	-	+
LDM	+	+	-	-	-

Table 1. Graph database model comparison

2.2 Graph Aggregation Algorithms

When graphs of extracted social networks are large, effective graph aggregation and visualization methods are helpful for the user to understand the underlying information and structure. Graph aggregations produce small and understandable summaries and can highlights communities in the network, which greatly facilitates the interpretation.

The automatic detection of communities in a social network, can provide this kind of graph aggregation. The community detection is a clustering task, where a *community* is a cluster of nodes in a graph [4] [31], such the nodes of

the cluster must be more connected with inside nodes, than with nodes outside of the cluster (see found [5] and [6] for extended reviews).

The first class of clustering algorithms are the *partitional algorithms*, which try to find a partition of a set of data, with a given number of clusters, using jointly, most of the times, similarity or a dissimilarity **measures** and a quality criterion of the found partition. The most popular partitional algorithm (with several variants), the *k-means clustering* [7], tries to find a partition of the set of objects which minimizes the *sum-of-square* criterion which adds the dissimilarities from each object to the centre of its own cluster. Several (di)similarity measures can be defined in the social network context, like those based on the *Jaccard index*, which measures similarity between the sets of *neighbours* of the two nodes, but other measures can be defined ([5] and [6]).

Hierarchical clustering algorithms try to organize data into a hierarchical structure, and are divided into *agglomerative* and *divisive* algorithms, depending on whether the partition is coarsened, or refined, at each iteration. The basic idea beyond *agglomerative algorithms* is simple: at the starting point, the objects to cluster are their own classes, and then at each stage we merge the two more similar clusters. Of course a dissimilarity measure between two clusters is mandatory, and for a given dissimilarity measure d between objects, several cluster-dissimilarities exist. The result of the clustering process is a *dendrogram*, which can be cut to give one single partition. *Divisive clustering algorithms,* split the dataset iteratively or recursively into smaller and smaller clusters, with respect to a quality criterion. The most popular method for divisive hierarchical clustering of social networks uses the notion of edge betweenness [8], because finding the connecting edges between communities is also finding these communities. The algorithm given in [4] splits the network into clusters by removing, step after step, the edge with the higher betweenness value. The use of a stopping criterion which measures the improvement at each step should permit to stop when no improvement is gained with an iteration. In most cases the *modularity* [9] is used. SuperGraph [28] employs hierarchical graph partitioning to visualize large graphs.

Specially designed for graphs, *spectral algorithms* [10] are based on the notion of connected components. These algorithms work with a Laplacian, matrix based onthe adjacency (or weight) matrix [11] [12]. If the graph of the social network contains k , completely disjoints communities (i.e. without any link between them), called *connected components*, then the k *eigenvectors* associated to the eigenvalue 0 are indicator vectors of the k connected components . If the clusters of the social network do not contain "clean" connected components (i.e. if there are links between existing communities), then a simple clustering on the k *eigenvectors* associated to the k least eigenvalues, can retrieve the k communities.

Some other algorithms works on graph aggregation use statistical methods to study graph characteristics, such as degree distributions [26], hop-plots [27] and clustering coefficients [30]. The results are often useful but difficult to control and especially to exploit. Methods for mining frequent graph patterns [32] are

also used to understand the characteristics of large graphs. Washio and Motoda [26] provide an elegant review on this topic.

However, all the previous algorithms use only on links between nodes of the graph of the network, and do not take into account the internal values contained in each node, while classical clustering algorithms applied on tables of values, work only on these values ignoring completely the possible link between individual. An algorithm which can take into account both kind of information would be very valuable. Designed for graphical graph aggregation the k-SNAP algorithm [29], in its divisive version, begins with a grouping based on attributes of the nodes, and then tries to divide the existing groups thanks to their neighbours groups, trying minimizing a loss information measure.

3 Social Network extraction from relational databases

In this section we describe our approach of graph extraction which is based on two steps. The first step to perform is to transform relational databases into graph databases according to a graph model.

This transformation allows the extraction of all the entities in the relational database on the form of nodes and outlines the relations between them which facilitate, in further steps, the selection of the desired entities. Also, nodes in graph database are more complex than a simple graph which can encapsulate all the attribute of entities in the same node and give us a simple graph of entities.

The second step is to define a method to transform the graph according to chosen entities. This method has to deal with the identification of entities of interest for a particular user and to reorganize the graph – nodes and relationships – according to this point of view. Then, we applied this approach to extract social network.

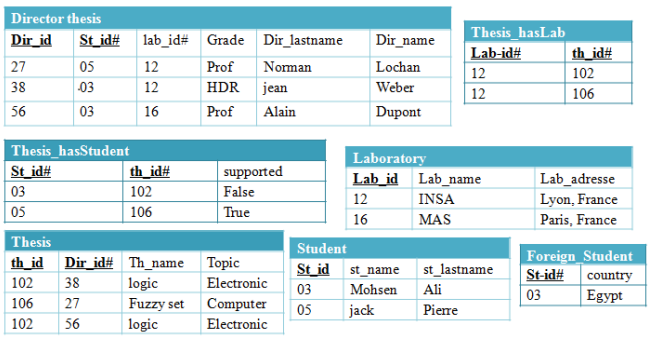
3.1 Graph extraction

The graph extraction approach is based on two main steps: (1) converting the relational database into graph database and (2) Extracting the heterogeneous graph (with chosen entities) from the graph database.

Converting relational database into hypernode database Having a graph database instead of a relational database will provide a clearer view of existing objects in the initial database. Indeed, all these objects will be presented in the form of nodes, and the relations between them will be outlined thus facilitating the selection of the desired objects of interest in a further step. In addition, nodes in a graph database can encapsulate all the attributes of objects in the same node and give us a simple graph of objects.

Using the comparison between existing graph database models (Table 1), we have chosen to work with the hypernode model [19] because the hypernode database with its nested graphs can provide an efficient support to represent each real-world object as a separate database entity.

The transformation of a relational database into a graph database includes schema translation and data conversion [24]. The schema translation can turn the source schema into the target schema by applying a set of mapping rules. In our work, we propose a translation process which directly transforms the relational schema into a hypernode schema. A data conversion process converts data from the source to the target database based on the translated schema. Data stored as tuples (rows) in the relational database are converted into nodes and edges in the graph database. This involves unloading and restructuring relational data, and then reloading them into a target database in order to populate the schema generated during the translation process. The main advantages of this transformation are (1) to discover underlying graphs of objects from relational databases, taking into account the implicit relations expressed by the means of primary and foreign keys and (2) to model data in a more flexible way (objects can easily be added or removed in a graph). The reader interested by details about this approach can refer to [25]. The resulting hypernode database schema



Director thesis

Dir_id	St_id#	lab_id#	Grade	Dir_lastname	Dir_name
27	05	12	Prof	Norman	Lochan
38	03	12	HDR	jean	Weber
56	03	16	Prof	Alain	Dupont

Thesis_hasLab

Lab-id#	th_id#
12	102
12	106

Thesis_hasStudent

St_id#	th_id#	supported
03	102	False
05	106	True

Laboratory

Lab_id	Lab_name	Lab_adresse
12	INSA	Lyon, France
16	MAS	Paris, France

Thesis

th_id	Dir_id#	Th_name	Topic
102	38	logic	Electronic
106	27	Fuzzy set	Computer
102	56	logic	Electronic

Student

St_id	st_name	st_lastname
03	Mohsen	Ali
05	jack	Pierre

Foreign_Student

St-id#	country
03	Egypt

Fig. 2. Relational database

(Fig. 3) is composed by two sets H_b and R_b built from the sets of hypernodes H and relations R. The first set $H_b := \{(h, N_h), h \in H\}$is defined by:

- h denotes the name of H,
- N_h denotes a set of nodes $N_h := \{n_h | n_h := \langle n, t \rangle\}$ where n is the node name, t is the type. t is a predefined type (Integer, String,...) or a H element.

The second one $R_b := \{\langle r, h_s, h_d \rangle, r \in R, h_s, h_d \in H\}$:

- r denotes the name of R,
- h_s denotes the hypernode source name
- h_d denotes the hypernode destination name

From the relational database tables (Fig. 2), we extract six hypernodes (Fig. 3) : $Thesis, Laboratory, Thesis\_hasStudent, Student, Director\_thesis$ and $Foreign_Student$.

The table $Thesis\_hasLab$ is transformed into a relation because it contains only foreign keys. The hypernode Thesis:

$$\begin{aligned} Thesis = (\text{“}Thesis\text{”}, \{ &< Th\_id, Integer >, \\ &< Th\_name, String >, \\ &< Topic, String >, \\ &< Dir\_id >, \\ &< Director\_thesis > \}) \end{aligned}$$

has three nodes $(Th\_id, Th\_name, Topic)$, with predefined types and one node $Dir - id$ having $Director\_thesis$ as type because it is a foreign key exported from $Director\_thesis$ hypernode. Thesis has the relation

$$< \text{“}Part\_of\text{”}, Thesis, Thesis\_hasStudent >$$

with the hypernode $Thesis\_hasStudent$. Then, for each hypernode in the hyper-

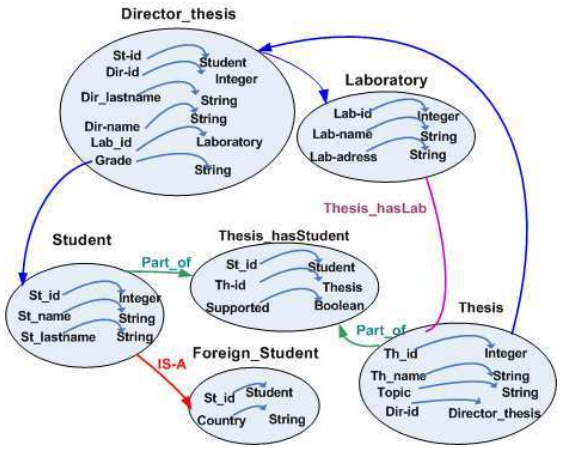


Fig. 3. Hypernode database schema

node database, a set of instances hypernode HI is extracted from the relational tuples.

The set of instances hypernode $\mathcal{H}$ is defined by $\mathcal{H} = \{h_i | h_i := \langle h, l_i, N_i \rangle\}$ where:

- $h_i$denotes the instance hypernode,
- h denotes the hypernode source name,
- l_i denotes the name of H_i,
- N_i denotes a set of nodes $N_i := \{n | n := \langle l_n, t_n, val_n \rangle\}$ where l_n is the label of the node, t_n is the type, and val_n mentions the node value.

For example , $Thesis\_1$ is an instance of $Thesis$ and is defined by:

$$Thesis\_1 = \langle Thesis, Thesis\_1\ , \{ < Th\_id, Integer, 102 >, \\ < Th\_name, String, '' Logic'' >, \\ < Topic, String, '' Electronic'' >, \\ < Dir\_id, Director\_thesis, Director\_thesis\_1 > \}\rangle$$

For each relation in R_b, a set of instance relations R is extracted using the value

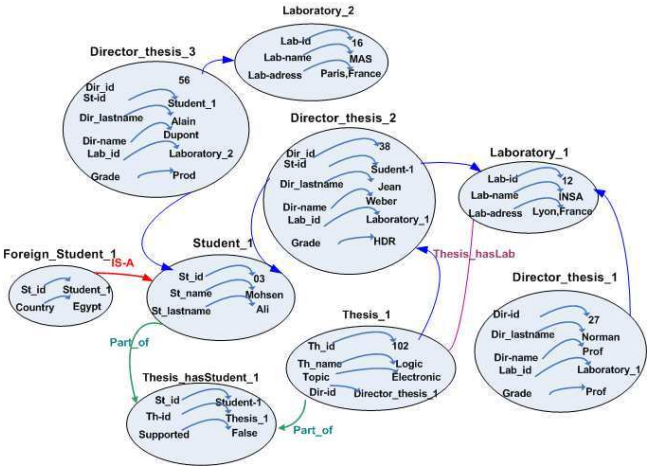


Fig. 4. Part of the *Hypernode* database instance.

of keys on the relational tables. RI is defined by $R := \{r_i | r_i := \langle r, h_{s_i}, h_{d_i} \rangle\}$ where:

- r denotes the relation which is instanciated by r_i,
- h_{s_i} denotes the hypernode source instance,
- h_{d_i} denotes the hypernode destination instance,

Finally, transformed data are loaded into the hypernode database schema. An excerpt of the hypernode database instance is shown in Fig. 4.

Converting the graph database to a graph containing *objects of interest* (from a user point of view) The Hypernode database represents the graph of objects. From this graph, we may apply transformation rules according to the user's interest (the set of objects the user would like to see their interaction).

The graph containing the objects of interest is defined by: $GO = (O_I, R_O)$ where:

- O_I is a finite set of object such $O_I = \{o_I | o_I \in \mathcal{H}\}$,
- R_O is a finite set of relations between objects such $R_O := \{r | r := \langle l, o_{I_1}, o_{I_2} \rangle, o_{I_1}, o_{I_2} \in O_I\}$ where l is the relation name.

Transforming the graph leads to cope with two main problems: objects of interest (named *objects* in the following) identification and relations extraction and transformation.

Objects identification Object identification is the process used to identify hypernodes that contain the elements of interest for the user (for example Laboratory or Student). These objects will constitute the nodes of the transformed graph according to the user's point of view.

The identified objects have the type chosen by the user, e.g. persons, organization, process, etc.

Many problems occur at this step. First, an object can be described by the mean of different tables in the relational database, and then many hypernodes can represent the same object. Second, the names of the hypernodes are not all the time significant.

Each object has a number of characteristics which help to identify it. In order to identify the chosen objects in the graph database, a domain ontology is used. In fact, ontology contains concepts and relationships that describe a domain.

Then, we use an ontology having the same domain than the initial database. For instance, we use an enterprise ontology if the initial relational database is an enterprise database. This ontology, which describes the objects and their relations, is built semi-automatically by using information collected from domain documents.

The proposed approach is based on three main phases:

- Building minimal enterprise ontology from scratch (manually) using existing enterprise models and pattern,
- Learning ontology from web document: Global enterprise ontology is learned from enterprise websites and using the minimal ontology,
- Population and enrichment of the generic ontology: more specific enterprise ontology is obtained by enriching the enterprise ontology.

The hypernode set in the hypernode database schema is analyzed, considering ontology concepts, and more specifically concepts related to the objects chosen by the user. If the hypernode contains some characteristics related to the desired object, it will be selected to be one of the objects in the transformed graph.

After identifying the hypernodes that contain objects of interest for the user, from the hypernode schema, we add their hypernodes instances to O_I.

Relation construction After the objects identification step, we define relations (edges) between identified entities. The identified objects are instance hypernodes. In the hypernode database, instance hypernodes can share relations. Then, we try to use the existing relations and find hidden ones.

In our process to transform the relational database into a hypernode database [25], we have defined four types of relations: *IS-A*, *Part-of*, *dependency with known name* (using the initial relational tables containing only foreign keys), *dependency with unknown name*.

Identified objects can be related by these exiting relations. In this step we try to find other hidden relations between these objects.

In order to facilitate this task, we define a set of relation patterns (Table. 2) using the schema of the hypernode database. These patterns are used to create the objects relations. A pattern relation P_r is defined by $P_r = \langle n, o_{I_1}, o_{I_2}, o_m \rangle$ such as n is the name of the relation, o_{I1} and o_{I2} the entities which share the relation, o_m a mediator for this relation (the hypernode used to find the new relation).

Indeed, these patterns will be used to find relations between the identified objects: existing relations objects already share in the hypernode database and new relations (build using key value and mediator objects). A mediator is an object facilitating the communication between two other objects (acting as a router). After having identified relations and objects (of interest), we build the transformed graph corresponding to chosen objects. In the next section, we present a use case experimentation corresponding to a social network perspective extracted from an actual relational database.

3.2 Social network extraction using graph transformation

In this section, we present an experiment to transform a graph extracted from a relational database into another one according to a social network perspective. A social network can be represented as a graph [23], where the nodes represent people and the edges represent relationships among people, such as values, visions, idea, financial exchange, friends, kinship, conflict, trade, web links, airline routes, etc.

As a consequence, the resulting structures are often very complex. Choosing the right methods and techniques of information extraction requires having a rich and high quality source of information.

The existing approach of social network extraction uses just web data like: e-mail messages [1], Friend-of-a-Friend (FOAF) documents [3], and observing face-to face communications [2].

For example, Flink [3] uses four different types of knowledge sources: HTML pages, FOAF profiles, public collections of emails and bibliographic data. Flink employs a co-occurrence analysis technique to extract the social network from this data. However, these existing approaches are designed only to extract social network from web data and are not able to use other rich sources like relational databases.

Indeed, in the context of enterprise and business data are principally stored into a database. Enterprise databases contain information about people, objects manipulated in the enterprise and the associated processes. The objective is to highlight all this information and the relationships between people, objects and processes for a better performance in the enterprise.

Initial Relation	Pattern	Process and description
$R_1 := \langle$"$IS-A$", $h_s, h_d\rangle$ where h_s or h_d instances $\in O_I$	----	h_s or h_d instances are added to O_I
$R_2 := \langle r, h_s, h_d\rangle$ where h_s and h_d instances $\in O_I$	$\Pr_1 := \langle r, h_s, h_d, null\rangle$	Find all the existing relations between h_s and h_d instances then add them to R_O
	$Pr_2 :=< Same\_(h_d.name), o_{Ii}, o_{Ij}, hd >$ where $o_{Ii} = h_{is}$, $o_{Ij} = h_{is}$ and $i! = j$	Find all the h_s instances which have relations with a same h_d instances and link them with a new relation "$Same\_(hd.name)$"
$R_3 := \langle r, h_s, h_d\rangle$ where h_s instances $\in OI$ (finite set of entities) and $h_d \notin O_I$	$Pr_3 :=< Same\_(h_d.name), o_{Ii}, o_{Ij}, h_d >$ where $o_{Ii} = h_{is}$, $o_{Ij} = h_{is}$, $i! = j$ and $r!$ = "Part-of"	Find all the h_s instances which have relations with a same h_d instances and link them with a new relation
	$Pr_4 :=< Same\_h_j.name, o_{I1}, o_{I2}, h_j >$ where $o_{I1} = h_s$, $o_{I2} = h_s$, r ="Part-of" and $h_j \in \{h \mid h$ has the relation $R_h :=<$ "Part-of", h_j, $h_d >\}$	-Find all the hypernodes h_j having a "Part-of" relation with h_d such as $R :=<$"Part-of", h_j, $h_d >$ - add new node to hs containing the name of h_j - then the pattern Pr_4 is applied: Find all the h_d instances which have relations with a same h_j instances and link them with a new relation
$R4 := \langle r, h_s, h_d\rangle$ where $h_s \notin O_I$ and $h_d \in O_I$	$Pr_5 :=< Same\_hs.name, o_{I1}, o_{Ij}, h_d >$ where $o_{I1} = h_d$ and $o_{I_j} \in \{o_I \mid o_I$ has relation with $h_s\}$.	-add a new node on h_s containing h_d -if h_s have relations with other entities h_j :we link each h_j with each h_d instance if they are in relation with the same h_s.

Table 2. Pattern used to extract relations

We have applied our approach to extract an underlying social network from a relational database. In this section, we describe this process using the Hypernode database depicted in Fig. 3 and Fig. 4.

Person identification The chosen objects in this case are the hypernode representing persons. Then, in this step, we describe the process to identify people. The hypernode database schema is used to extract candidate hypernodes (those which may represent persons). Then, the instances are used to deeply analyze the candidates and detect those containing people.

Hypernodes candidates detection. A person has a number of characteristics like name, surname, birthday, address, email, etc. Some of these characteristics are used when designing databases containing persons. Based on characteristics from various ontologies such FOAF ontology and person ontology (schemaWeb), we have manually designed a person ontology (PO) containing all these characteristics and their synonyms (collected from WordNet). Figure 5 shows an excerpt of this ontology. Using the person ontology, the set of nodes related to each hypernode in the hypernode database is analyzed.

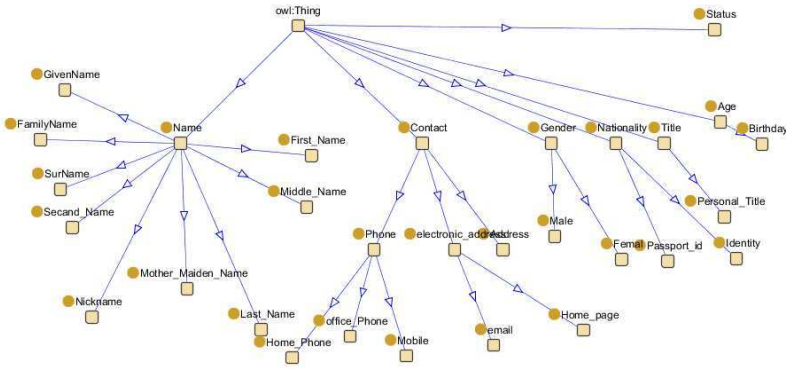


Fig. 5. Part of the Person Ontology

- If the node's name is one of the PO concepts, the number of characteristics for this hypernode is incremented.
- If the number of characteristics for the hypernode >= 1 and one of them contains a name, the hypernode h is a candidate to contain persons.

Candidate hypernodes Analysis. Each candidate hypernode has a set of instance hypernodes h_i. In order to analyze the name found in each instance (we take just the 10 first instances), the name is sent to a web search engine. The top 10 returned documents are downloaded and form the set D_i of the considered instance, and this set is parsed using DOM[4] . Each document in D_i is analyzed

[4] `http://www.w3.org/DOM/`

using the NER tool (Named Entity Recognition) developed in Stanford [5] and which is able to put three kinds of tags (Person, Location or Organization). We give for each document d, a weight w_d. If the name is tagged in the document by Person, the document is weighted by $w_d = 1$, otherwise $w_d = 0$. The mean assigned to the name found in the hypernode instance h_i ($\overline{h_i}$) counts how many times it is considered as a person name in the documents (where the tag of this name is Person):

$$\overline{h_i} = \frac{\sum_{d \in D_i} w_d}{|\mathrm{D_i}|} \tag{1}$$

The mean assigned to the hypernode ($\overline{H}$) calculates the mean where the names found in its instances are considered as a person name:

$$\overline{H} = \frac{\sum_{h_i} \overline{h_i}}{|\{h_\mathrm{i}\}|} \tag{2}$$

NER has a precision around 90% for finding Person entities; so, a hypernode is considered as representative of a person if more than 60% of its instances contain a person name (we take only 60% as a threshold due to problems such as wrongly written names, use of abbreviations, etc. which lower the precision of NER).

Relation construction From the previous step, the entities identification process identifies the Hypernodes "Student "and "Director_thesis" as person entities. Then, all their instance hypernodes are added to the entities set.

The relation construction process is then performed using the identified entities. We start by identifying the relation R_1 in order to search hidden entities.

In our example, the process identifies "*Foreign-Student"* as an entity due the relation $R_1 =<'' IS - A'', Foreign - Student, Student >$.

We will detail the identified relation in what follows using the set of pattern described in Table. 2.

From the relation $R_{h1} :=<$ "", $Director\_thesis, Student >$, we identify two patterns (Table 3):

- $Pr_1 :<$ "", $Director\_thesis, Student, null >$: in the database schema, *Student* and *Director_thesis* share the relation R_{h1}. Using Pr_1 and the value of the foreign key $St - id$, we search on the instance hypernode database for each *Student* the corresponding *Director_thesis*. R_{h1} relates directly *Student* and *Director_thesis* then we have no mediator (null).
- $Pr_2 :=< Same\_Student, Director\_thesis_i, Director\_thesis_j, Student >$: two thesis director may have the same *Student* (same value of $St-id$). Then, we search on the instance hypernode database all the instances of *Director_thesis* which have the same *Student* (mediator for this pattern) in order to add between them the relation Same_ Student.

[5] `http://nlp.stanford.edu/ner/index.shtml`

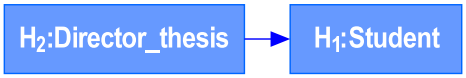
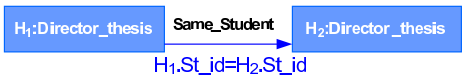
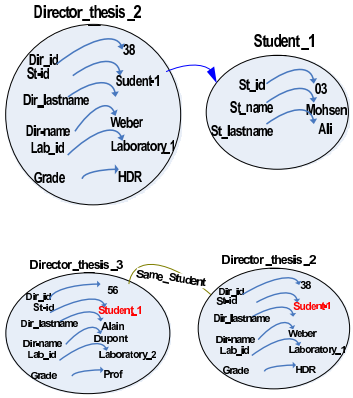
Relation and identified pattern	Example of extracted relations
Relation: $R_{h1} :=<'''', Director\_thesis, Student, >$ **Patterns:** $Pr_1 :=<'''', Director\_thesis, Student, null >$  $Pr_2 :=< Same\_Student, Director\_thesis_i, Director\_thesis_j, Student >$  	

Table 3. Relation R_{h1} Pattern

From the relation $R_{h2} :=<'''', Director\_thesis, Laboratory >$, we identify one pattern (Table 4): *Laboratory* is not an entity (of interest) then its instances are not included in the final graph:

- $Pr_3 :=< Same\_Laboratory, Director\_thesis_i, Director\_thesis_j, Laboratory >$: using the value of the foreign key *Lab_id* in each hypernode instance of the entity *Director_thesis*, we will link those having the same value of *Lab_id* by the relation *Same_Laboratory*.

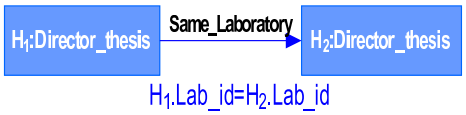
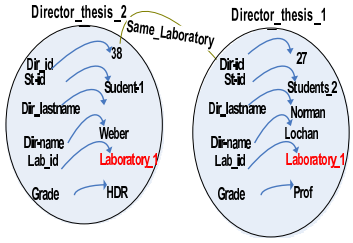
Relation and identified pattern	Example of extracted relations
Relation: $R_{h2} :=<'''', Director\_thesis, Laboratory >$ **Pattern:** $Pr_3 :=< Same\_Laboratory, Director\_thesis_i,$  *Laboratory* is not connected to other entities	

Table 4. Relation R_{h2} Pattern

From the relation $R_{h3} :=<'' Part-of'', Student, thesis\_hasStudent >$, we identify one pattern (Table 5) and we add some information:

– $Thesis\_hasStudent$ shares two relations "Part-of" with $Student$ and $Thesis$. We add a new node on the hypernode $Student < Thesis,\ Thesis\_i >$, corresponding to his Thesis. Then, we can apply the pattern Pr_4.
– $Pr_4 :=< Same\_Thesis, Student_i, Student_j,\ Thesis\_hasStudent >$, by this pattern we search all the students which share the same thesis. We did not find such relation which is semantically inexact.

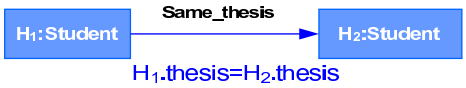
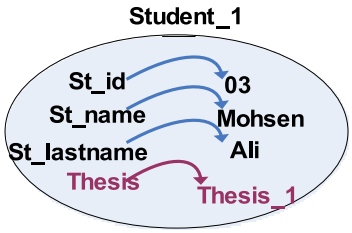
Relation and identified Pattern	Example of extracted relation
Relation: $R_h :=<'' Part-of'', Student,$ $thesis\_hasStudent >$ **Pattern:** - $Thesis\_hasStudent$ shares two relations "Part-of" with $Student$ an $Thesis$ - Add the node $n :< Thesis, Thesis\_i >$ to each instance of $Student$ - $Pr_4 :=< Same\_Thesis, Student_i, Student_j,$ 	

Table 5. Relation $Rh3$ Pattern.

From the relation $R_{h4} :=<'''', Thesis, Director\_thesis >$, there are no identified patterns because $Thesis$ is not related to other entities (Table 6). Considering

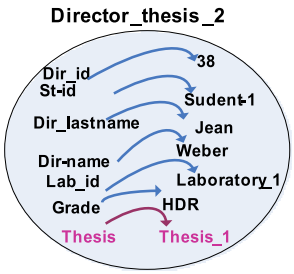
Relation and identified Pattern	Example of extracted relation
Relation: $R_{h4} :=<'''', Thesis, Director\_thesis >$ **Pattern:** - Thesis has no relations with other entities then no pattern detected. - Add the node $n :< Thesis, Thesis\_i >$ to each instance of Student	

Table 6. Relation R_{h4} Pattern

the identified patterns and the hypernode database instance (Fig.4), a first social network is extracted by applying the set of patterns to the instance hypernodes.

In order to obtain a more sophisticated social network, we merge the entities which share a relation "IS-A". For example, we merge the entity "Foreign-student" with the entity "Student" by adding all the information found in this hypernode to the hypernode "Student". Additionally, if "Foreign-student" has relations with other entities, these relations will be added to "Student". Finally, we obtain the social network depicted in Fig. 6. In the previous steps, we have

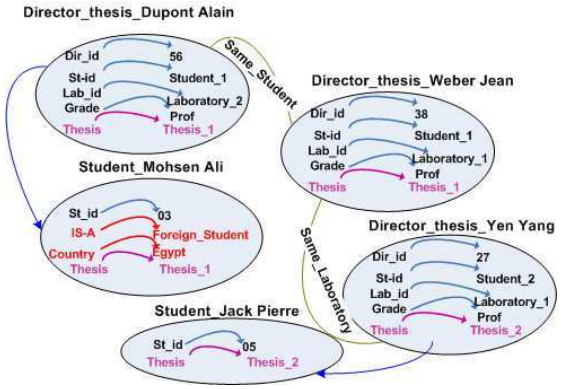


Fig. 6. The corresponding Social Network

extracted a social network from a relational database. However, a relational database can contain hundreds or thousands of tuple then we will obtain a very large social network.

From the resultant social network in Figure Fig. 6, a graph composed with homogeneous hypernodes has been extracted. The type of the selected hypernode is specified by the user (for example Director-Thesis) in order to perform K-SNAP algorithm. Then we will have an aggregated view of this graph which will facilitate its visualization and analysis.

The extracted graph has the hypernode instances of "Director_thesis" as vertex and their relations as edges.

4 Visualizing the social network using the k-SNAP graph aggregation

In the previous section, we were able to extract a social network from a graph database. However, as the actual relational databases contain hundreds or even thousands of records, we obtain as a result, the large social network seen in figure 7. An efficient graph aggregation will be valuable to visualize such graph in highlighting its underlying structure. And even if any classical community detection could be used to aggregate such a graph, do not use the inner values of the extracted nodes would be a wasting of efforts and information. Then we

chose to aggregate our graphs with the k-SNAP algorithm because it permits an aggregation on the both kind of information : links and attributes.

Let use recall here the basic concepts of graph theory. A *graph* G is defined as a pair of sets: $G = (V, E)$, where V is the set of *vertices* or *nodes*, and E the set of *edges* or *links* which connect vertices. The two vertices connected by an edge are called *endpoints* of this latter. Conversely, the edges connecting a vertice to the other vertices are called *incident* edges of the node. A graph can be *directed* or *undirected*. In the first case, also called *digraph*, an edge is denoted as pair (v, w), where v is the origin and w the target, and in the social networks framework it means: "v is in relation with w", the opposite being true if and only if there exists an edge (w, v). If the graph G is undirected, then an edge is denoted as the set of its vertices: $\{v, w\}$. Most of the times, vertices are labelled, but it can also be the case for edges, then G is called a *labelled graph*. Moreover, if exists a function $\omega : E \to \mathrm{R}$ that assigns a weight for each edge, then G is a *weighted* graph. Two vertices v and u are called *neighbors* or *adjacent*, if they are connected by an edge. The set of neighbors of a node v, denoted $\Gamma(v)$, is called its *neighborhood*. The topology of the graph can be captured in the *adjacency matrix* A, where the element $a_{i,j}$ is equal to one when $(v_i, v_j) \in E$, and zero otherwise.

From here, when we denote a graph by $G = (V, E)$ with V the set of nodes, the set of edges will be $E = \{E_1, E_2, ..., E_r\}$ the set of edge types, with each $E_i \subseteq V \times V$ representing the set of edges of a particular type.

And each node of V will be characterized by a set of attributes $\Lambda(G)$. For a set of attributes $A \subseteq \Lambda(G)$, a function Φ defined on V which is called *Attributes Compatible Grouping* or simply *A-compatible*, if it satisfies the following condition: $\forall u, v \in V, if \Phi(u) = \Phi(v)$ then $\forall a_i \in A, a_i(u) = a_i(v)$, it will be simply denoted by Φ_A. This function induce a partition $\{g_1, g_2, ...g_k\}$ on V where in each group g_i, every node has exactly the same values for the set of attributes A.

In fact, The *A-compatible* Φ_A only considers the node attributes. However, nodes do not just have attributes, but also participate in pairwise relationships represented by the edges.

For that, we consider now relationships when grouping nodes. For a grouping, we denote the neighbor-groups of node v in E_i as $NeighborGroups_{\Phi,E_i}(v) = \{\Phi(u)|(u, v) \in E_i\}$ which represents the set of groups on the partition associated with Φ where at least one element is connected to v by the relation E_i.

4.1 Attributes and Relationships Compatible Grouping

For a set of attributes $A \subseteq \Lambda(G)$ and a set of relations R, a grouping is compatible with the set of attributes A and relationship types R or simply (A, R)-*compatible*, if Φ satisfies the following conditions:

1. Φ is *A-compatible*,
2. $\forall u, v \in V$ if $\Phi(u) = \Phi(v)$,

then $\forall E_i \in R$, $NeighborGroups_{\Phi,E_i}(u) \equiv NeighborGroups_{\Phi,E_i}(v)$.

In each group of an (A,R)-*compatible* grouping, all the nodes are homogeneous in terms of both the set of attributes A and the set of relations R. In other words, every node inside a group has exactly the same attributes of A, and is adjacent to nodes in the same set of groups for all the relations in R.

Now, we can formally define the graph aggregation algorithm k-SNAP. Note that all calculations will be made from the incidence matrix $A_t = (\overset{t}{a}_{i,j})_{1<i,j<n}$ associated with the relation E_i.

4.2 k-SNAP Algorithm

First of all, we must mention that k-SNAP has been introduced to improve SNAP by relaxing the homogeneity requirement for the relations, *i.e*, for all relationships between two groups for example, there is no requirement that all nodes in these two groups are involved, however we want to maximize the ratio of participation while maintaining the homogeneity requirement for the attributes (the grouping remains *A-compatible*).

For this, we propose the evaluation measure Δ that allows to determine for each iteration the best group to be split until the size of the grouping is equal to k.

But first, we define N_{E_t} the participation matrix of rank $|\Phi_A|(|\Phi_A|$ is the cardinal of the partition of V induced by $\Phi_A)$ corresponding to the relation E_t by:

$$(\overset{t}{n}_{i,j})_{1<i,j<|\Phi_A|} = \sum_{k=0}^{|g_i|}(1 - \prod_{l=0}^{|g_j|}(1 - a_{kl}^t)) \tag{3}$$

Then, we define P the matrix of rank $|\Phi_A|$ which contains the ratios of participation of different groups with respect to the relation E_t:

$$(\overset{t}{p}_{i,j})_{1<i,j<|\Phi_A|} = \frac{n_{ij}^t + n_{ji}^t}{|g_i| + |g_j|} \tag{4}$$

For a given graph G, a set of attributes A and a set of relations R, the evaluation measure Δ of a *A-compatible grouping* Φ_A is defined as follows:

$$\Delta(\Phi_A) = \sum_{1\leq i,j\leq|\Phi_A|}\sum_{E_t\in R}\delta_{ij}^t \text{ avec } \delta_{ij}^t \begin{cases} n_{ij}^t & if\ \ p_{ij}^t \leq 0.5 \\ |g_i| - n_{ij}^t & otherwise \end{cases} \tag{5}$$

This measure is based on determining the difference in participation of each pair of groups with respect to the relationship E_t ,*i.e*, Δ-measure counts the minimum number of differences in participations of group relationships between the given *A-compatible grouping* and a hypothetical (A,R)- *compatible grouping* of the same size. According to equation (5) we have two possible cases:

1. If this group, the relationship is weak ($p_{i,k}^t \leq 0.5$), then it counts the participation differences between this weak relationship and a non-relationship ($p_{i,k}^t = 0$).

2. On the other hand, if the group relationship is strong ($p^t_{i,k} > 0.5$) , it counts the differences between this strong relationship and a 100% participation-ratio group relationship ($p^t_{i,k} = 1$).

Finally we define a matrix $W_{E_t} = (\delta^t_{ij})_{1 \leq i,j \leq |\Phi_A|}$ from equation (5), that evaluates the part of the Δ value contributed by a group g_i with one of its neighbors g_j in a group relationship of type E_t.

Given k the desired number of groups, the k-SNAP operation produces an (A, R)-*compatible grouping* with the minimum Δ value, starting from a A-*compatible grouping* and Δ initialized to zero, the procedure is to look for each iteration the group to split. For this, we introduce a heuristic that chooses the group that makes the most contribution to Δ with one of its neighbour groups. More formally, for each group g_i, we denote $CT(g_i)$ as follows: $CT(g_i) = max\left\{\delta^t_{ij}\right\}$.

Then, at each iterative step, we always choose the group with the maximum CT value to split, based on whether nodes in this group g_i which have relationships with nodes in its neighbour group g_t, where: $g_t = \arg\max_{g_j}\{\delta^t_{ij}\}$ and then split it into two sub-groups according to the following strategy: one of these groups contains all nodes participating in the relationship with the group g_t and the other contains the rest, i.e. the nodes that have no relation with the group g_t.

Now we will apply the algorithm on the graph extracted from the social network of thesis director (Fig. 7). In this experiment, we are interested in analysing

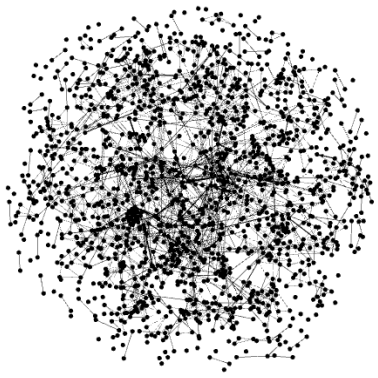


Fig. 7. The complete graph

how thesis directors in the database interact with each other based on two relations *Same_Laboratory* and *Same_student*. Each node in this graph has one attribute called *grade*, a direct visualization highlights our inability to interpret this graph without further treatment.

In order to explain the process of classification of k-SNAP, we will analyze the result of this classification on a sample of the real graph.

At first, k-SNAP generates a summary formed by three groups (*A-Compatible Grouping*) in accordance with the modalities *HDR* , *co-supervisor* and *prof* of the attribute *grade* as shown in Fig 8.a. The first iteration (Fig. 8.b) leads to the

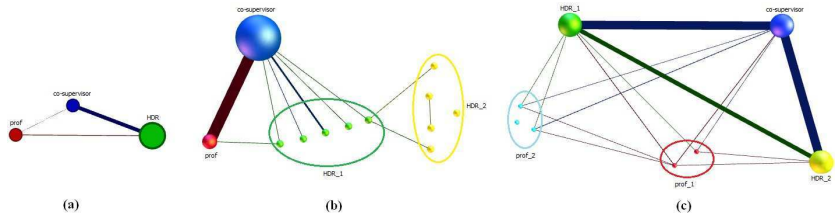


Fig. 8. The *A-Compatible Grouping and the 2 first iterations*

subdivision of the HDR group into two subgroups according to the relationship *Same_Student*, because it maximizes the contribution of Δ. This iteration gives rise to two groups: HDR_1 group consists of the HDRs that supervise a student with at least one professor or "co-supervisor".

However, HDR_2 group consists of the HDRs who supervise students that have only as director, HDRs. After the second iteration (Fig.8.c), the group of

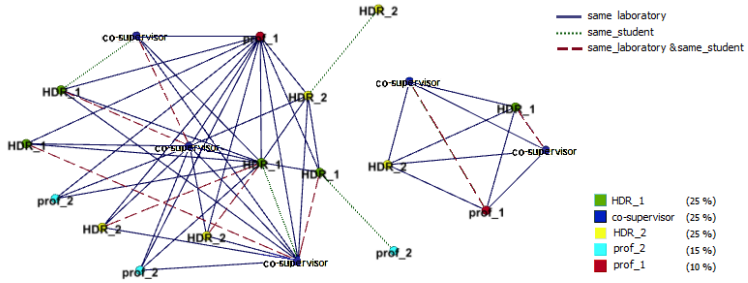


Fig. 9. Overview of the graph after the second iteration

professors is divided into two subgroups according to the relationship *Same_Laboratory* (Fig 9); the professor of $prof\_1$ group shares the lab with at least one of the other groups (HDR and co-supervisor), i.e, the laboratory to which they belong has members with various degrees. Even if our aggregation algorithm is rather of a semantic nature (takes into account the contents of the node), it's interesting to apply some of SNA metrics and try to compare its results with our criterion. We have chosen degree centrality as a measure and it distribution within the network is shown in the figure below (Fig 10). According to the subdivision of the HDR group (Fig 8.b), we can notice that nodes belonging to the HDR_1 group have globally a high degree of centrality. This is in accordance with our

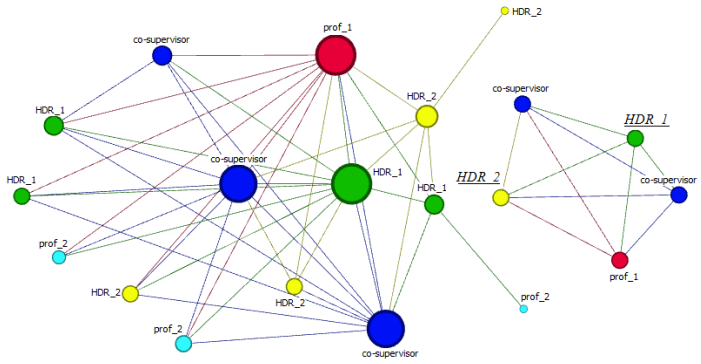


Fig. 10. Degree centrality of the graph

principle of division. In fact, all nodes of HDR_1 group supervises a student with at least one professor or "co-supervisor" thus, this creates a link which consequently increases the degree of centrality. It's the same interpretation for professors in the second iteration (Fig 8.c).

We can say that our results are consistent with the centrality measure.

However, if two nodes (highlighted and written in italic (Fig 10)) have the same median index of centrality, they may do not belong to the same group because their relationships with others are of different types. In this case, the measure of centrality is not adequate to our criterion; it depends on the nature of the relationship.

By changing the resolutions of summaries, users can better understand the characteristics of the original graph data and also explore the differences and similarities across different iterative steps.

5 Conclusion

In this paper, we have presented an approach to extract a social network from a relational database, then the aggregation method of the resulted social network using K-SNAP algorithm.

The social network extraction process is an application of the graph extraction approach from a relational database. This process allows having different graphs using as input the same relational database and the type of entities chosen by the user. The extraction approach is based on two steps: (1) translation of a relational database into a graph database, and (2) graph transformation which is realized after a process of objects identification then a graph rearrangement (nodes and relations). We have applied our approach using a real database.

The main interest to use K-SNAP was to show that algorithms designed for being applied on other kind of data sources can be use without any adaptation after applying our method for extracting social networks.

In our future work, we will focus on how to improve the extracting method by the use of an automatically built ontology describing the relations between

the entities on the relational database. Then, we will try to define a storage system based on the hypernode model and a graph query language better suited to the social network structure. We will also improve the aggregation method by combining it with conceptual clustering.

Acknowledgments

This work is partially financed by the ARSA project (Social Networks Analysis for Public Administrations) and by the STIC INRIA-Tunisia project "Social network exploration for recommender systems". The Academic chair in Business Intelligence is funded by SAP.

References

[1] Culotta, R., Bekkerman, McCallum, A.: Extracting social networks and contact information from email and the web. In: CEAS-1, (2004)
[2] Pentland, A.: Socially Aware computation and communication. In: Computer, Vol. **38** (2005) 33–40
[3] Mika, P.: Flink: Semantic web technology for the extraction and analysis of social networks. Journal of Web Semantics, **3** (2005) 211–223
[4] Girvan, M., Newman, M. E. J.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA, **99** (2002)
[5] Schaeffer, S. A.: Graph clustering. Computer Science Review, **1** (2007) 27–64
[6] Santo, F.: Community detection in graphs. Physics Reports, **486** (2010) 75–174
[7] MacQueen, J.: Some methods for classification and analysis of multivariate observations. Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif. 1965/66 bf 1 (1967) 281–297
[8] Freeman, L.C.: A set of measures of centrality based upon betweenness. Sociometry, **40** (1977) 35–41
[9] Newman, M. E. J.: Detecting community structure in networks. The European Physical Journal B, **38** (2004)321–330
[10] von Luxburg, U.: A tutorial on spectral clustering. Technical Report Technical Report, Max Planck Institute for Biological Cybernetics **149** (2006)
[11] Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, **22** (2000)888–905
[12] Ng, A. , Jordan, M., Weiss, Y., Dietterich , T., Becker, S., Ghahramani Z.: Advances in Neural Information Processing Systems, **14**, chapter On spectral clustering: analysis and an algorithm. MIT Press, (2002)
[13] Angles, R.,Gutierrez, C. : Survey of graph database models. ACM Comput. Surv. **40** (2008) 1-39
[14] Gyssens, M., Paredaens, J., Gucht, D. V.: A graph-oriented object model for database end-user interfaces. SIGMOD Rec. **19** (1990) 24–33
[15] Andries, M., Gemis, M., Paredaens J., Thyssens, I., Bussche, J. D.: Concepts for Graph-Oriented Object Manipulation. In Proceedings of the 3rd international Conference on Extending Database Technology: Advances in Database Technology, **580** Springer-Verlag, London, (1992) 21–38.
[16] Kuper, G. M., Vardi, M .Y.: The Logical Data Model. ACM Trans. Database Syst, (1993) 379–413

[17] Gemis, M., Paredaens, J.: An object-oriented pattern matching language. In: JSSST, Springer-Verlag, **742** (1993) 339–355.
[18] Hidders, J.: Typing Graph-Manipulation Operations. In: Proceedings of the 9th International Conference on Database Theory (ICDT). Springer- Verlag. (2002) 394–409.
[19] Levene, M., Loizou, G. : A Graph-Based Data Model and its Ramifications. IEEE Trans. on Knowl. and Data Eng. **7** (1995) 809-823
[20] Graves, M., Bergeman, E. R., Lawrence, C. B.: Graph database systems for genomics. IEEE Eng.Medicine Biol. **14** (1995) 737–745.
[21] Levene, M., Poulovassilis, A.: An object-oriented data model formalized through hyper-graphs. Data Knowl. Eng. **6** (1991) 205–224.
[22] Barnes, J. A.: Class and committees in a Norwegian island parish. Hum. Relat (1954) 39–58.
[23] Xu, X., Zhan, J., Zhu, H.: Using Social Networks to Organize Researcher Community. In: Proceedings of the IEEE ISI 2008 Paisi, Paccf, and SOCO international Workshops on intelligence and Security informatics. Springer-Verlag, Heidelberg, (2008) 421–427
[24] Maatuk, A., Akhtar, M., Rossiter, B.N.: Relational Database Migration: A Perspective. In: DEXA'08, (2008) 676–683
[25] Soussi, R., Aufaure, M.A., Baazaoui, H.: Towards Social Network Extraction Using a Graph Database. In: Proceedings of Second International Conference on Advances in Databases, Knowledge, and Data Applications, (2010) 28–34
[26] Newman, M. E. J.: The structure and function of complex networks. SIAM REVIEW , **45** (2003) 167–256.
[27] Chakrabarti, D., Faloutsos, Zhan, C. Y.: Visualization of large networks with min-cutplots, a-plots and r-mat. Int. J. Hum.-Comput. Stud. **655** (2007) 434–445.
[28] Rodrigues, Jr., J. F., A. J. M. Traina, C. Faloutsos, Traina, C. Jr.: Supergraph visualization. In ISM '06 : Proceedings of the Eighth IEEE International Symposium on Multimedia, Washington, DC, USA, IEEE Computer Society. (2006) 227–234
[29] Tian, Y., R. A. Hankins, J. M. Patel: Efficient aggregation for graph summarization. In SIGMOD '08 : Proceedings of the 2008 ACM SIGMOD international conference on Management of data, New York, NY, USA, (2008) 567–580
[30] Watts, D. J. , Strogatz , S. H.: Collective dynamics of 'small-world' networks. Nature **393(6684)** (1998) 440–442.
[31] Newman M.E.J, Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E, **69** (2004).
[32] Yan X., Han J.: gSpan: Graph-based substructure pattern mining. In Proceedings of ICDM'02, (2002) pages 721-724.
[33] Washio T., Motoda, H.: State of the art of graph-based data mining. SIGKDD Explor. Newsl. **5** (2003) 59–68.