



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

A MODEL FOR SOCIAL NETWORKS DATA MANAGEMENT

TESIS PARA OPTAR AL GRADO DE
DOCTOR EN CIENCIAS MENCIÓN COMPUTACIÓN

MAURO ADOLFO SAN MARTÍN RAMAS

PROFESOR GUÍA:
CLAUDIO GUTIÉRREZ GALLARDO

MIEMBROS DE LA COMISIÓN:
PABLO BARCELÓ BAEZA
BÁRBARA POBLETE LABRA
MARCELO ARENAS SAAVEDRA
LAKS LAKSHMANAN

Este trabajo ha sido parcialmente financiado por el Proyecto MECESUP UCH 0109.

SANTIAGO DE CHILE
AGOSTO 2012

Resumen

En el contexto de la administración de datos para redes sociales, esta tesis aborda sus necesidades de manipulación de datos proponiendo un modelo de datos basado en un conjunto exhaustivo de casos de uso tomados del dominio de las redes sociales (SN, del inglés “social networks”), y en el trabajo teórico existente sobre modelos de datos, bases de datos, y lenguajes de consulta.

Un modelo para la administración de datos de redes sociales debe permitir compartir los datos de redes sociales, así como su reutilización e integración, con apoyo para esquemas flexibles y metadatos apropiados para datos con estructura de grafos. El lenguaje de consulta deseado debe proveer la expresividad adecuada bajo límites factibles de complejidad, siendo además accesible y atractivo para los usuarios. Un requisito encontrado frecuentemente en los casos de uso de SN es la necesidad de reestructurar una red, por ejemplo creando nuevos nodos a partir de grupos existentes, o a partir de valores de atributos. Los lenguajes de consulta tradicionales que son capaces de crear valores u objetos suelen tener la capacidad de expresar todas las consultas computables, por lo tanto la evaluación de las consultas se vuelve computacionalmente costosa.

Para abordar estos requisitos se introduce un modelo de datos (SNDM), y un lenguaje de consulta (SNQL). La estructura de de datos utilizada es semiestructurada y está basada en un modelo de triples. SNQL se ha diseñado siguiendo las líneas de lenguajes de consulta ampliamente conocidos, usando como punto de partida una versión de *Datalog* con una extensión que facilita el cómputo de nuevos valores e identificadores de acuerdo a los requisitos de la manipulación de SN. Dicha extensión se basa en las *second-order tuple-generating dependencies*, originalmente propuestas en el contexto de intercambio de datos para capturar la composición de asignaciones entre esquemas.

El lenguaje así definido resuelve, con una complejidad computacional eficiente, los requisitos de los casos de uso típicos del análisis de redes sociales. En efecto, su poder expresivo abarca todas las operaciones de SN relevantes, y su evaluación permanece en NLOGSPACE. Se muestra que las características de este lenguaje satisfacen estas metas demostrando sus propiedades formales y con implementaciones prototípicas del modelo, así como con traducciones desde y hacia a otros modelos.

Abstract

This thesis addresses the data manipulation requirements for the management of social networks proposing a data model, which is based on a comprehensive set of use cases from the social networks (SN) domain and on existing theoretical background on data models, databases, and query languages.

A model for social networks data management must promote the sharing, reuse and integration of social network data, supporting flexible schemas and metadata for a graph data domain. The desired query language must provide the adequate expressiveness under feasible complexity bounds, while being accessible and appealing to the users. One frequent requirement found in SN use cases is to be able to restructure a network, e.g. creating new nodes corresponding to groups or clusters of existing nodes or to values of attributes. Traditional query languages that are capable of object or value creation tend to have the power to express all computable queries, thereby making query evaluation computationally expensive.

To address these requirements, we introduce a data model, SNDM, and a query language, SNQL. The data structure is semi-structured and based on the triple model. We designed SNQL on the lines of well known query languages, using a version of *Datalog* as the departure point, with an extension that facilitates the computation of new values and identifiers according to requirements of SN manipulation. This extension is based on second-order tuple-generating dependencies, originally proposed to capture composition of schema mappings in data-exchange settings.

The language so defined meets the requirements of typical social networks analysis use cases and grants computational complexity efficiency. In fact, we show that its expressive power is sufficient to cover all relevant SN operations, and its evaluation remains in NLOGSPACE. The features of this language are shown to satisfy these goals, by proving its formal properties and showing proof-of-concept implementations of the model, along with translations from and to other models.

A Raquel.

Agradecimientos / Thanks

Tengo mucho que agradecer...

A mi familia y amigos por el apoyo y la paciencia.

Y a todos y cada uno de quienes me ayudaron, de muchas maneras, a alcanzar los hombros de los gigantes.

I have a lot to thank...

To my family and friends for the support and the patience.

And to each and everyone who helped me, in many ways, to reach for the shoulders of the giants.

Contents

Introduction	1
1 Problem Definition and Contributions	4
1.1 Research Proposal: A Model for Social Networks Data Management	4
1.1.1 Hypotheses	5
1.1.2 Objectives	5
1.2 Contributions and Deliverables Produced	6
1.2.1 Contributions	6
1.2.2 Deliverables Produced	6
2 Technical Preliminaries	8
2.1 Social Networks (SN) and Social Network Analysis (SNA)	8
2.1.1 Social Networks	9
2.1.2 Social Networks Data Sets	11
2.1.3 Social Networks Analysis	15
2.2 Database Theoretical Framework	17
2.2.1 Datalog	17
2.2.2 GraphLog	18
2.2.3 Second Order tgds	20
3 Social Networks Data Management Requirements	24
3.1 Sources of Requirements	24
3.1.1 SNA Practice	25
3.1.2 Social Web and other Online Social Networks	27
3.2 Data Management for Social Networks	28
3.2.1 Social Network Data Lifecycle	28
3.2.2 Social Network Analysis as a Consumer/Producer	30
3.2.3 Desiderata for a Data Model for Management of SN Data	32
3.3 Design Issues and Principles	32
3.4 Proposed Solution	35
3.4.1 Data Structure Requirements	35
3.4.2 Query and Transformation Requirements	37
3.5 Related Work	39
3.5.1 Social Network Analysis	39
3.5.2 Data Models	40
3.5.3 Current Support for Social Networks Data Management	42

4 Social Networks Data Model I:	
Data Structure	43
4.1 Data Structure Requirements Summary	43
4.2 Data Structure	44
4.2.1 Classical SNA Model	44
4.2.2 SNDM Data Structure	46
4.2.3 Graphical Syntax	50
4.3 Triples Based Syntax	51
4.4 GraphLog Based Syntax	53
4.5 Related Work	55
5 Social Networks Data Model II:	
Query and Transformation Language	56
5.1 Query Language Requirements Summary	57
5.2 Language Overview	58
5.3 Query and Transformation Language: Syntax	59
5.3.1 Query Language Elements	60
5.3.2 SNQL: Use Cases and their Queries	61
5.3.3 SNQL Text Based Syntax	63
5.3.4 Graphical Syntax for Query Patterns	66
5.4 Query and Transformation Language: Semantics	68
5.4.1 Translation of Patterns to Datalog	68
5.4.2 Query Evaluation Process	71
5.4.3 Data Manipulation and Set Theoretical Operations	73
5.5 Expressiveness and Complexity	75
5.5.1 Expressiveness	75
5.5.2 Complexity	77
5.6 Related Work	78
6 Social Networks Data Model III:	
Implementations	82
6.1 General Implementation Remarks	82
6.1.1 Software Design Choices	82
6.1.2 Translations/Mappings to Existing DBMS	83
6.2 Software	86
6.2.1 SNAQuery: A First Prototype	86
6.2.2 SNDMS	87
6.2.3 SiRSL	91
6.3 Preliminary Performance Assessment	94
6.3.1 Experimental Setup	94
6.3.2 Results	95
6.3.3 Discussion	97
Conclusions and Future Work	98
Bibliography	103
A SNA Current Practices Study: Summary Tables	111

B	SocialScope Translation to SNQL	118
B.1	Data Structures	118
B.1.1	Comparison of Data Elements	118
B.1.2	Mapping between Data Structures	119
B.2	Translations to SNQL	120
B.2.1	Unary Operators	120
B.2.2	Basic Binary Operators	122
B.2.3	Advanced Binary Operators	125
B.2.4	Aggregation Operators	127

List of Tables

2.1	Example of two-mode matrix: companies (1-4) and directors (A-E). A cell containing a 1 denotes that the company (row) has the director (column) in its board.	11
2.2	Example of one-mode matrix: company by company. Two companies are related if they share at least one director.	12
2.3	Example of one-mode matrix: directors by directors. Two directors are related if they meet at least at one company board.	12
2.4	Evolution of Social Networks Data Sets	14
3.1	Pajek Use Cases	30
3.2	Selected social network data management use cases and their requirements.	38
4.1	Friendship network (Fig. 4.8) represented as sets of triples	53
5.1	Selected social network data management use cases, their requirements, and the corresponding query sketches over source network S . See use cases descriptions in Table 3.2.	62
5.2	Table <i>Temp</i> : collection of extraction pattern mappings.	73
A.1	List of articles included in our study sample and their activities for each SNA data workflow stage.	112
A.2	Summary of the study of the current SNA data workflow.	115

List of Figures

3.1	<i>Classical vs. New Online Social Networks.</i> The graphic shows a gap of orders of magnitude in network sizes (in actors) between classical and new online social networks. Classical networks are represented by data sets reported in articles published in the Social Networks journal –SN series in the graphic– whose sizes are all but one below the 10.000 boundary. (Data from our initial survey.)	25
3.2	Data Lifecycle Workflow for Social Networks. Each social application is a consumer/producer of social networks, producing and/or collecting network data, and consuming data produced by other applications. The need for a standard representation and query language follows.	29
4.1	<i>One-mode Friendship Network (Classic Model).</i> Directed labeled graph representing a network of friends. Arc labels represent how close each person rates the named friend. Note that attributes are stored in a separate structure. . .	44
4.2	<i>Two-mode Persons-that-appear-in-Pictures Network (Classic Model).</i> A two mode social network is a bipartite graph usually used to represent participation in groups or events. In this example all persons that appear in the same picture.	45
4.3	<i>Two one-mode networks from Persons-that-appear-in-Pictures Network in Figure 4.2 (Classic Model).</i> Transforming the paths of length two in edges, it is possible to obtain two one-mode social networks from one two-mode social network.	46
4.4	<i>Multi-modal Multi-relational Social Network.</i> In this variety of social network there exist actors of several families (persons and pictures) and different kinds of relations (appears-in, friend-of, and works-for). Different kinds of relations are represented by arrows with different line styles).	47
4.5	<i>Multi-modal Multi-relational Social Network with Attributes on Actors.</i> This is a mm-mr social network where each actor may additionally have an arbitrary number of attributes describing its features.	48
4.6	<i>Generalized Social Network.</i> A <i>generalized social network</i> is a tripartite labeled directed multigraph, where the node set N is partitioned into actors (round nodes), relations (square nodes), and attributes (small gray round nodes). Families are represented by gray labels attached to actors and relations. . . .	49

4.7	<i>Social Network Graphical Syntax.</i> Form left to right, the four graphical building blocks of a social network: an actor (above) and its family label, a relation and its family label, a participation role of an actor in a relation, and attributes on an actor and a relation. For each block the equivalent triple representation is also shown.	51
4.8	<i>Friendship Network.</i> A social network representing the friendship relation (square node) between <i>Mary</i> and <i>John</i> , who were introduced by <i>Ann</i> (actors as round nodes, and attribute values as grey dots). The cities of residence are also represented as actors.	51
4.9	<i>Edge semantics in GraphLog.</i> The meaning of an edge in <i>GraphLog</i> is defined by its translation to a <i>Datalog</i> predicate.	54
5.1	<i>Friendship Network and Simple Queries Results.</i> a) A social network representing the friendship relation (square node) between <i>Mary</i> and <i>John</i> , who were introduced by <i>Ann</i> (actors as round nodes, and attribute values as gray dots). b) The same social network after promoting <i>city</i> attributes to actors. c) The social network result after grouping persons by city and computing aggregate attributes: <i>inhabitants</i> of each city, and <i>number</i> of friendships between cities.	59
5.2	<i>Querying SNDM data.</i> The main components of a query are: a data extraction pattern P_E , and a construction pattern P_C . Each query proceeds as shown in the figure: (1) P_E is matched against the source network, (2) only the subnetworks that match are extracted, and grouped if needed for aggregation, (3) values bound to elements of P_E are used to populate elements of P_C and, possibly, to compute new values, and (4) finally the result network is constructed as the union of all instances of P_C	60
5.3	<i>Graphical representation of a simple SNQL query.</i> This is the graphical representation of P_E and P_C for the query that promotes city attributes to actors producing the network depicted in Figure 5.1b from the network depicted in Figure 5.1a. The correspondence between variables from P_E and P_C is indicated using the same names in both patterns. $A2$ and $R2$ represent new object identifiers created with functions: $A2 = g(L1)$ and $R2 = f(A1, A2)$	61
5.4	SNQL Syntax.	64
5.5	<i>FILTER operation graphical syntax.</i> The graphical syntax for FILTER operation simply adds the condition at the bottom of the pattern ($L1 > 18$).	66
5.6	<i>AND-NOT operation graphical syntax.</i> The graphical syntax for EP1 AND-NOT EP2 operation allows the combination of the two patterns in one diagram. Dashed lines identify EP2.	66
5.7	<i>TC operation graphical syntax.</i> The graphical syntax for TC($A1, A2, EP1$) WITH $L1 = \text{"John"}$ operation. Transitive patterns elements are drawn using double lines; start and end elements use bold lines. Conditions are included at the bottom of the pattern.	67
5.8	<i>AGG operation graphical syntax.</i> The graphical syntax for AGG($L1, \text{count}(*), EP1$) operation. The pattern is surrounded by a frame, including the grouping criteria and aggregated functions in its lower right corner. (This pattern is part of the query discussed in detail in Example 6).	67

5.9	<i>The evaluation process of a SNQL query.</i> The evaluation process occurs in two stages: (1) a <i>Datalog</i> program equivalent to P_E is evaluated over the data source network D producing intermediate tables of variable bindings; (2) portions of the result are produced from the obtained tuples using the <i>SO tgds</i> equivalent to P_C . All partial results are gathered in the final result network D' .	68
5.10	<i>Grouping and Aggregation.</i> Patterns EP1 and CP1 group people by ‘city’ and count the number of <i>inhabitants</i> in each city. Patterns EP2 and CP2 group and count friendship relations between pairs of cities.	71
5.11	Translation of query in Example 6 to <i>Datalog</i>	72
5.12	Evaluation Algorithm.	77
6.1	<i>A Modular Software Architecture for SNDM.</i> The figure shows the three layers that form the SNDM modular architecture, including service providers and libraries.	83
6.2	<i>SNDM Data Structure as a Relational Schema.</i> This schema implements the SNDM data structure (N , R , and M tables), and additional tables for queries and metadata.	84
6.3	SNQL Query Evaluation Steps with SQL. It is possible to evaluate most SNQL queries in two steps. First, an SQL query (equivalent to PE) extract datas from the source network and produces a temporary table $Temp$, then three different SQL queries build each triple set in the result (PC).	85
6.4	<i>SNAQuery Prototype Main Window.</i> The main window shows the complete network and the query to select the friends that have the same boss.	86
6.5	<i>SNAQuery Result Window.</i> This window shows the result network after selecting the friends that have the same boss.	87
6.6	<i>SNDMS Architecture.</i> The figure shows the packages and libraries that form the SNDMS.	88
6.7	Graphical representation of KHTM network imported to SNDMS.	89
6.8	Query editor with an SNQL query that selects only advice relation.	89
6.9	Result social network from query in Figure 6.8 over source social network in Figure 6.7.	90
6.10	Advice subnetwork in Pajek.	90
6.11	Detail of <i>manager18</i> in the KHTM network enriched with centrality values computed in Pajek	91
6.12	<i>SiRSL Snapshot.</i> Main result window of SiRSL showing the result of searching for the term <i>history</i> in publication <i>titles</i> . All matching titles are shown as a list in the text view (left), and as a graph in the graphical view (right). Double-clicking on a title in the graph adds to the view all neighbors: titles that have been borrowed by a user that also borrowed that title. (Some labels have been translated from Spanish)	92
6.13	<i>Social Network Schema for Library: Books View.</i> Extraction pattern (a) matches pairs of books that have been borrowed by the same user. This result is used by the construction pattern (b) to produce a network of related books. A more restrictive view requiring, besides at least one user in common, that two books have a subject in common, can be obtained with the extraction pattern EP1 AND EP2, the conjunction of the patterns (a) and (c).	93

6.14	<i>Experiment Sequence.</i> Data import, attribute promotion query (authors from attributes to actors), path contraction (coauthorship network).	95
6.15	<i>Q1 - Authors Promotion from Attributes to Actors.</i> Promoting authors to actors connect publications that shares the same author. Figure depicts the patterns in the SNQL query and its four equivalent SQL queries.	96
6.16	<i>Q2 - Building the coauthorship network.</i> This query works on the result of the previous one. It contracts paths of length 2 between authors, where there is a coauthored publication. The figure depicts the patterns in the SNQL query and its four equivalent SQL queries.	96
6.17	<i>Q1 and Q2: Query Evaluation Times.</i> Evaluation times in ms for Q1 (a) and Q2 (b) for each subset, per number of processed tuples (input + temp + output).	97

Introduction

Today there are huge amounts of digital data being constantly produced in disparate fields from science to everyday errands: survey astronomical data, genomic data, business transactions, text messages flowing seamlessly across different networks, just to name a few.

Several technological developments in the last decades have concurred to support this trend. One that has had wide media coverage, and a crucial role, is the Web. These phenomena have been technological but also social. For instance, the evolution from the original services provided over the Internet to the current Social Web¹ has been driven by the interest of the public, and their willingness to provide their data and to consume data from others' work and personal lives.

Evidently the information hidden in this plethora of data has the potential to advance our understanding of the universe and of ourselves, not to mention its considerable monetary value. However, the effective collection, storage, and manipulation of complex data at this scale are not trivial tasks. To some extent, the existing data-managing infrastructure can be leveraged to cope with the scale of the problem; nevertheless, there exist areas where new approaches are needed. Specially designed data-managing tools are required.

Some of these problematic areas require the management of complex structured data for which there are no well-developed solutions. One example is network-structured data: data sets where elements can be related in arbitrary patterns, and where the information about the relations and their patterns is more or equally important as the information about the elements. Mathematically, they are usually represented as directed labeled graphs. Actual network data sets also require to record attributes on elements and relations.

The computational processing on these networks has been done mainly in an ad-hoc manner or from a data mining perspective. This strategy does not solve the inherent and classical data management issues, like data querying and manipulation, archiving, automatic collection, and interoperation. In fact, the need for data management support has surfaced an explicit and urgent issue in some application domains, due to recent developments; for instance, on biosciences due to industrialization of biological data gathering, e.g. sequencing farms [63], and in social networks analysis, where there is a need for automatic and continuous data collection for extensive analysis [26]. This situation has opened a number of problems related to data management support of network data, currently not addressed [62].

¹The term Social Web was originally coined by H. Rheingold in 1996 [91]

There are plenty of networks around us. In the last decade and a half, scientists from several fields have embarked on the identification, collection, and study of networks: physical phenomena like percolation [40], technological networks like the Web [1], biological networks like yeast protein interaction network [61], and of course several types of social networks like scientific collaboration networks [82] and the new online social networks [74, 101].

Most of the renewed enthusiasm in networks of the late 1990's occurred in the context that was called *complex networks* [83], an umbrella term coined by physicists to label all the recently identified sparse networks that share a number of the following properties: small diameter, high clustering coefficient, a scale-free node degree distribution, and in some cases navigability.

Social networks, as data objects, represent a set of actors (individuals, groups, or objects), and their ties and interactions. Among the several domains that work nowadays with network-structured data, social networks show a substantial amount of activity probably due to the availability of social networks data sets, unprecedented in complexity and size, and the development of new tools to study them. These new data sets and analysis tools have installed the urgent need for a specialized network data management infrastructure to store and manipulate the data for research, sharing, and preservation in the short and long term.

Formally the study of the data-managing problem in a given domain is addressed through a data[base] model. This approach allows the study of the data structures, manipulation operations, and constraints required to provide the desired and feasible data managing solution.

Social networks –independent of their origin– have common characteristics [12, 51, 83] useful to provide a foundation for a common data model, as defined by Codd [27], i.e. as a set of data structures, a collection of transformation and query operators, and integrity constraints. While the dominant relational data model does not provide support for basic network operations (e.g. path finding and motif searching [62]), other data models, like graph data models [8] and semi-structured data models, may offer a better support. Additionally, a review of research in database and data mining conferences, shows that database support for social networks, backed by an appropriate data model, remains an open problem [63].

Mainly, two aspects have made social networks to be an interesting subject from a data management point of view. First, they have a well-defined methodological framework for their study, developed in the last century in the contexts of sociology, anthropology, and psychology. It provides a rich and well-defined set of standard network operations [51, 98, 110]; moreover, the data in this field is dynamic, increasing the need for database support [20]. Second, in the last years the automatic recording of social interactions has produced social network data sets with a level of detail and at a scale never seen before. Consider for instance that a popular social web service may produce social interactions data in the order of gigabytes per day.

This thesis addresses the problem of social networks data management, proposes a custom data model for social network data management, studies the properties of this model, and explores some of its possible implementations. Special attention is given to the key trade-off in the design of a query language: the balance between the complexity of query evaluation, which

determines the practical feasibility of the solution, and the expressiveness of the language, understood as the set of queries that can be expressed with it.

Defining a data management model that copes with the requirements of these networks poses several challenges both at the abstract and implementation levels. For instance, consider at the abstract level a network with relations whose arity is not fixed at modeling time, and operations that may require to transitively traverse the structure. At the implementation level, there are no clear locality boundaries or an easily identifiable primary sorting order to facilitate basic operations like search.

Document Organization

This document is organized as follows:

- Chapter 1, Problem Definition and Contributions. Definition of the research problem addressed, and summary of the results and deliverables produced.
- Chapter 2, Technical Preliminaries. Technical aspects of social networks and social networks analysis, and of databases languages and formalisms used to define and study the data model.
- Chapter 3, Social Networks Data Management Requirements. Requirements analysis from the context of social networks, particularly from the social network analysis practice.
- Chapter 4, Social Network Data Model I: Data Structure. Specification of the data structure to represent social networks data: actors and the relations among them, along with their attributes.
- Chapter 5, Social Networks Data Model II: Query and Transformation Language. Syntax and semantics of the proposed query and transformation language, including the analysis of its complexity and expressiveness.
- Chapter 6, Social Networks Data Model III: Implementations. Practical exploration of the model through implementations of proof-of-concept prototypes.
- Conclusions and Future Work. Closing discussion of the main results of this thesis and of ideas to further study and develop the model and its implementations.
- Appendix A, Survey of SNA Data Management Practices. Summary tables of the survey of publications in the Social Networks journal to profile the SNA community data management practices.
- Appendix B, SocialScope Translation to SNQL. Translation of SocialScope, a data management framework for social content sites, to the social networks data model query language (SNQL).

Chapter 1

Problem Definition and Contributions

In this chapter we summarize the proposed research and solution, and enumerate the contributions and deliverables produced: publications, software, and other work related to the main research.

1.1 Research Proposal: A Model for Social Networks Data Management

The problem that we will address here is the lack of a database support for social networks. In particular, we will focus at the logical level of databases and in the data managing requirements (querying, transforming, reusing, integrating, etc.) of social networks study and systems.

To the best of our knowledge, this problem has been poorly addressed, and current solutions are ad-hoc solutions without solid data-managing theoretical ground. A good example is Pajek [13, 35], one of the tools of choice of social networks analysis practitioners, which works only with implementation-oriented data structures, inherited from manual social networks analysis. Another example of a more database oriented tool, NetIntel [103], is hardwired to a specific relational database schema, which in turn severely limits the expressible networks.

Social networks analysis has a homogeneous underlying model: data structures, operations and constraints are similar across the domain. This is not surprising given its long tradition of formal study starting in the early 1930s [50].

Following Codd's assertions, a data model for the management of social networks data should provide:

1. Data structure types to represent networks, its components and relations.
2. Operations to perform network data manipulation (selection, insertion, deletion and update). These operations could require some analysis capabilities over the network,

for instance to find connected components and to select them.

3. Integrity constraints to keep the network consistent as a network and under domain-specific restrictions.

We propose the design and development of a data model for social networks data to provide a data managing infrastructure to social networks analysis.

1.1.1 Hypotheses

This work is based on the following hypotheses:

- Social networks are general enough to justify a data model. By “general enough” we mean that they are present as many instances in different contexts, representing an amount of data that needs automatic management methods.
- A specific data model, with a graph database point of view, will improve the support for the automation of data managing in the social networks field. As a result, it is expected that productivity will improve, as in the well-documented case of the relational data model and its application domain. (The argument is that a data model will let users and programmers access data through an abstraction layer similar to that used in the application domain, with all low-level data managing hidden by the database management system [28].)
- The common data structure characteristics, and the well defined set of operations over social networks, give design advantages over a generic graph database model, allowing, for instance, the use of algorithms which are not usable in a more general context.

1.1.2 Objectives

General Objective

To design and develop a database model for the management of social networks data.

Specific Objectives

- To show, by presenting this work in appropriate publications and conferences (see first deliverable below), to the social networks community that there are benefits in database techniques, and to the database community that the social networks field brings relevant challenges.
- To study social networks analysis data representation requirements and its characteristic data manipulation tasks.
- To define the appropriate collection of data structures as part of a social networks data model.

- To define the appropriate set of operations as part of a social networks data model, i.e. a query language to implement the required data manipulation tasks.
- To define the appropriate set of integrity constraints for a social networks data model.
- To evaluate different implementation techniques through proof-of-concept prototypes against the requirements of the data model.
- To demonstrate the feasibility of the data model by testing the performance of the proof-of-concept prototypes with actual social network data.

1.2 Contributions and Deliverables Produced

1.2.1 Contributions

As a result of our research, we defined, characterized, and implemented a data model that fulfills the data managing requirements of social network analysis practice. The main contributions of this work are:

- The identification and documentation of data managing requirements in the context of social network analysis practice.
- The definition of a data structure capable of representing generalized social networks.
- The definition of a query and transformation language to manage social networks, with the adequate expressiveness and a feasible data complexity.
- The implementation of proof-of-concept prototypes to show the feasibility of the data model.

1.2.2 Deliverables Produced

This section enumerates the publications, other works, and software produced while working on this thesis.

Publications

- M. San Martín, P. Wood, C. Gutierrez. Transforming and Querying Social Networks. AMW2011, V Alberto Mendelzon International Workshop on Foundations of Data Management. Santiago, Chile. 2011.
- M. San Martín, C. Gutierrez. Personal Management of Social Networks Data. Computational Science and Engineering (SocialCom, CSE '09), International Conference on. p. 765-770. IEEE CS 2009(04).
- M. San Martín, C. Gutierrez. Representing, Querying and Transforming Social Networks with RDF/SPARQL, en The Semantic Web: Research and Applications (L. Aroyo et al. eds.). European Semantic Web Conference (ESWC2009). Lecture Notes in Computer Science v. 5554, p. 293-307. Springer. ISBN 978-3-642-02120-6. 2009.

- M. San Martín. A Data Model for Social Network Analysis. AMW2007, II Alberto Mendelzon International Workshop on Foundations of Data Management. Punta del Este, Uruguay. 2007.
- M. San Martín, C. Gutierrez. A Database Perspective of Social Network Analysis. International Sunbelt Conference XXVI. Vancouver, Canada. 2006.

Other Related Works This thesis has produced the following graduation projects for the computer engineer title at the Universidad de La Serena, and at the Universidad de Chile.

- Sistema Mejorado de Búsqueda Bibliográfica para SIBULS (Improved Bibliographic Search System for SIBULS). Students: F. Ferreira y J. Palominos. Advisor: M. San Martín. Ingeniería en Computación - ULS, 2010.
- Aplicación para la Creación y Administración de Redes Sociales (An Application to Create and Manage Social Networks). Student: M. Bahamondez. Advisor: C. Gutierrez. Ingeniería Civil en Computación - UChile, 2009.
- Sistema de Administración de Datos de Redes Sociales (Social Networks Data Management System). Students: R. Barquín, O. Gálvez, y F. Zumarán. Advisor: M. San Martín. Ingeniería en Computación - ULS, 2009.

Proof-of-Concept Prototypes The following software systems are presented in detail in the Chapter 6, *Social Networks Data Model III: Implementations*.

- SiRSL: Sistema de Recomendación Social de Libros (Social Recommender System for Books). Library search system augmented with social information. The system allows text and graphical search and browsing of a library catalog. Links between publications are computed based on bibliographical information (v.g. author and topics), and users activity: two books are connected if the same user has borrowed them. Relations have weights and can be combined. The system is online at <http://sn.dcc.uchile.cl/sirsl>.
- SNDMS: Social Networks Data Management System. Database management system that implements the data model over existing DBMS infrastructure, for instance, an RDBMS. It is implemented in Java following a modular design with three main packages: a module to create, edit and visualize social networks (including query results); a query evaluation module that translates SNDM queries and results between the system and the DBMS in the backend; and a module to import and export from and to XML and Pajek (a popular social network analysis tool).
- KHTM Prototype. Small Java application that allows the visualization and evaluation of predefined queries over the well-known social network dataset Krackhardt's High Tech Managers, as reported by Wasserman [110].

Chapter 2

Technical Preliminaries

In the last years, the development of an infrastructure to automatically record and store scientific data in machine processable formats [53], and the development of the social web and other repositories of complex structured data, created the pressing need for adequate support for the management of network structured data [81].

From the computer science perspective, the management (representation and manipulation) of network-structured data is a relevant but hard problem and the definition of a general computational infrastructure to address it has been elusive. The research efforts in this direction have been developed intermittently, and no definitive answer has been reached [8].

In this chapter we present the technical preliminaries in the two main areas related to the proposed data model: social networks and social network analysis, and databases.

First, we define the basics concepts regarding social networks (e.g. actors, relations, one and two-mode networks) and social networks analysis. Further details, when needed, are discussed along with the requirements analysis in the next chapter.

In the second section we present the languages and formalisms from the databases field that provide the theoretical foundations for the proposed data model: *Datalog*, *GraphLog*, and second order tuple-generating dependencies.

2.1 Social Networks (SN) and Social Network Analysis (SNA)

Social networks (SN) are plentiful examples of network-structured data that occur in the real world. They have been under organized study by sociologists, psychologists, and anthropologists for about a century: the first modern research efforts can be traced to well-documented studies published in the first half of the last century, since the foundational works of Moreno on small groups [50, 110].

Social networks data management requirements have a well-developed and established methodological framework. This framework addresses the collection, including the problem of sampling/boundaries, representation, preparation, and analysis of social networks data. Social network analysis (SNA) is understood as the identification and the discovery of social and psychological significance of the structures found in a social network.

We focus our work on data collection, representation, and preparation, leaving analysis outside of the scope of the present work (under the same criteria that usually separate data management from data mining in other problem domains). In the presence of massive data sets, the means to identify and access meaningful subsets are crucial to the work of the users.

We present here the fundamental concepts from the social networks domain, and specifically from a social networks analysis perspective [26, 51, 110].

2.1.1 Social Networks

To build the formal definition of social networks, we will define some key concepts, following the classical definitions from this domain [110, p. 17]. Later, in the definition of the model, we will generalize some of these concepts to extend their scope.

Actors. An actor is a social entity, which is under study along with its social interactions. In the strict sense, actors can be defined as individual, corporate, or collective social units. Examples of actors are people in a group, departments within a corporation, public services agencies in a city, or nation-states in the world system. The use of the term actor does not imply that these entities necessarily have volition or the ability to act. Furthermore, most classical social networks applications focus on collections of actors that are all of the same type (for example, people in a workgroup). Sometimes, however, the research needs to look at actors of conceptually different types or levels, or from different sets. The data may include non-relational attributes associated to the different actors.

Relational Ties. Actors are linked to one another by social ties. The range and type of these ties can be quite extensive. The defining feature of a tie is that it establishes a linkage between a pair of actors. Some of the more common examples of ties employed in network analysis are:

- Evaluation of one person by another, e.g. declared friendship, liking, or respect.
- Transfer of material resources, e.g. business transactions, lending or borrowing.
- Association or affiliation, e.g. jointly attending a social event, or belonging to the same social club.

Relations. The collection of ties of a specific kind among members of a group is called a relation. For example, the set of friendships among pairs of children in a classroom, or the set of formal diplomatic ties maintained by pairs of nations in the world, are ties that define

relations. For any group of actors, we might measure several different relations; for example, in addition to formal diplomatic ties among nations, we might also indicate the existence of trade in a given year. Relations (or rather, specific ties) may have attributes describing them; for instance, in the case of trade, its total monetary value may be recorded.

Groups and Subgroups. A subgroup is a subset of actors and all ties among them; for instance, dyads (two actor sets) and triads (three actor sets). A cohesive subgroup is defined by the pattern of ties among its actors; for example, a clique. On the other hand, the notion of group has been given a wide range of definitions by social scientists. For our purposes, a group is the collection of all actors on which ties are to be measured. One must be able to argue by theoretical, empirical, or conceptual criteria that the actors in the group belong together in a more or less bounded set. Indeed, once one decides to gather data on a group, a more concrete meaning of the term is needed. A group, then, consists of a finite set of actors that for conceptual, theoretical, or empirical reasons are treated as a finite set of individuals on which network measurements are made.

The restriction to a finite set or sets of actors is an analytic requirement. Modeling finite groups presents some of the more problematic issues in network analysis, including the specification of network boundaries, sampling, and the definition of groups.

Observation unit vs. Analysis unit. Observation units are the components of the model used to record the facts from the real world. In social networks studies and analysis the observation unit defines about which objects we are going to record attributes and social ties. For instance, students in a school, social occasions in a community, an so on.

In contrast, the analysis units (modeling units in Wasserman and Faust [110]) are components of the model used to define the social network to be analyzed, for instance: actors, dyads, triads, subgroups, and subnetworks.

In a given context, the observation and analysis units may be the same, for instance students; that is, the information collected is about students, and the network analyzed is a network where the actors are students. This is usually the case when the data is collected for a specific experiment. However, in many cases both units may not be the same, for instance the data may be collected for each student, but the analysis may be performed regarding a network of classes (groups of students) of the school. This is an increasingly common situation, given that data collection is usually performed automatically and independently of the possible analysis that may be performed later.

Social Network. A social network consists of one or several finite sets of actors, together with the relation or relations defined among them. The presence of relational information is a critical and defining feature of a social network. A social network is a particular case of network, and hence its structure may be formalized as a graph.

In addition to the use of relational concepts, Wasserman and Faust [110, p. 4] note the following important considerations:

- Actors and their actions are viewed as interdependent, rather than independent, autonomous units.
- Relational ties (linkages) between actors are channels for transfer or “flow” of resources (either material or nonmaterial).
- Network models focusing on individuals view the network structural environment as providing opportunities for or constraints on individual action.
- Network models conceptualize structure (social, economic, political, and so forth) as lasting patterns of relations among actors.

2.1.2 Social Networks Data Sets

Social networks analysis is centered on the analysis of data about relations, which in this field is called *relational data*. The common social network analysis practice, following the idea of representing social networks as graphs, is to organize this data in adjacency matrices or sociograms. The classical SNA approach defines three types of matrices [98]:

- Case-by-variable matrix: This kind of matrix has cases (actors) in the rows, and some attributes of them in the columns. For instance, a case-by-variable matrix could contain information about gender, age and income of persons in the network under study. Usually, this is not considered *relational data*.
- Case-by-affiliation matrix: In this kind of matrix, there are also cases (actors) in the rows, but in the columns are labels of groups or events (affiliation). Each cell contains a number indicating the degree of participation of the corresponding case in the corresponding affiliation. Also known as two-mode matrix or incidence matrix. Table 2.1 is an example of a two mode matrix (taken from Scott [98, p. 45]) that has its rows labeled with companies (1-4) and its columns with members of their boards of directors (A-E):

	A	B	C	D	E
1	1	1	1	1	0
2	1	1	1	0	1
3	0	1	1	1	0
4	0	0	1	0	1

Table 2.1: Example of two-mode matrix: companies (1-4) and directors (A-E). A cell containing a 1 denotes that the company (row) has the director (column) in its board.

- Case-by-case matrix: This kind of matrix is a square matrix that has cases (actors) in both rows and columns. Cells also contain a number to measure the level of relation between corresponding cases. It is also possible to have an affiliation-by-affiliation matrix, with similar characteristics. Also known as one-mode matrix or adjacency matrix. Tables 2.2 and 2.3 show the two one-mode matrices that can be derived from the example above (see Table 2.1) where relations between cases (companies or directors) has been weighted by, in the first case, the number of shared directors and, in the second case, by the numbers of boards where two directors meet each other.

	1	2	3	4
1	-	3	3	1
2	3	-	2	2
3	3	2	-	1
4	1	2	1	-

Table 2.2: Example of one-mode matrix: company by company. Two companies are related if they share at least one director.

	A	B	C	D	E
A	-	2	2	1	1
B	2	-	3	2	1
C	2	3	-	2	2
D	1	2	2	-	0
E	1	1	2	0	-

Table 2.3: Example of one-mode matrix: directors by directors. Two directors are related if they meet at least at one company board.

There are three key related aspects regarding SN data sets: collection methods, network boundaries, and data sampling.

The usual workflow starts by specifying data to be collected in the field using surveys or direct observation. In this stage, the support media varies greatly from paper and pencil to some general-purpose computational tools like word processors and spreadsheets. Then the data is validated and depurated. Again, at this stage, general-purpose data manipulating tools are used. Finally, relational data is used to construct a case-by-variable matrix and a case-by-affiliation matrix, and from them –if it is needed– to build different case-by-case matrices, or sociograms, to perform the desired analysis. Most SNA tools and software are designed following this practice.

Data collection by surveys or direct observation and its subsequent validation usually constitutes the most expensive stage on this kind of a research project; situation that naturally raises questions of network boundaries: Which is the network under study? Who are the actors involved? What relations must be recorded? With non-relational data, the usual solution is to sample the universe to the level in which statistical significance is reached. However, with social networks data usually there is not enough information before analysis to decide the relevance of a given actor. Furthermore, omitting a key actor may bias dramatically many properties of the network, like the diameter. In practice, researchers prefer well-known, defined, and rather small networks to completely avoid considering sampling. This characteristic calls for an appropriate data-managing infrastructure that permits to incrementally collect and integrate network data to progressively enrich data sets.

Another interesting aspect is networks dynamics; that is, how they change over time. The data sets that include information for more than one instant are called longitudinal data by the SN community [110]. As it is well known, the inclusion of temporal data is a complex task in databases [2, 73]. We do not address time related issues in the present work.

Through the extensive history of SN study, the data sets have undergone an evolution from small, static, and manually curated, to big dynamic, and automatically collected and stored. This evolution has been driven in part by the sophistication of analytical methods, but mostly by the development of automated systems that collect data continuously. Given the costs involved in classical data collection procedures, it was not until recently (15 years) that the studies addressing big networks became practical for a wide range of research initiatives. In spite of these developments, tools for network data management are not present yet.

Table 2.4 shows a comparison among social networks data sets using five examples, ranging from the “classical” and well known, profusely used in SNA, to the new online social networks and the social web.

The columns represent distinct groups of data sets in SNA via an example. *S. Women* corresponds to the Davis *southern women event participation data* as reported by Doreian [38], which represents a classical example of data directly collected by researchers and hand curated. These types of data sets are usually rather simple (limited in size, number of attributes and kind of relations collected), static, and collected with the sole purpose of SNA. *Intl. Trade* describes the *countries trade data* as reported by Wasserman [110]; this data set is similar to the previous one but more complex in that it records five relations (as five different data sets), and though it is used only for SNA purposes, it was extracted from a data set compiled originally with other purposes. *EIES* summarizes the properties of Freeman’s *Electronic Information Exchange Network* of SNA researchers, as reported by Wasserman [110]; this data set is an early example of a new trend: SNA data sets that are collected automatically during the operation of a functional system. It is also slightly more complex than the previous examples, and includes longitudinal data, but it is not truly dynamic yet (once collected and prepared, the network remains static). The last two cases, *DBLP*¹ and *Facebook*², are examples of online social networks and the social web which exhibit new levels of data management complexity: hundreds of thousands of actors, multiple kinds of actors and relations, and attributes on both; domain dependent uses, and online access. A key difference between them is that *DBLP* changes in centrally controlled discrete events, while *Facebook* data changes continually as the users interact with the system.

We use the term ‘online social networks’ to refer to the network data sets that are collected, curated, and typically available online, even if they do not represent relations among people; in most cases, the relations at least resemble or are produced by direct or indirect social interaction.

A particularly rich on-line source of social networks is the Social Web. The Social Web evolved from the Web when it became participative, and even more as it became the new media of choice for social interactions.

As it is shown, there is no clear boundary but instead some sort of evolution from classical to current social networks. Nevertheless, there are three central aspects that characterize the new social networks:

1. *Size and Complexity:* These new networks are attributed multimodal multinetworks

¹<http://dblp.uni-trier.de/>

²<http://www.facebook.com>

Table 2.4: *Comparative examples of social network data sets.* Examples of data sets containing social networks (in columns), from classical to new online social networks. Comparison parameters (in rows) illustrate the transition from: small and simple to big and complex, static to dynamic, and SNA dedicated to application dependent purpose. In “Size” n and m correspond to the number of nodes and edges, respectively.

Dataset Name	Classical			Online		
	Southern Women	International Trade	EIES	DBLP	Facebook	
Modes	2-mode	1-mode	1-mode	multi-mode	multi-mode	
Size (n, m)	((18,14),89)	(24,310)	(32,650)	(M, Ms)	(MM, MMs)	
Kinds of Relations	one	one (5 sets)	one (4 sets)	many	many	
Attributes on Actors	demographic	demographic	discipline and citations	many	many	
Attributes on Relations	no	no	yes (a number)	yes	yes	
Primary Purpose	SNA	SNA	SNA	bibliographic database	social interaction	
Dynamic	no	no	yes (two states)	yes	yes	
Collection	Observation and manual collection	Imported data	Manual and automatic collection	Automatic (assisted)	Automatic	
Update	No	No	No	Periodic updates	Online	
Date	1930s	Published in 1994. Built from existing data (1979 and later)	1979	Since 1980s	Since 2004	

–many kinds of actors and relations, possibly with some relations linking more than two actors– with hundreds of thousands or even millions of actors.

2. *Purpose*: Even though they contain relational data, their primary purpose is not SNA. Instead, they usually support some kind of online information and/or communication system, like online bibliographic databases and online communities. These services are considered online if they serve many users, even if they are not publicly available.
3. *Dynamics*: The state of the stored networks changes every time the state of the system changes. For instance, each time a new user is added or two users interact, the network changes to reflect these events.

The importance of the online social networks and the Social Web for the development of the social networks field must be stressed. It is at least two-fold: lowering the costs of data collection and accessing big and complex automatically collected datasets from different sources. The collection of data using surveys or direct observation for dozens or hundreds of elements may require half of the resources of a research project [81]. Online social networks provide data sets containing thousands or millions of elements. Even more, there is much useful information –attributes and relations– that can be recorded automatically and precisely, e.g. ids, timestamps, and locations. Additionally, the collection can be incremental, that is, a data set can be progressively enriched when the information becomes available. Finally, data collection can be continuous, producing longitudinal data at temporal granularities which are impossible with the traditional approach.

On the other hand, the usual semi-manual data management approach cannot cope with the requirements of these unprecedented complex and big data sets. What we need is a set of standard tools to extract from a data set the information actually sought by the researcher, or to integrate two or more different data sets that describe the same network. For instance, consider a data set of a few years of interactions of a popular social web site: several terabytes of data with hundreds of transactions each second. Such data set is completely intractable from a network analysis point of view, some kind of data preparation is required. For example, consider this task: organize the data in a sequence of temporal slices, and for each slice aggregate the information available: grouping actors and relations, and aggregating attributes. Due to the lack of standard and appropriate data management tools, most of this work is made by research team members programming ad-hoc software to extract the network information needed from the data set. It is better, in terms of costs, repeatability, and reuse, to extract data with standard data management tools.

2.1.3 Social Networks Analysis

Breiger [21] introduces social networks, and defines social networks analysis (SNA) as follows: “Study of social relationships among actors –whether individual human beings or animals of other species, small groups or economic organizations, occupations or social classes, nations or military alliances– is fundamental to the social sciences. Social network analysis may be defined as the disciplined inquiry into the patterning of relations among social actors, as well as the patterning of relationships among actors at different levels of analysis (such as persons and groups).” Scott [98, Ch. 2] discusses the history of social networks analysis

since its origins in the early 1930s. In the SNA lineage there are three main traditions. *Sociometric analysis*, focused on small groups, is rooted in the gestalt tradition in psychology, which stresses the organized patterns through which thoughts and perceptions are structured. Moreno proposed the sociometric star, where the structure of relations around an individual is drawn as arrows between him or her and the others members of the group. He generalized this idea in the sociogram, which can be seen as the union of sociometric stars for a given set of individuals. Later, in the 1950s, Cartwright and Harary generalized the ideas of Moreno, using graph theory. In parallel, a group of researchers at Harvard University started their work with similar goals –to discover the subgroup structure of social systems– but focused on large-scale systems. Given the amount of data and the processing tools available at the time, one of the main technical challenges –from a network analysis point of view– was the reduction of data without loss of meaning. Later, in the 1950s, Manchester anthropologists built their work upon these two traditions. Barnes, Bott and Nadel, and later Mitchell, worked with many kinds of relations in the networks under study. They focused their work on the study of structure. These three groups converged in the 1960s and 1970s, when contemporary social network analysis was forged, again at Harvard. Wasserman and Faust give an exhaustive review of the field in their book (1994) [110], and a good hands-on reference for social network analysis has been published on the Web by Hanneman and Riddle [56].

In the last decade, social networks have been re-discovered by researchers from a wide range of disciplines. Given this renewed interest, social networks studies have benefited from these different points of view, generating numerous new research topics. For instance, Butts [23] analyzed the computational complexity of social networks and its impact on data simplification techniques like structural equivalence, ranking the complexity of known public domain data sets; Jin et al. [65] developed simple models of growth for social networks, taking into account the effects of mutual friends, decay of friendship and a maximum number of friends by individual; Newman [84] discussed some properties that differentiate social networks from other types of networks, e.g. non-trivial clustering; and Dodds [37] reported his work on performing an email routing experiment, inspired by Milgram’s mail experiment [50].

Algorithms and data structures used to perform network analysis in main memory are well known; see Brandes and Erlebach (eds.) [19]. The authors in that reference organize structural analysis according to three levels:

- Actor level. At this level, the basic question that needs to be answered, from a SN point of view, is how “important” an actor is to the network. Here, the exact definition of importance depends of the domain; for instance, a common importance measure in social networks is betweenness centrality (local or global), which is based on how many shortest paths go through an actor. Other approaches are discussed by White [112].
- Group Level. At this level, the main purpose of the analysis is to recognize, search and compare groups of nodes, usually with specific connections patterns or attribute values. In some application domains it is critical to be able to recognize and search user-defined patterns that, again, have domain dependent meaning. For instance, Newman studies scientific collaboration networks [82], Vázquez [109] discusses the importance of motifs (patterns) on transcriptional regulatory networks of *E. coli*(gut bacteria) and *S. cerevisiae* (yeast). Valverde and Solé study the relation between network growth and motifs in software architecture [105]. Analysis performed at this level allows also to

break down big networks into meaningful smaller pieces.

- Network Level: here, besides the properties needed to characterize social networks (e.g. diameter, clustering coefficient, etc.), it is also of interest to compare networks and to fit them into statistical models.

2.2 Database Theoretical Framework

To provide formal semantics and study the expressiveness and complexity of our social networks query language, we use well known database languages and formalisms: *Datalog*, *GraphLog*, and *second order tuple-generating dependencies (SO tgds)*.

Datalog is a query language based on formal logic that has been extensively studied, providing a good framework to characterize other query languages.

SO tgds are database formalisms to specify compositions of schema mappings. We use them to provide functional computation of new values in the proposed query language.

2.2.1 Datalog

There exist many variants of *Datalog*. In the presentation below we focus on the characteristics and properties that are used in the definition of the data model. (for further details and proofs see Abiteboul et al. [2] and Levene and Loizou. [73, ch. 9]).

A *term* is either a variable or a constant. An *atom* is either a *predicate formula* $p(x_1, \dots, x_n)$, where p is a predicate name and each x_i is a term, or an *equality formula* $t_1 = t_2$ where t_1 and t_2 are terms. A *literal* is either an atom (a *positive literal* L) or the negation of an atom (a *negative literal* $\neg L$).

A *Datalog rule* is an expression $H \leftarrow B$ where H is a positive literal called the *head*³ of the rule and B is a set of literals called the *body*. A rule is *ground* if it does not have any variables. A ground rule with an empty body is called a *fact*.

A *Datalog program* Π is a finite set of Datalog rules. The set of facts occurring in Π , denoted $\text{facts}(\Pi)$, is called the *initial database* of Π . A predicate is *extensional* in Π if it occurs only in $\text{facts}(\Pi)$, otherwise it is called *intensional*.

Definition 1 (Linear Recursive Datalog) *A Datalog rule is linear recursive if the predicate in the head of the rule occurs exactly once in the body of the rule. A Datalog program is linear recursive if at least one of its rules is linear recursive.*

A *substitution* θ is a set of assignments $\{x_1/t_1, \dots, x_n/t_n\}$, where each x_i is a variable and

³We may assume that all heads of rules have only variables by adding the corresponding equality formula to its body.

each t_i is a term. Given a rule r , we denote by $\theta(r)$ the rule resulting from substituting the variable x_i for the term t_i in each literal of r .

The *meaning* of a Datalog program Π , denoted $\text{facts}^*(\Pi)$, is the database resulting from adding to the initial database of Π as many new facts of the form $\theta(L)$ as possible, where θ is a substitution that makes a rule r in Π true and L is the head of r . Then the rules are applied repeatedly and new facts are added to the database until this iteration stabilizes, i.e., until a *fixpoint* is reached.

A *Datalog query* Q is a pair (Π, L) where Π is a Datalog program and L is a positive (goal) literal. The *answer* to Q over database $D = \text{facts}(\Pi)$, denoted $\text{ans}_d(Q, D)$ is defined as the set of substitutions $\{\theta \mid \theta(L) \in \text{facts}^*(\Pi)\}$.

2.2.2 GraphLog

A specially interesting subset of *Datalog* is *GraphLog* which additionally provides a graphical syntax whose semantics is described in *Datalog*. *GraphLog* provides several features on the lines of the proposed data model.

Definition 2 (Graphlog Query Graph) *A query graph [30] G_p is a directed labeled multigraph with a distinguished edge*

$$(N, E, L_N, L_E, \iota, \nu, \varepsilon, e, L_e)$$

where: N is a finite set of nodes, E is a finite set of edges, L_N is a set of sequences of variables, L_E is a set of literals and closure literals (literals followed by the positive closure operator), ι is the incidence function, ν is the node labeling function, ε is the edge labeling function, e is the distinguished edge, and L_e is a set of positive literals; there are no isolated nodes. Edges may be labeled with literals, closure literals or path expressions, except for the distinguished edge which must be labeled with a positive non-closure literal. The distinguished edge denotes the result of the query graph when the rest of the graph matches the data source.

Definition 3 (GraphLog) *GraphLog [30] is the query language defined by the set of graphical queries (finite sets of query graphs) \mathcal{G} , whose query graphs do not form cyclic dependencies. There is a dependence between two query graphs if one references the distinguished edge of the other. The meaning of a graphical query \mathcal{G} can be expressed using stratified Datalog.*

GraphLog was a seminal query language for graph data, designed to be expressive while at the same time having low computational complexity. Apart from standard features, it includes aggregation and transitive closure making it suitable for many SN queries. However, *GraphLog* does not provide functionality to compute new objects/values, a crucial requirement for SN.

GraphLog queries are based on *graph patterns* that should be present for a *distinguished edge* to define a relation.

Definition 4 (Directed Labeled Multigraph with a Distinguished Edge) A directed labeled multigraph with a distinguished edge $G_{\varepsilon(e)}$ is a *ninetuple*

$$(N, E, L_N, L_E, \iota, \nu, \varepsilon, e, L_e)$$

where $(N, E \cup e, L_N, L_E \cup L_e, \iota, \nu, \varepsilon)$ is a directed labeled multigraph, $\varepsilon(e) \in L_e$ and $\varepsilon(e') \in L_E$ for each $e' \in E$.

Graph patterns, formally defined as *query graphs*, are directed labeled multigraphs with a distinguished edge, without isolated nodes, and having the following properties:

- The nodes are labeled by sequences of variables.
- Each edge is labelled by a literal (i.e. a positive or negative occurrence of a predicate applied to a sequence of variables and constants) or by a *closure literal* (a literal followed by the positive closure operator $^+$). Closure literals may only appear between nodes labelled by sequences of the same length.
- The distinguished edge must be labelled by a positive non-closure literal.

Definition 5 (Query Graph) A query graph G_p is a directed labeled multigraph with a distinguished edge

$$(N, E, L_N, L_E, \iota, \nu, \varepsilon, e, L_e)$$

where L_N is a set of sequences of variables, L_E is a set of literals and closure literals, L_e is a set of positive literals, and there are no isolated nodes (i.e., for each $n \in N$ there exist $n' \in N$, $e' \in E$ such that either $\iota(e') = (n, n')$ or $\iota(e') = (n', n)$), and such that for each $e' \in E \cup \{e\}$ with $\iota(e') = (n_1, n_2)$, $|\nu(n_1)| = k_1$, $|\nu(n_2)| = k_2$, and $\varepsilon(e') = s$, where s has l positions, the arity of s is $k_1 + k_2 + l$. If $k_1 \neq k_2$, then s is not a closure literal.

Formally the semantics of query graphs is defined by a translation to a *Datalog* program.

Definition 6 (Logical Translation) The logical translation function λ is a function from query graphs to logic programs given by $\lambda(G_{p_0}) = \mathcal{P}$ where

$$G_{p_0} = (N, \{e_1, \dots, e_k\}, L_N, L_E, \iota, \nu, \varepsilon, e_0, L_{e_0})$$

and such that the rule

$$s_0 \leftarrow s_1, \dots, s_k \tag{2.1}$$

is in \mathcal{P} , where, for $0 \leq i \leq k$, if $\varepsilon(e_i) = s$, $\iota(e_i) = (n_1, n_2)$, $\nu(n_1) = \overline{X_1}$ and $\nu(n_2) = \overline{X_2}$, then

1. if s is $p(\overline{X_3})$ (resp. $\neg p(\overline{X_3})$), the s_i is $p(\overline{X_1}, \overline{X_2}, \overline{X_3})$ (resp. $\neg p(\overline{X_1}, \overline{X_2}, \overline{X_3})$),
2. if s is $=$ (resp. \neq), then s_i is $\overline{X_1} = \overline{X_2}$ (resp. $\overline{X_1} \neq \overline{X_2}$)⁴,
3. if s is $p(\overline{X_3})^+$ (resp. $\neg p(\overline{X_3})^+$), then s_i is $p'(\overline{X_1}, \overline{X_2}, \overline{X_3})$ (resp. $\neg p'(\overline{X_1}, \overline{X_2}, \overline{X_3})$) and the following two rules are also in \mathcal{P} :

$$p'(\overline{X}, \overline{Y}, \overline{W}) \leftarrow p(\overline{X}, \overline{Y}, \overline{W}) \tag{2.2}$$

⁴More precisely, the notation stands for a sequence of equality (resp. inequality) atoms, one for each pair of variables in the same component of each sequence. Note, however, that equality atoms are seldom (if ever) used.

$$\begin{aligned}
p'(\overline{X}, \overline{Y}, \overline{W}) &\leftarrow p(\overline{X}, \overline{Z}, \overline{W}), \\
&p'(\overline{Z}, \overline{Y}, \overline{W}).
\end{aligned}
\tag{2.3}$$

where $|\overline{X}| = |\overline{Y}| = |\overline{Z}| = |\overline{X}_1| = |\overline{X}_2|$, $|\overline{W}| = |\overline{X}_3|$ and there are no repeated variables in $|\overline{X}|$, $|\overline{Y}|$, $|\overline{Z}|$, $|\overline{W}|$.

and no other rules are in \mathcal{P} .

A query graph corresponds to a logical rule (plus the transitive closure rules when needed). Therefore, it is possible to express a set of rules by means of a set of graphs.

Definition 7 (Graphical Query) *A graphical query \mathcal{G} is a finite set of query graphs whose edge labels contain two classes of predicate symbols: IDB predicates, denoted p, p_1, p_2, \dots , which are the ones that appear in the label of the distinguished edge of some query graph; and EDB predicates, denoted q, q_1, q_2, \dots , which are the ones that do not appear in the label of the distinguished edge of any query graph.*

As with logical programs, we will associate with a graphical query a graph with information about its structure.

Definition 8 (Dependence Graph) *The dependence graph of a graphical query \mathcal{G} is a directed graph whose nodes are the IDB and EDB predicates that appear in the edge labels of the query graphs in \mathcal{G} and such that there is an edge from p_j (resp. q_j) to p_i iff there is a query graph G_{p_i} in \mathcal{G} whose distinguished edge is labeled p_i and which has p_j (resp. q_j) labeling some non-distinguished edge of G_{p_i} .*

The definition of the logical translation function λ is extended to operate on graphical queries by simply taking the union of the rules generated for each query graph. The semantics of a graphical query is determined by the usual semantics for the associated set of stratified *Datalog* rules.

We allow as expressions of the *GraphLog* query language only those graphical queries with an acyclic dependence graph. Note that, although we disallow explicit recursion, recursion is nevertheless implicit in the use of closure literals.

Definition 9 (*GraphLog*) *GraphLog is the query language defined by the set of graphical queries \mathcal{G} whose dependence graph is acyclic. The meaning of a graphical query \mathcal{G} is the meaning of the program $\lambda(\mathcal{G})$ under stratified *Datalog* semantics.*

2.2.3 Second Order tgds

Second-order tuple-generating dependencies (*SO tgds*) are a type of database constraint formalism introduced to specify composition of schema mappings [45].

We use *SO tgds* to formalize the semantics of the construction stage of evaluation of queries. Recall that this stage may require to compute new values, a feature that *SO tgds*

may provide while keeping complexity below acceptable bounds, as in the case of *Datalog*.

First, we define a simpler formalism which is not enough for the requirements of the language, but is the base of *SO tgds*: source-to-target tuple-generating dependency.

Definition 10 (Source-to-target tgd) *Let S be a source schema and T a target schema. A source-to-target tgd is a first-order formula of the form*

$$\forall x(\phi_S(x) \rightarrow \exists y\psi_T(x, y)),$$

where $\phi_S(x)$ is a conjunction of atomic formulas over S and $\psi_T(x, y)$ is a conjunction of atomic formulas over T . Note that x, y are tuples/sets of variables. We assume that every variable in x appears in ϕ_S .

Definition 11 (Full source-to-target tgd) *A full source-to-target tgd is a source-to-target tgd of the form*

$$\forall x(\phi_S(x) \rightarrow \psi_T(x)),$$

where $\phi_S(x)$ is a conjunction of atomic formulas over S and $\psi_T(x, y)$ is a conjunction of atomic formulas over T . We again assume that every variable in x appears in ϕ_S .

Source-to-target tgds have been used to formalize data exchange. Moreover, they have been used in data integration scenarios under the name of GLAV assertions.

Example 1 *Consider the following three schemas S_1 , S_2 and S_3 . Schema S_1 consists of a single binary relation symbol *Takes*, that associates student names with the courses they take. Schema S_2 consists of a similar binary relation symbol *Takes₁* and of an additional binary relation symbol *Student* that associates each student name with a student id. Schema S_3 consists of one binary relation symbol *Enrollment* that associates student ids with the courses the students take. Consider now the schema mappings $M_{12} = (S_1, S_2, \Sigma_{12})$ and $M_{12} = (S_2, S_3, \Sigma_{23})$, where*

$$\Sigma_{12} = \{\forall n\forall c(\text{Takes}(n, c) \rightarrow \text{Takes}_1(n, c)),$$

$$\forall n\forall c(\text{Takes}(n, c) \rightarrow \exists s\text{Student}(n, s))\}$$

$$\Sigma_{23} = \{\forall n\forall s\forall c(\text{Student}(n, s) \wedge \text{Takes}_1(n, c) \rightarrow \text{Enrollment}(s, c))\}$$

*These three formulas are source-to-target tgds. The second formula in Σ_{12} is an example of a source-to-target tgd that is not full, while the other two formulas are full source-to-target tgds. The first mapping, associated with the set Σ_{12} of formulas, requires that copies of the tuples in *Takes* must exist in *Takes₁* and, moreover, that each student name must be associated with some student id (s) in *Student*. The second mapping, associated with the formula in Σ_{23} , requires that pairs of student id and course must exist in the relation *Enrollment*, provided that they are associated with the same student name.*

The social network query language requires the functional creation of values; *source-to-target tgds* are not enough. A slight extension with existential second-order features, *Second-Order tgs*, are *source-to-target tgds* extended with existentially quantified functions and equalities. A particular case of *SO tgds* with predefined functions provides the required functionality.

Definition 12 (*SO tgds*) Let \mathbf{x} be a collection of variables and \mathbf{f} be a collection of function symbols. A term is defined as follows: every variable in \mathbf{x} is a term, and if f is a k -ary function symbol in \mathbf{f} and t_1, \dots, t_k are terms, then $f(t_1, \dots, t_k)$ is a term. Let \mathbf{S} be a source schema and \mathbf{T} a target schema. A second order tuple-generating dependency (*SO tgd*) is a formula of the form:

$$\exists \mathbf{f}((\forall \mathbf{x}_1(\phi_1 \rightarrow \psi_1)) \wedge \dots \wedge (\forall \mathbf{x}_n(\phi_n \rightarrow \psi_n)))$$

1. Each member of \mathbf{f} is a function symbol.
2. Each ϕ_i is a conjunction of
 - atomic formulas of the form $R(y_1, \dots, y_k)$, where R is a k -ary relation symbol of schema \mathbf{S} and y_1, \dots, y_k are variables in \mathbf{x}_i , not necessarily distinct, and
 - equalities of the form $t = t'$ where t and t' are terms based on \mathbf{x}_i and \mathbf{f} .
3. Each ψ_i is a conjunction of atomic formulas $S(t_1, \dots, t_l)$ where S is an l -ary relation symbol of schema \mathbf{T} and t_1, \dots, t_l are terms based on \mathbf{x}_i and \mathbf{f} .
4. Each variable in \mathbf{x}_i is a safe term with respect to ϕ_i and \mathbf{f} . A safe term with respect to ϕ_i and \mathbf{f} is defined recursively as one of the following: (a) a variable x occurring in a relational atomic formula of ϕ_i , (b) a variable x occurring in an equality term of the form $x = t$ or $t = x$ of ϕ_i , where t is a safe term with respect to ϕ_i and \mathbf{f} , or (c) a term $f(t_1, \dots, t_k)$ where f is in \mathbf{f} and t_1, \dots, t_k are safe terms with respect to ϕ_i and \mathbf{f} .

The fourth condition is a “safety” assumption –not to be confused with the safe condition in Datalog– that makes the second-order tgds domain independent (so that their truth does not depend on any underlying domain, but only on the “active domain” of elements that appear in tuples in the source). In the case of first-order tgds, where equalities are not present, this condition becomes a simpler one by requiring that every universally quantified variable appear in one of the relational atomic formulas in the left-hand side of the tgd. This condition is not always made explicit in the literature in the definition of first-order tgds, although it should be.

The following formula is a valid example of a second-order tgd where all universally quantified variables are safe:

$$\exists f \forall x \forall y \forall z (R(x) \wedge (y = f(z)) \wedge (z = x) \rightarrow S(x, y, z))$$

Example 2 Consider the following three schemas S_1 , S_2 and S_3 . Schema S_1 consists of a single unary relation symbol *Emp* of employees. Schema S_2 consists of one binary relation symbol *Mgr1*, that associates each employee with a manager. Schema S_3 consists of a similar binary relation symbol *Mgr* and an additional unary relation symbol *SelfMgr*, intended to store employees who are their own managers. Consider now the schema mappings $M_{12} = (S_1, S_2, \Sigma_{12})$ and $M_{23} = (S_2, S_3, \Sigma_{23})$, where

$$\Sigma_{12} = \{\forall e(Emp(e) \rightarrow \exists m Mgr_1(e, m))\}$$

$$\Sigma_{23} = \{\forall e \forall m(Mgr_1(e, m) \rightarrow Mgr(e, m)), \\ \{\forall e(Mgr_1(e, e) \rightarrow SelfMgr(e))\}\}$$

*It is easy to verify that the composition of M_{12} and M_{23} is M_{13} , where Σ_{13} is the following *SO tgd*:*

$$\exists f(\forall e(Emp(e) \rightarrow Mgr(e, f(e))) \wedge \forall e(Emp(e) \wedge (e = f(e)) \rightarrow SelfMgr(e)))$$

Remark. Note that in the definition of the construction stage of social network queries we only need a restricted form of the *SO tgd*s where the existential variables for functions are known constants (see Chapter 5).

Chapter 3

Social Networks Data Management Requirements

One of the hypotheses driving this work is that a data model tailored to the SN community will have a positive impact in the development of the field, improving productivity of current users as well as attracting new ones. To succeed, a data model must capture the representation and manipulation needs of the domain, while being accessible to the users, and having a complexity that does not forbid the existence of actual implementations. The design of such a data model requires a deep understanding of the problem and several design tradeoffs, such as expressiveness vs. feasibility, and generality vs. appeal to potential users.

We use social networks analysis as our main source of requirements, because it has a century-long tradition studying social networks, and it provides a well-defined and documented methodological framework. However, until recently most of the data sets studied were small, with dozens to hundreds of elements. To complement this view, we include ideas from online social networks and the Social Web, which provide examples of data sets at bigger scales.

This chapter starts with the description of the sources of requirements: social network analysis and Freeman’s maximal structure experiment, and online social networks. Then, we discuss the main ideas behind the design. Finally, we present the requirements for the social networks data model in two parts: data structure and query language.

3.1 Sources of Requirements

To identify and formalize the requirements for social networks data management, we used mainly two sources: the well-developed and well-defined SNA practice, and the new dynamic demands imposed by the social web and online social networks.

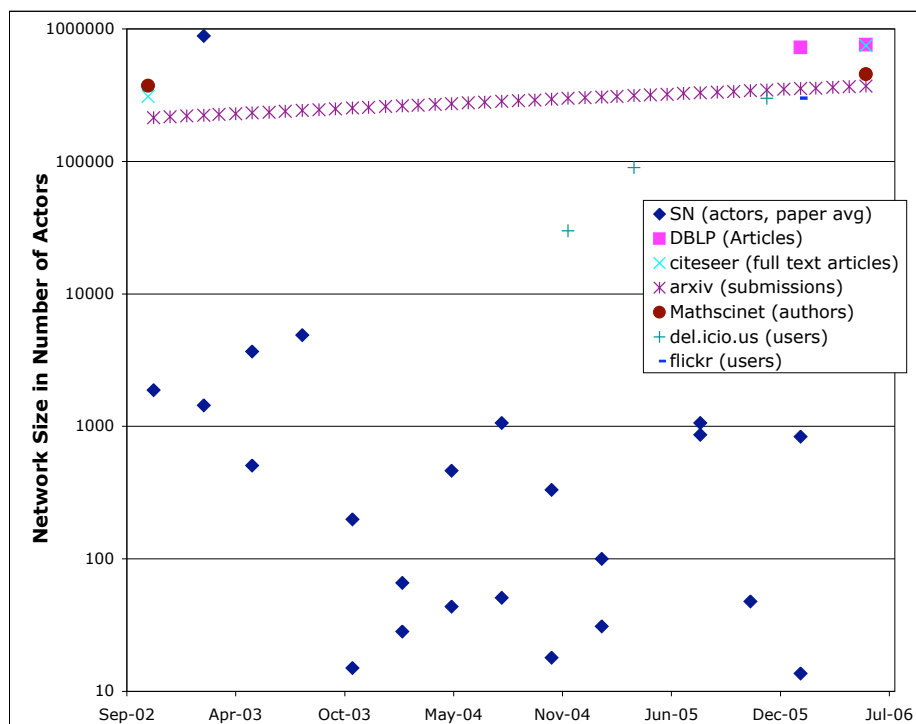


Figure 3.1: *Classical vs. New Online Social Networks*. The graphic shows a gap of orders of magnitude in network sizes (in actors) between classical and new online social networks. Classical networks are represented by data sets reported in articles published in the Social Networks journal –SN series in the graphic– whose sizes are all but one below the 10,000 boundary. (Data from our initial survey.)

3.1.1 SNA Practice

Freeman [50] discusses the history of SNA arguing that the foundational aspects were already present long before Moreno started his works in the early 1930s. The three aspects that shaped SNA practice are: (1) a structural intuition, (2) a sustained effort towards systematic data collection, and (3) the development of visualization devices, and mathematical and computational models.

In order to study actual social networks’ use cases, at the beginning of this work, we analyzed a sample of the works published in the journal *Social Networks*. Table A.1, in the appendix, lists each article in the sample, describes briefly the data used in each one, and summarizes how the characteristic SNA data workflow is present in each article. Table A.2 shows the most relevant features of each data set used in the analysis (part of this data was used in Figure 3.1 to draw the SN series). The most relevant findings can be grouped as follows:

- *Characteristic data workflow*: From Table A.1 it is clear that a characteristic data workflow, captured diagrammatically in Figure 3.2, is present in SNA practice. It is consistent with practices described in the reference works of Wasserman and Faust [110],

Scott [98] and Hanneman [56]. Furthermore, its sequential nature –in particular the lack of large scale reuse– is emphasized by the fact that in most cases there is no record of availability of data after the studies.

- *Interoperation, reuse and provenance*: Several articles –nine over twenty four– use data from previous studies, but in most of these cases the reused data sets are small (less than twenty elements in each set of actors) and/or simple (one or two-mode networks without attributes). We hypothesize that this is an effect of the prohibitive cost of reusing data due to lack of proper metadata –meaning and provenance– in existent data sets.
- *Unit of observation vs. Unit of analysis*: In many cases, the unit of analysis is different from the unit of observation due, for instance, to the fact that most data from surveys is naturally in ego-centered form (see Chapter 2). Furthermore, in several cases the analysis is made at more than one level, thus demanding a stage where network data is not only stored but transformed to obtain the appropriate data at the desired analysis levels.

Freeman’s Maximal Structure Experiment. From what we have presented in previous sections, it is clear that modeling more sophisticated data sets should be an important methodological concern of the SNA community. Actually, when Freeman [51, ch. 1] defined his *maximal structure experiment*, he foresaw many of the key issues present today in online networks.

Freeman starts from the simplest case: a single relation recorded at a single time over an undifferentiated and unchanging population. He then defines an experiment which uses two kinds of information: a set of social units (which at the lowest possible level refers to individuals, i.e. persons), and a set of pairs of social units that exhibit some social relation of interest between them. From this basic setting, Freeman progressively builds the maximal social structure experiment by adding the following elements:

1. more than a single kind of relations,
2. two or more types or levels of social units,
3. structures that change through time,
4. sets of social units that grow or shrink,
5. attributes of social units, and
6. attributes that change.

To complete the requirements for a data model capable of representing new online social networks, two elements must be added: attributes of relations, and n-ary relations, i.e. relations linking more than two actors, also called supradyadic relations (see for instance Bonacich et al [16]).

3.1.2 Social Web and other Online Social Networks

The development of the Web into the Web 2.0, and later into the Social Web, created a fruitful space for the production of large and complex social networks. The widespread embracing of social networking services—such as Facebook, MySpace and LinkedIn—as the indispensable tools to manage people’s digital and physical lives has resulted in rapidly growing amounts of social activity data. Furthermore, the social information represented by many sites usually includes not only people but a great variety of objects (generically referred to as *actors* in the social networking literature) and relations [6]: photographs (Flickr), other sites (del.icio.us), places (Yahoo! Travel), goods (Amazon), and so on. In all these services, each time a user sends a message, chooses a tag or annotates a resource, one or multiple relations are recorded. Thus, the automatic recording of social interactions over the Web, and the tagging and annotation of resources, are creating social networks involving persons, groups, a rich variety of objects, and their attributes [66, 81]. These social networks have to be managed, queried, and transformed.

Researchers, developers, and users have realized that there is useful information in these networks [81]. Consequently, they have already been used in structural analysis experiments; see for instance the works of Finin et al [48], and Ereteo [44]. Many of the techniques used come from social networks analysis (SNA). Nevertheless, the processing and managing of this huge amount of data is still an unsolved problem for the common user and developer. The problem is not only the amount of data available, but also the means of querying, mixing and transforming such data with standard models. Furthermore, most data formats currently in use do not have explicit semantics nor support for provenance [81].

Consider, for instance, Facebook’s social network which contains hundreds of millions of users, objects, and interactions. The service main user interface is primarily designed to support interactions among users and for the manual browsing of the network. However, this access to the network is not adequate for developers and researchers. The objective of developers of applications for the Facebook environment is to extend and enrich the user experience while logged in the service. To achieve this purpose, developers must have access beyond the immediate neighborhood of the user to identify objects and structures that may be of interest for the user. On the other hand, given that the detailed structural analysis of the whole Facebook network is well beyond the current feasibility limit, researchers need the means to identify and extract significative but tractable subnetworks. To address developers needs, Facebook provides its Graph API¹ which supports the access of applications to the collections of elements in the direct neighborhood of a user. Facebook also provides the Facebook Query Language² (FQL), an SQL-like query language that accesses the elements around a node as tables. Support for researchers needs is not as well defined. Several considerations, user privacy and system performance among them, may have prevented its development. Nevertheless there are some SNA works published regarding subnetworks from Facebook³ (see for example Lewis et al. [74]).

¹<http://developers.facebook.com/docs/reference/api/> (Login required.)

²<http://developers.facebook.com/docs/reference/fql/> (Login required.)

³“Anatomy of Facebook”

<https://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859>

Several of these challenges have already been voiced. Fifteen years ago Freeman defined the *maximal structure experiment* that extended the basic network representation to include attributes as well as to accommodate changes over time [51]. More recently, Carley [25] and Mika [81] identified the need to improve both network data formats in the context of the Social Web as well as data management services for large and dynamic social networks. In this work, we consider online social networks and applications primarily as sources of social networks data.

3.2 Data Management for Social Networks

A social networks data management infrastructure, with an appropriate theoretical support, should provide access to the data through standard operations at the level of abstraction required by the programmer, allowing the successful separation of data manipulation chores from application logic. Thus, issues like data representation and storage, integrity, security, and efficient query evaluation, are resolved by the DBMS and provided as services to applications.

The successful implementation of such infrastructure should provide the same benefits to the social networks field, as the relational model and RDBMS provide to the business domain.

3.2.1 Social Network Data Lifecycle

We consider that the correct context for the study of management of social networks data is a network of producers and consumers. Producers collect social network data by diverse means; for instance, direct observation, surveys, and/or automatically capturing the activities of users. Consumers, on the other hand, require access to the captured data to extract the subsets that are relevant for their applications. If the unit of collection for the data does not coincide with their unit of modeling, the model should provide a facility to transform from one to the other. For instance, SNA can be viewed as a consumer of social networks to be analyzed and a producer of structural measures; annotators and users are producing and updating social networks; developers of applications are consuming and producing networks, etc.

In this context, a DBMS implementing the model articulates the network of producers and consumers (see Figure 3.2). To achieve this goal, such DBMS must provide data management services to each consumer/producer:

- to delete, insert, and update social networks and their elements;
- to extract a subnetwork of a given social network;
- to transform a social network into another, mapping the source schema to a new schema;
- to share the social network database with multiple users; and

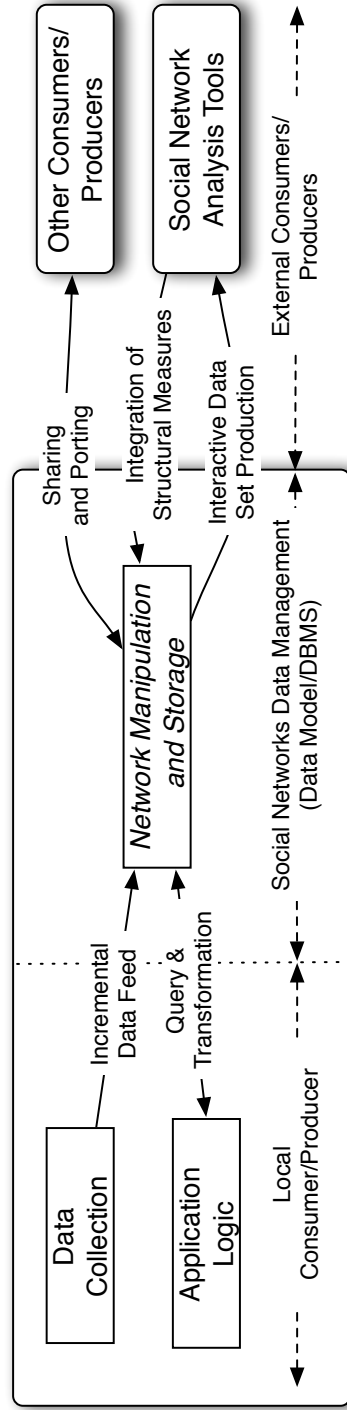


Figure 3.2: Data Lifecycle Workflow for Social Networks. Each social application is a consumer/producer of social networks, producing and/or collecting network data, and consuming data produced by other applications. The need for a standard representation and query language follows.

- to import and export a social network to a portable format to be integrated in the database or to be used in other applications.

The fluid interactions among all the components produce the data workflow of SN data depicted in Figure 3.2, which along with the requirements mentioned before, points to an abstract data model and query language [95].

3.2.2 Social Network Analysis as a Consumer/Producer

We consider SNA the archetypical consumer/producer of social networks data. SNA practitioners directly collect social network data, and produce new networks and measures through analysis. On the other hand, SNA consumes networks produced by others. Consequently, the data model for social networks data is designed to provide data managing services to consumer/producers like SNA tools, not as an alternative analysis tool.

As part of the definition of the requirements for the SN data model, we surveyed the SNA community data management practices from the Social Networks journal, and from well known reference books and software tools.

As an illustrative example, Table 3.1 systematizes use cases from the book *Exploratory Social Network Analysis with Pajek* [35], which documents in detail the features of the Pajek SNA tool. Each chapter of the book addresses a characteristic SNA activity or approach, and for each one Table 3.1 briefly describes at least one relevant data-management use case.

Table 3.1: Archetypical social networks data management operations from the book *Exploratory Social Network Analysis with Pajek* [35], which also recurrently appear in the literature [35, 98, 110].

Chapter Title	Use Case	Description
Looking for Social Structure	Directed to undirected Binary Relations (<i>arcs to edges</i> in Pajek).	Replace all binary directed relations with undirected relations.
	Remove relations (<i>remove edges</i> in Pajek).	Remove all relations that do not fulfill a given condition.
Attributes and Relations	Extract a subnetwork based on attributes.	Extract the subnetwork consisting of the actors and relations whose attributes fulfill the given condition.
	Group actors based on attributes (<i>shrink network</i> in Pajek).	All actors that share the same value for a given attribute are coalesced into a single new actor. This may require the computation of aggregate functions, such as the count of actors in each group, and the assignation of a new id to each new actor.

Continued on next page

Table 3.1 – continued from previous page		
Chapter Title	Use Case	Description
	Selective grouping of actors (<i>contextual view</i> in Pajek).	Similar to the previous, but all except one group of actors are coalesced.
Cohesive Sub-groups	Extract the subnetwork induced by cliques of size n .	Extract all cliques of the given size n , join them in the result.
Sentiments and Friendship	Extract subnetwork by time attribute.	Extract all elements with the given timestamp.
Affiliations	Two-mode network to one-mode network.	Replace the paths of length two in the bipartite network by a new direct relation. It may require grouping and aggregation if several paths exists between the same pair of actors.
Center and Periphery	Group multiple binary relations (<i>remove multiple lines</i> in Pajek).	Coalesce all binary relations that fulfill the given condition. It may require to compute aggregate functions.
Brokers and Bridges	Extract the egonetwork of an actor.	Extract the given actor and its direct neighbors.
	Remove relations between groups (<i>remove lines between clusters</i> in Pajek).	Identify each group, then remove all relations between actors of different groups.
Diffusion	Selective counting of neighbors.	Count only the neighbors that fulfill a given condition.
	Operations between attributes (<i>divide vectors</i> in Pajek).	Requires functions over attributes values, e.g. arithmetic operations.
	Change relation direction based on attributes.	Invert relations that fulfill the given condition.
Prestige	Discretize an attribute.	Requires functions over attributes values, e.g. int, round, etc.
Ranking	Find triads by type.	Identify and extract motif consisting of three actors. In each triad some relations are allowed and some are forbidden. Requires negation.
Genealogies and Citations	Loop removal.	Extract all relations except loops (self relations).

3.2.3 Desiderata for a Data Model for Management of SN Data

The basic desiderata for managing social network data should include at least the following: 1) to be able to represent and store any kind of social network, including provenance information; 2) to be able to share and mix social networks (or parts of them) among users and applications; 3) to have a set of standard operations over these social networks, to be able to query, transform and update them (a query language).

A data model for social networks should provide support to each stage in the workflow of social networks analysis (before analysis itself); in a broader sense, to each stage in the life cycle of network data: creation (insertion), update, query, and archive; additionally, it should support and promote interoperation. For instance, given a data set over which some analysis has been performed, when some data change or some extra data become available, it is usually needed to rebuild almost by hand the entire data set to perform a new analysis with current tools. It is expected that a working data model allows changes in data with a minimum administrative burden, i.e., when new or updated data becomes available, a query should suffice to get an updated dataset which could, in turn, be analyzed with specific tools at the application level.

Social networks and social networks analysis have a common set of characteristics and operations that form an appropriate data model. The set of operations is one of the most evident components of a data model, because it is directly related to user requirements. From what we have discussed so far and a review of social network analysis tools [59], we group SNA operations in three levels of service: basic data management (update, archiving, transport, and integration of data), network oriented data manipulation (extract and transform relevant data and structures from the database), and domain specific data manipulation (analytical operations and processing).

A network data model should provide support for the first two levels over an appropriate set of data structures.

Natural candidates to provide the needed support, given the intrinsic nature of network data, are graph data models [8] and semi-structured data models, like RDF [7]. However, as far as we are aware, there is no complete and implemented model of these types, nor a model that provides the required query specificity. A bottom-up approach, starting from a well defined and domain-specific social networks model, can use the possible implementation advantages of a concrete and well-defined set of requisites, and borrow the conceptual framework of those general models.

3.3 Design Issues and Principles

This section presents the design issues and principles that should guide the design and implementation of the data model as a whole.

Progressive Data Collection, Data Integration and Reuse

The classical approach of data collection, where it can take up to fifty percent of the project resources [81], is being replaced by an approach more focused on sharing and reusing data. The availability of online social networks, the Social Web, and the scientific trend to maintain public data repositories creates many opportunities to reuse and integrate data [58].

The decreasing costs of data collection and the storage infrastructure poses another question: what should be recorded? In terms of SNA, what should be the unit of observation? Should we record every measure or event? Should data be cleaned or summarized? Whatever the decision may be, in a research initiative the data model must not restrict the granularity or variety of the recorded data.

Progressive data collection must also be considered. In some scenarios, periodically or almost constantly new data arrives that increments our knowledge of the network. For instance, Social Web services like Twitter, where users produce data (short messages) almost in a constant global stream, and where they also change the structure of the social network by selecting whom to follow. This situation calls for a semistructured data model that provides schema flexibility and metadata.

Filtering and Transforming Data

Besides the computational cost, the size of network data sets determines the viable options for data manipulation. For instance, visualization and graphical browsing, as strategies to find elements or discover structural patterns, are practical only for networks with a few dozens of nodes.

Furthermore, a scenario of sharing and reusing implies that a given data set may have data relevant to one study but not to another. To access the required data, a mechanism to specify a subnetwork of interest is necessary. This mechanism becomes mandatory as the size of the network grows.

It is possible as well that the observation unit of the source data set is not the expected modeling unit. For instance, when the data was recorded as a network of persons, but we need instead the network of companies where these persons work. This problem generalizes to the case when the schema of the result of a query is different from the schema of the source network; the source schema is transformed into the result schema.

Additionally, the cost of data collection may force the researcher to narrow the definition of the boundaries of the network under study, or to work with a sample of the network. However, there does not exist a simple criteria to sample an arbitrary network, while ensuring that its structural properties will be preserved. To avoid this problem SNA practitioners prefer to choose small and well defined groups: all the employees of a department of a company, all the students in a class, etc. The result of a query does not have the problem of random sampling, because the query itself defines bounds based on the network data.

Richness of data

SN representation requires a rich data structure because certain SN properties manifest themselves only when several facets are represented: e.g., searchability is present only when multiple relations between actors, representing different interaction domains, are modeled [111]. SN data representation needs to be flexible enough to incorporate on-the-fly attributes (e.g. for data curation). In addition, data is used and seen from diverse points of views and by different users. Hence, classical modeling, in terms of a fixed set of entities –attributes and relationships– does not work well. For example, in SNA, it is common for attributes to become actors, for aggregated data of actors to become attributes, and for relations to have arity greater than two, or a variable arity.

Appealing to the users

The data model must offer clear advantages in order to be used by the community. To facilitate the modeling of actual data management solutions in a domain it must offer a good level of abstraction, that is a set of abstract elements and tools that correlate directly with the element that exists in the universe of discourse. In the case of social networks, the data structure must have elements to represent actors, relations, and attributes. A good level of abstraction must be supported by data independence: applications must be impervious to the physical organization of the data on the storage media.

The query language benefits from a good level of abstraction, but it also requires simple syntax, easy to comprehend and learn. In this sense, it is expected that different subgroups of users have conflicting preferences: users without a programming background may prefer a graphical or query-by-example language, while advanced users and programmers may prefer a flexible and powerful text-based syntax, even if it has a steeper learning curve.

Scalability

Due to the growing volume of SN data, any transformation and query language for SN has a basic requirement: its scalability. A practicable transformation and query language should conform, from a theoretical perspective, to strict complexity bounds, and, because of practical concerns, be simple and modular while being sufficiently expressive.

We measure the size of the data managing problem in terms of number of elements: the size reflects the total number of actors, relations, attributes, and their associations. We define three scales:

- *Small Size Social Networks (with dozens to hundreds of items)*. Data is known by the user, and to some extent it is verifiable by hand (item by item or visually). At this level of data volumes it is possible for the user to get to know very well a given data set. He or she may know at a glance what is happening in the network. Query language is used to select parts of the network for analysis or to facilitate visualization through

filtering, and to change the modeling unit (transformation).

- *Medium Size Social Networks (with thousands to several millions of items)*. The user does not know the data a priori, possibly because it was not collected by the user, so the user must rely on the metadata (if any) that accompanies the data set. Furthermore, the user may need to integrate several data sets from different data sources. Data volumes forbid any manual inspection or visualization of the data set as a whole. At this level, data sets range from thousands to a few millions of triples rendering impractical any manual verification process. This situation is aggravated because in many cases the user is not involved in data collection; thus, he/she must rely on any metadata available to discern its model, provenance, and quality. The availability of means to manipulate the data set is critical to define and extract a potentially relevant subnetwork, and to transform the current model to the intended analysis model. The case-by-case programming of such tools may introduce biases and errors, difficult to detect and correct.

In this context we expect primarily research-oriented users who, when using the query language may state by themselves the required queries on the network under study. Being knowledgeable of the complexity of the problem, these users may tolerate not instantaneous response times as long as the results are reliable.

- *Large/Huge Volumes (dozens of millions and above)*. In this case, the problem reaches the hard constraints set by current technology: data storage capacity, data access bandwidth, computational costs of query processing. Large SN need a completely different approach regarding user interface and navigation and query tools, as well as storage and performance issues [29].

3.4 Proposed Solution

To solve the social networks data management problem, we propose a data model that fulfill the following requirements.

3.4.1 Data Structure Requirements

From the SNA domain, as we discussed above, Freeman's *maximal structure experiment* [51, ch.1] is a reference base for a SN data structure requirements. Freeman starts typifying a social network from the simplest case: a single relation recorded at a single time over an undifferentiated and unchanging population. Subsequently he defines an experiment which uses two types of information: a set of social units (which at the lowest possible level refers to individuals, e.g. persons), and a set of pairs of social units that exhibit some social relation of interest between them. From this basic setting, Freeman progressively builds up the maximal social structure experiment adding the following elements: (1) several kinds of relations, (2) two or more types or levels (groups) of social units, (3) structures that change through time, (4) sets of social units that grow or shrink, (5) attributes of social units, and (6) attributes that change. To cope with the requirements developed in recent years, two further elements

should be added: attributes of relations, and a variable number of participants in relations, i.e. relations linking a number of actors which is not fixed at modeling time [16].

The natural and traditional choice for representing social networks is to use graphs where actors are nodes and relations are edges. However, doing so limits the representation power to that of binary relations and forbids attributes on relations.

The requirements discussed above demand representations more elaborate than the simple graphs (or matrices) of the classical modeling. This challenge is not exclusive of the SN's domain and it has been explored extensively in other contexts [8,15,54]. From this background and trends in information exchange, indicating that all the information should be in the same data structure and that it should support the addition of arbitrary metadata (e.g. provenance), basic requirements for such a model emerge. First, attribute values should be part of the graph; and second, relations should be represented as nodes instead of edges or arcs, allowing the seamless representation of n-ary relations and attributes on relations over the same data structure.

Thus, our logical data structure, the social networks data model (SNDM), is a graph where actors, relations, and attributes are all modeled as nodes; and edges associate attributes with the actors or relations which they describe; and actors with the relations in which they participate.

In terms of the data structure that will represent social networks, each data element must support at least the following characteristics:

Actors. Each actor element requires a unique id to be distinguished among all the other actors; thus, an actor might be identified by its id and/or structural position. Additionally, each actor may be of different types and may have an arbitrary collection of attributes.

Relations. Each relation element also requires a unique id to be distinguished among all the other relations; thus, a relation might be identified by its id, structural position, and/or the participating actors. Each relation may be of different types and may have an arbitrary collection of attributes.

Attributes. Attributes associate values with meanings to an actor or relation, e.g. an integer value that should be interpreted as the age of the actor. Attributes do not require an id of their own, given that they are always associated with an object that already has one. Each attribute may be of only one type.

Networks. In a data management setting that requires sharing, reusing, and integration, metadata at the level of datasets (networks) is required. This metadata must provide at least provenance.

3.4.2 Query and Transformation Requirements

Table 3.2 presents a shorter list of generic use cases (Table 3.1), complemented with the features that the query language must provide to implement that use case. Each use case is selected for its relevance and justifies the inclusion of a query language feature.

Filtering by object identifiers and attributes. To be able to extract a subnetwork based on the identity of objects and values of attributes the query language must provide a mechanism to define filters.

Pattern matching and Negation. In social networks data management, a common task is to search for the occurrences of a given fixed structure. Pattern matching can be seen as a structural filter. From the use cases, it is clear that negation is required. For example, in certain situations, a successful match requires that a subnetwork of the pattern does not exist.

Pattern production and creation of new objects and values. The data collected by matching a pattern (and possibly a filter) in the source network is used to produce the result. In the simplest case, the found pattern, or a portion of it, may be copied into the result. However, some use cases require to compute new values or object identifiers using the data collected from the source. This is a transformation operation.

Induced Subgraphs. Some use cases require that once a subnetwork is identified, some additional elements are induced from the source and included in the result. For instance, once all neighbors of a given actor are identified, all relations among them are induced.

Transitive Closure. To capture certain groups, a fixed pattern is not enough, as in all the reachable actors from a given one. In this case, a more powerful operation is required, such as transitive closure, which allows to build a match by repeatedly applying a pattern in a sequence. Regarding complexity, a key design decision is the nature of the allowed results: only start and end points (e.g. start and end nodes in a path), all the matched subgraphs in the sequence (e.g. the path itself), and/or aggregated values from the matches (e.g. the length of the path).

Set theoretical operations. Given two social networks with compatible schemas at least two set theoretical operations are useful: union and difference. Set theoretical union can be used to integrate two networks; and the difference, to compare networks.

Table 3.2: Selected social network data management use cases and their requirements.

Use Case	Description	Required QL Features
1. Selecting Groups	Select a subnetwork of actors and relations that satisfy conditions on their attribute values and/or participation in certain relations.	Pattern matching, filtering by attributes values.
2. Promoting Attributes to Actors	From an actor A_1 and one of its attributes (att, v) produce a new actor $A_2 = f(v)$ and a new relation $R = g(A_1, v)$ (f and g functions): all actors with value v for att will be connected to A_2 .	Pattern matching, creation of new objects, pattern production.
3. Identifying Brokers	Each time a characteristic brokerage pattern is found, label the broker in the output accordingly. Some brokerage patterns require that certain relations do not exist.	Pattern matching, negation, pattern production.
4. Counting Binary Relations	Select all relations of a given type, group by participant actors, count. Produce only one relation per group with the new attribute count.	Pattern matching, aggregation, pattern production.
5. Ego-network selection	Select an actor along with all its direct neighbors, and the relations between them.	Pattern matching, induced subgraphs.
6. reachable neighborhood	Similar as above but selecting all reachable neighbors instead of only the immediately adjacent ones.	Pattern matching, transitive closure, induced subgraphs.

3.5 Related Work

3.5.1 Social Network Analysis

In this context, our initial survey was focused primarily on SNA software tools, e.g Pajek [35], Ucinet [59], and *network* and the *sna* packages of R [24,55,60]. These tools mainly addressed the implementation of algorithms for structural analysis and visualization, and they largely assume that the network data has been prepared by external means (i.e. manually or with other software). Huisman and van Duijn [59] provided a comprehensive survey of the SNA tools available at the time. We summarize their review from a data management point of view.

- *Data entry and manipulation*: All tools use file stored data structures, based on adjacency matrices, lists of nodes and edges, and attributes. Only one, NetMiner II, supports connection to a database. Common operations in this category are transformation (e.g. two-mode to one-mode, permuting and transposing), symmetrization, dichotomization, selection, and merge. All tools, except STRUCTURE, support file import and export to popular formats.
- *Visualization techniques*: Only Multinet, Netminer and Pajek include network visualization options. They use known graph drawing algorithms like Kamada-Kawai and Fruchterman-Rheingold.
- *Descriptive methods*: Common operations in this category are detection of cohesive groups and regions, centrality analysis (for individual and groups), ego networks analysis, and structural holes analysis. Some tools support binary operators and a layered treatment of networks.
- *Procedure based analysis*: Common operations in this category are cluster analysis, multidimensional and two-mode scaling, structural equivalence, and fitting core/periphery models.
- *Statistical modeling*: Operations range from simple statistics to model fitting. There is no clear common base between tools.

Most of the research activities on social networks, besides structural analysis itself, were performed with general-purpose applications, like spreadsheets, or with software tools especially developed for specific tasks, often coded by someone in the research team itself [63]. The researcher must get involved in data preparation at a low level, in a per file and per experiment basis. This situation has a bad impact in environments with high data throughput and in interoperation tasks [53,98].

The survey showed that SNA tools relied mostly on text files, with proprietary data formats, for networks persistent storage. These data file formats were based on adjacency matrices, lists of nodes and edges, or a combination of both. Some file structures are positional and others use different kinds of markup languages. Of these, just a few are supported by semistructured schemas (e.g. GraphML⁴ and DynetML [104]). Note that these formats are suitable for network manipulation and analysis only in main memory. However, if the data

⁴<http://graphml.graphdrawing.org/about.html>

set (a network or a set of networks) does not fit in main memory, it is not efficient to apply internal memory graph algorithms like the ones described by Brandes and Erlebach [19]. There exists a relevant literature on data structures and algorithms for graphs in external memory [67, 88, 97]. This topic is beyond the scope of this work and it will be touched only tangentially in this thesis.

An aspect that differentiates social networks from other networks is the fact that details about actors matter, i.e., attributes for every actor in the net must be recorded. However, data handling in SNA tools forbids the use of big and detailed networks, forcing researchers to select some attributes and to sample actors, which introduces biases that are difficult to measure. Instead of exposing increasingly complex file structures to the users and forcing them to develop their own tools directly on them, it would be better to provide a higher level of service –data management– raising the abstraction level of tools and languages.

The general situation of SNA tools, regarding data management, has not changed substantially. The most popular tools are under active development with regular releases of new versions and bug fixes, but this development is mainly focused on improving their analysis capabilities, not the data managing ones.

In the broader context of networks research, Network Workbench⁵ (NWB) addresses several relevant issues. NWB [18] is a large-scale network analysis, modeling and visualization toolkit for biomedical, social science and physics research. While it does not provide data managing functionalities (networks are still stored in text files), its purpose is to support the research data workflow in a consistent and flexible environment aware of the complexity of current data sets. NWB development started several years ago, and in september 2009 reached its official 1.0 release.

3.5.2 Data Models

The aim of the database approach is to provide the theoretical framework and the actual data management systems required in a given domain. To reach this goal, the characteristic requirements must be identified and a solution implemented as: a data structure capable of representing any possible object in the domain, and a feasible language to manipulate and extract the information from the data.

There is no one best data model for every possible problem domain; instead the database community has defined different models to face different kind of data and problems. We describe briefly some of the most relevant ones [73] and briefly discuss them in the context of the requirements presented above.

Relational Model. In this model all data is represented as mathematical relations, hence the name. Its main contribution is the separation of physical and logical levels, thus providing data independence. It has been the dominant data model in the last decades, due to its solid mathematics foundations and its suitability for business and managerial data. However, it

⁵<http://nwb.cns.iu.edu/index.html>

is not well suited for complex data types, and its implementations do not handle structural operations efficiently, e.g. pattern matching and path finding.

Semantic Models. These models focus on higher levels of abstraction, closer to application domains, which allow users to specify the complex structure of databases in familiar terms. The level of abstraction complicates the query language design, and implementation. Thus, this kind of models is mostly used as a conceptual tool in the design of databases; for instance, the entity-relationship (ER) model.

Object-Oriented Model. Object-oriented data models port the concepts of object-oriented programming languages to databases: inheritance, encapsulation, and overloading. This kind of models is well suited to deal with complex objects and data types. Each class of objects in the database not only holds a complex data structure, but also a set of methods to manipulate this kind of objects. The database is a network of complex objects. Query languages are object-oriented languages. In the case of social networks, actors and relations are simple objects, which do not require specialized methods.

Deductive Model. This model is rooted in mathematical logic, which provides a sound formal framework to study expressiveness and complexity. A deductive database is comprised by a list of logical facts that can be queried with logical rules. *Datalog*, presented in Section 2.2.1, follows this model. We use a translation to *Datalog* to study the properties of the proposed social networks data model. However, *Datalog* is too abstract to fulfill the stated requirements.

Graph Models. These models give the same importance to data elements and to the relations among them. Graph models differ in the complexity of their data structures, ranging from plain graphs to hypergraphs of several types; and there is a matching variety of query languages. Graph query languages usually support operations that identify the occurrences of a given structure in the database, and with this information produce the result. The result may be tuples, paths, graphs, or hypergraphs. The main challenges in the design of these query languages come from the broad range of operations that may be required. Operations that rely in the matching of fixed finite pattern may be costly but are usually considered feasible. However, there are operations with hard algorithmic complexity, for instance those for identifying recursive maximal structures. Despite the efforts of the database community, a standard for graph databases has been elusive [8]. There are several proposals but none has produced a general and practical solution for network data management. The main problems seem to be the inherent computational complexity of the problem, the lack of a well defined minimal set of feasible operations that could fulfill the requirements of the field, and the intrinsic diversity of applications (everything could be represented as a graph).

Semi-Structured Models. The need to manage and exchange large number of documents (e.g. web pages) and other data with general structure but no fixed schema motivated

the definition of semi-structured models. Their main characteristics are: schema is not mandatory, and if present, the schema is encoded along with the data it describes. A well-known example is the eXtensible Markup Language (XML).

The proposed model adapts some elements from graph data models and semi-structured data models to cover the particular requirements of social networks data.

3.5.3 Current Support for Social Networks Data Management

Today, information management and database support for network structured data are still limited. Despite the automatic and continuous collection of social network data across the Internet and the Web, which is commonplace today, there are no fully-developed and implemented social networks DBMS. Ruffin et al. [93] evaluate the suitability of five different types of database systems for storing and querying social network data. They conclude that none of these systems satisfy the requirements of social networks data management.

There are extensions to commercial DBMS that provide network manipulation features over their original data model. For instance, in biosciences (but not limited to), the Oracle database since version 10g offers support for network data structures [100], where actual data is in the relational database (as nodes edges and paths), and all analysis functionality is outside of it (as APIs and applications). Even though an implementation with a relational database backend could be a valid option, a formal data model is still missing.

A review of the publications in the database and data mining conferences, also support this conclusion. Main events reviewed were:

- DBiBD (2005): Workshop on Database Issues in Biological Databases.
- CAiSE (2003): Conference on Advanced Information Systems Engineering.
- DMKD (2000-2004): Workshop on Research Issues on Data Mining and Knowledge Discovery.
- ICDT/EDBT (1999-2011, biannual until 2009, since then collocated with EDBT): International Conference on Database Theory/ International Conference on Extending Database Technology.
- SIGKDD (2001-2011): ACM Special Interest Group on Knowledge Discovery and Data Mining.
- SIGMOD/PODS (2000-2011): ACM Special Interest Group on Management of Data / Principles Of Database Systems.
- DBSocial(2011): Workshop on Databases and Social Networks.
- VLDB (2000-2011, VLDB Endowment since 2008): Very Large Databases Conference.

From all the research presented in these events, no single work addressed the topic of a data model for social networks. However, there is a clear trend of increasing activity regarding database and data mining support for networks, like the Web and sensor networks, and in recent years for social networks.

Chapter 4

Social Networks Data Model I: Data Structure

The data structure must be able to represent all the objects in the domain at the proper level of abstraction; provide support to and not hinder the performance of the query language; and satisfy the additional requisites of scalability and portability.

Accordingly, we propose a new data structure for social networks data management. This data structure is designed to represent all possible social networks in terms of actors, relations, and attributes; and to support the appropriate query language.

In this chapter we present the definition of the data structure for the SNDM, addressing the requisites presented in the previous chapter. Section 4.1 contains a summary of the requisites for the data structure. Section 4.2 defines the data structure: a tripartite directed labeled multigraph. In Section 4.3, we present a representation of the data structure as a model of triples which is used in the remaining chapters. In section 4.4 we study the relation of the model's data structure with the *GraphLog* data structure. In the last section of the chapter we discuss related work.

4.1 Data Structure Requirements Summary

We can summarize the requisites for the data structure to represent social networks (see Section 3.4.1) as follows:

1. Actors have a unique identifier and a set of attributes, and can participate in any number of relations.
2. Relations have a unique identifier, a set of attributes, and a number of participant actors. The number of participants can be one or more, and it may change without affecting the other properties of the relation.
3. Attributes have an associated meaning and a literal value. One attribute is identified by the identifier of the object to which it is attached (actor or relation), by its meaning, and by its literal value. The class of an object –either an actor or a relation– is a special kind of attribute: its *family*.
4. Actors, relations, attributes, and their connections form a social network. Sharing and reuse requires metadata at the network level to record, for instance, provenance of the data sets.

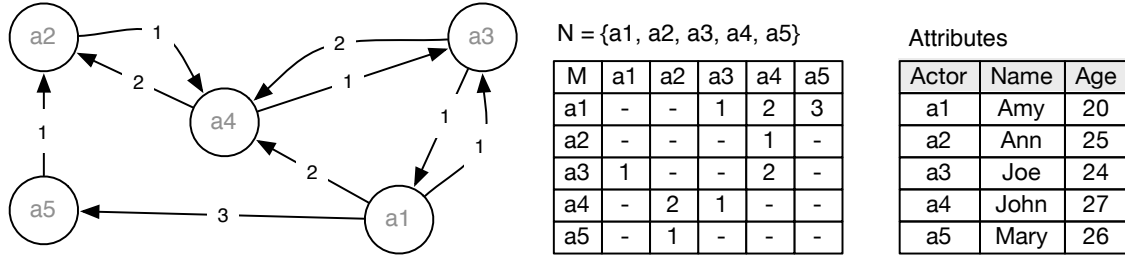


Figure 4.1: *One-mode Friendship Network (Classic Model)*. Directed labeled graph representing a network of friends. Arc labels represent how close each person rates the named friend. Note that attributes are stored in a separate structure.

4.2 Data Structure

This section deals with the data structure of the SNDM. We progressively build it starting from simple graph models. The classical SNA data modeling strategy represents social networks as graphs: nodes represent actors and edges represent social ties between pairs of actors. The attributes that describe actors are stored in separate tables. The SNDM data structure design departs from the classical strategy. Even though the structure is still a graph, the relations are represented as nodes and the attributes are part of the network structure. We also introduce in the following sections a graphical syntax used to depict social networks, and an equivalent triple based syntax used in the definition of the query language and in the implementations. The main advantage of using standard graphs as the underlying mathematical model for the data structure, is that they provide a sound and well-studied theoretical background for desired data model operations.

Graphs are the mathematical abstraction best suited for and most used to represent networks. There are other possible choices, such as hyper-edges to represent non-dyadic relations that involve three or more actors [8, 16]. In this work, to balance formalism and simplicity, we decided to follow the classical approach and to adapt as much as possible our data structure for networks to standard graphs representation.

4.2.1 Classical SNA Model

In *Social Network Analysis* (SNA) a social network is customarily represented by a directed or undirected labeled graph G [98, 110]. We refer to this approach as the *classical SNA model*.

Definition 13 (Directed Labeled Graph) *A directed labeled graph G is a sextuple*

$$(N, E, L_N, L_E, \nu, \varepsilon)$$

where: N is a finite set of nodes; $E \subseteq N \times N$ is a finite set of edges e ; L_N is a set of node labels; L_E is a set of edge labels; $\nu : N \rightarrow L_N$ is a one-to-one node labeling function; $\varepsilon : E \rightarrow L_E$ is an edge labeling function.

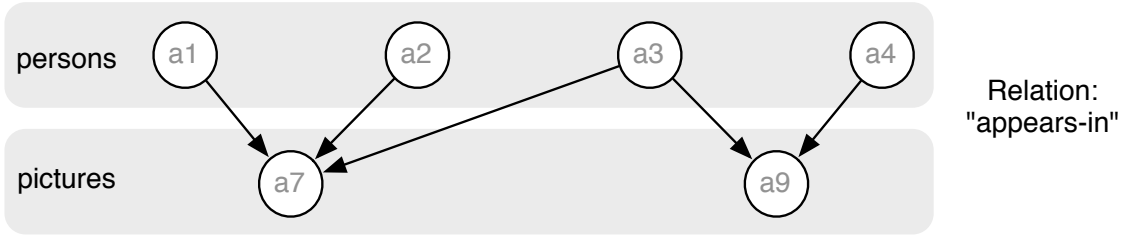


Figure 4.2: *Two-mode Persons-that-appear-in-Pictures Network (Classic Model)*. A two mode social network is a bipartite graph usually used to represent participation in groups or events. In this example all persons that appear in the same picture.

Definition 14 (One-mode Social Network) *A one-mode social network is a directed labeled graph $G = (N, E, L_N, L_E, \nu, \varepsilon)$, such that N represents a set of social actors, and E represents a set of social ties between pairs of these actors.*

In a *social network* the labels on its edges can be numbers expressing, for instance, choice preference or the strength of the relation. For example, in a *one-mode social network* representing the friendship ties among a group of persons, a numeric label on edges could represent the order of choice, for instance: the first choice (label “1”) is the closest friend (see Figure 4.1).

Definition 15 (k-partite Graph) *A graph G is called k-partite, with integer $k \geq 2$, if N admits a partition in k classes such that every edge has its ends in different classes (vertices in the same partition class must not be adjacent).*

When $k = 2$ the graph is called *bipartite*.

In a *one-mode social network* all actors are of the same kind. If there are two kinds of actors and the network is bipartite, it is a *two-mode social network*.

Definition 16 (Two-mode Social Network) *A two-mode social network is a directed labeled bipartite graph $G = (N, E, L_N, L_E, \nu, \varepsilon)$, such that $N = A \cup B, A \cap B = \emptyset$, where A and B represent two disjoint sets of social actors, and $E \subseteq A \times B \cup B \times A$ represents a set of social ties between pairs of actors of different kind.*

In a *two-mode social network* there are two disjoint sets of actors, and social ties exist only between members of different sets. This type of social networks are also called *affiliation networks* because they usually represent the co-participation of actors in groups or events (see Figure 4.2).

It is possible to produce two one-mode undirected networks from each two-mode network: two actors of the same set are connected if there is a transitive relation that includes one intermediary actor from the other set (see Section 2.1.2). For example, from the two-mode social network *Persons-that-appear-in-Pictures* in Figure 4.2, it is possible to obtain the two one-mode networks depicted in Figure 4.3: *Persons that appear in the same picture*, and *Pictures that contain at least one person in common*.

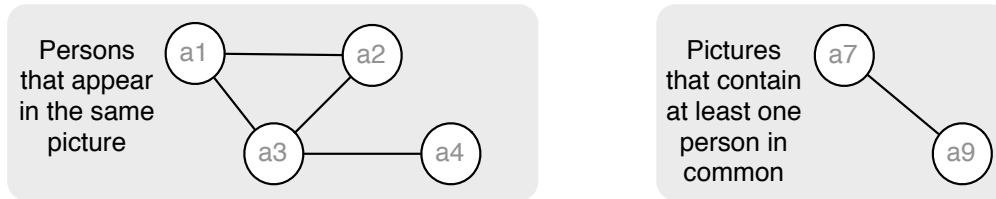


Figure 4.3: *Two one-mode networks from Persons-that-appear-in-Pictures Network in Figure 4.2 (Classic Model).* Transforming the paths of length two in edges, it is possible to obtain two one-mode social networks from one two-mode social network.

In this *classical model*, the attributes –that may exist to describe the actors and relations– are stored in separate tables, not as part of the graph itself (see Figure 4.1).

4.2.2 SNDM Data Structure

The *classical SNA model* has severe drawbacks to represent all data available today to describe social networks and their dynamics. Its appeal is mostly pragmatic, as it reflects historical limitations present in SNA practice: expensive and laborious data collection processes, lack of specialized tools for automatic data managing (storage and manipulation), and analytical models and tools bounded by available computing power.

Many of these limitations were already identified by Freeman [51] in the early 1980’s. He formalized an extended set of requirements, the *maximal social structure experiment* (see Section 3.1.1), which basically defines the need of support for: several types of actors and relations, attributes to describe these actors and relations, and the change of all these elements through time. More recently, Bonacich et al. [16] further extended these requirements with n-ary relations. Relations involving more than two actors arise in several scenarios. For instance, in a trade relation we may need to represent, along with the seller and buyer, other actors such as a broker or the product itself.

In order to overcome these limitations and give the necessary flexibility for its management, we propose a new data structure for social networks data management. We build this new data structure by progressively refining the *classical SNA model* until coping with the current requirements for data management of social networks.

First, to allow the existence of multiples edges representing different kinds of social ties, we use as the underlying structure a multigraph instead of a graph. Notice that in the following definition E is no longer required to be a subset of $N \times N$.

Definition 17 (Directed Labeled Multigraph [30]) *A directed labeled multigraph G is a directed labeled graph (see Definition 13) where there may exist several arcs between the same pair of nodes, this is achieved by adding an incidence function ι , thus it is a septuple*

$$(N, E, L_N, L_E, \iota, \nu, \varepsilon)$$

where: N is a finite set of nodes; E is a finite set of edges; L_N is a set of node labels; L_E is a set of edge labels; $\iota : E \rightarrow N \times N$ is the incidence function that associates with each

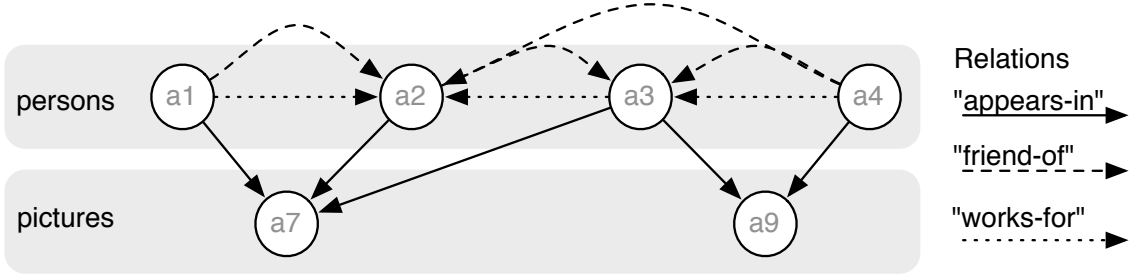


Figure 4.4: *Multi-modal Multi-relational Social Network*. In this variety of social network there exist actors of several families (persons and pictures) and different kinds of relations (appears-in, friend-of, and works-for). Different kinds of relations are represented by arrows with different line styles).

edge in E a pair of nodes from N ; $\nu : N \rightarrow L_N$ is the node labeling function; $\varepsilon : E \rightarrow L_E$ is the edge labeling function.

Figure 4.4 depicts a social network with multiple kinds of actors and relations, which is called multi-modal and multi-relational (see for instance Singh [99]). We define them in terms of a *directed labeled multigraph* as follows:

Definition 18 (Multi-modal multi-relational Social Network) *A multi-modal multi-relational (mm-mr) social network is a directed labeled multigraph (see Definition 17)*

$$G = (N, E, L_N, L_E, \iota, \nu, \varepsilon)$$

where N is a set of actors that is covered by a collection of k families of actors or modes; and E is a set of social ties that admits a partition in j families of relations.

In order to include attributes on actors we partition the set of nodes in two different types: an *actor* set and an *attribute* set. In this way attributes are represented in the graph structure itself and they can be naturally queried with the same status as the structure of relations.

Definition 19 (MM-MR Social Network with Attributes on Actors) *A multi-modal multi-relational (mm-mr) social network with attributes on actors is a mm-mr social network (see Definition 18). Formally*

$$G = (N, E, L_N, L_E, \iota, \nu, \varepsilon)$$

where the set of node N admits a partition in actors and attributes, such that $N = A \cup C$ is the disjoint union of the actor set A with the attribute set C ; additionally the actor set A is covered by a collection of k families of actors or modes; accordingly E admits a partition in relations and attribute meanings, such that $E = E_{AA} \cup E_{AC}$ is the disjoint union of the set of arcs between actors E_{AA} (relations) with the set of arcs between attributes and actors E_{AC} (meanings); the set of relations E_{AA} is partitioned into different families of relations; $L_N = L_{N_A} \cup L_{N_C}$ is the disjoint union of the actor label set L_{N_A} with the attribute label set L_{N_C} ; the remaining elements are adapted as needed.

A *mm-mr social network* with attributes on actors is a network where each actor may be

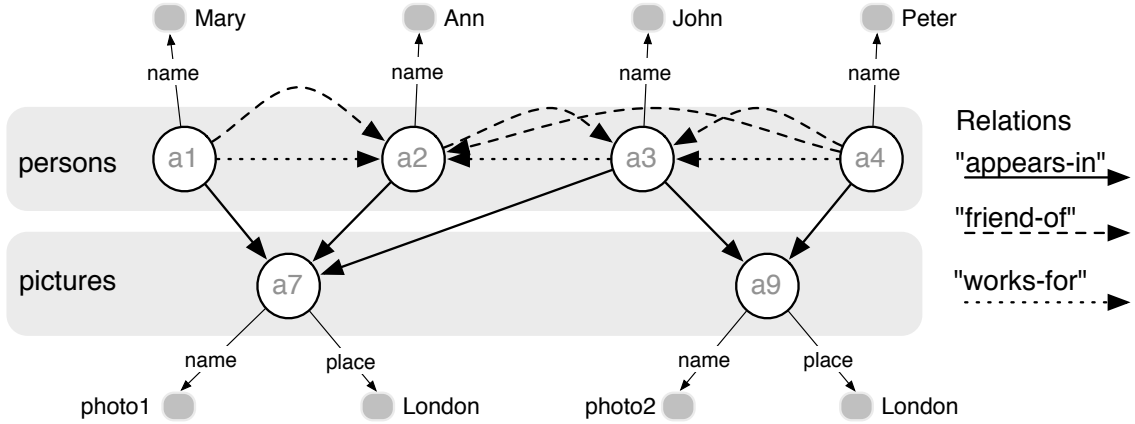


Figure 4.5: *Multi-modal Multi-relational Social Network with Attributes on Actors*. This is a mm-mr social network where each actor may additionally have an arbitrary number of attributes describing its features.

adjacent to a set of attribute nodes describing its features (see Figure 4.5).

Note that this new structure still does not support attributes on relations, nor n-ary relations. We add support for these requirements under two design restrictions. First, to include all information on the same data structure (allowing the definition of a pattern matching query language uniformly encompassing all data elements, including structure and attributes), and second, to keep the underlying model a graph (to use their well known properties).

To cope with these pending requirements under the stated restrictions, we define a *generalized social network*. In this kind of network, social ties (relations) are no longer modeled as edges, instead they are modeled as nodes.

Definition 20 (Generalized Social Network) *A generalized social network is a mm-mr attributed social network (see Definition 19) with social relations modeled as nodes, where:*

- *the set of nodes $N = A \cup T \cup C$ is a disjoint union of the actor set A , with the relation set T (social ties), and the attribute set C ;*
- *the collection of families of actors \mathcal{A} covers the actor set A ;*
- *the collection of families of relations \mathcal{T} covers the relation set T ;*
- *both actors and relations may be described using attributes;*
- *the network is tripartite regarding the partition of the node set N in actors, relations, and attributes;*
- *all arcs between the same pair a, r with $a \in A$ and $r \in T$ must have different labels;*
- *all the other elements are adapted accordingly.*

In a *generalized social network*, besides the standard representation capabilities, it is possible to connect a variable number of participating actors to each relation, and to describe

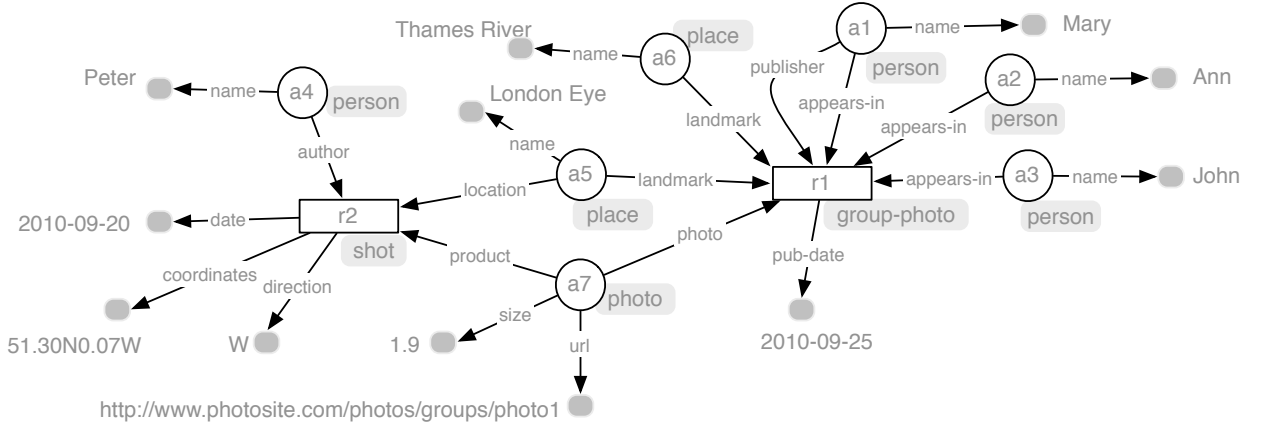


Figure 4.6: *Generalized Social Network*. A *generalized social network* is a tripartite labeled directed multigraph, where the node set N is partitioned into actors (round nodes), relations (square nodes), and attributes (small gray round nodes). Families are represented by gray labels attached to actors and relations.

these relations by attaching attribute nodes to them. See for instance the representation of a group photograph in Figure 4.6.

To further clarify the definition of a *generalized social network* we include its disaggregated formal details in the following extended version of the Definition 20.

Definition 20 bis (Generalized Social Network - Formal Details) A *generalized social network formally defined as a tripartite directed labeled multigraph (see Definition 17) plus a family labeling function f and a family label set L_f ,*

$$G = (N, E, L_N, L_E, L_f, \iota, \nu, \varepsilon, f)$$

where:

- The set of nodes $N = A \cup T \cup C$ is a disjoint union of the actor set A , with the relation set T , and the attributes set C .
- There exists a finite collection of families (subsets) of actors $\mathcal{A} = \{A_1, A_2, \dots, A_k\}$ such that every $A_i \subseteq A$ and $\cup_{1 \leq i \leq k} A_i = A$.
- There exists a finite collection of families (subsets) of relations $\mathcal{T} = \{T_1, T_2, \dots, T_j\}$ such that every $T_i \subseteq T$ and $\cup_{1 \leq i \leq j} T_i = T$.
- The set of arcs $E = E_{AT} \cup E_{AC} \cup E_{TC}$ is a disjoint union of the set of arcs between actors and relations E_{AT} , with the set of arcs between attributes and actors E_{AC} , and the set of arcs between attributes and relations E_{TC} .
- The set of node labels $L_N = L_A \cup L_T \cup L_C$ is a disjoint union of the set of actor labels L_A , with the relation labels set L_T , and the attribute labels set L_C .
- The set of arc labels $L_E = L_{AT} \cup L_{AC} \cup L_{TC}$ disjoint union of the set of labels of arcs between actors and relations L_{AT} (participation roles), with the set of labels of arcs between attributes and actors L_{AC} (actor attributes meanings), and the set of labels of arcs between attributes and relations L_{TC} (relation attributes meanings).

- The set of family labels $L_f = L_{f_A} \cup L_{f_T}$ is a disjoint union of the set of actor family labels L_{f_A} , and the set of relation family labels L_{f_T} .
- $\iota = \{\iota_{AT}, \iota_{AC}, \iota_{TC}\}$ is a set of incidence functions such that $\iota_{AT} : E_{AT} \rightarrow A \times T$ is an incidence function that associates each participation edge to an actor and a relation; $\iota_{AC} : E_{AC} \rightarrow A \times C$ is an incidence function that associates a meaning edge to an actor and an attribute; and $\iota_{TC} : E_{TC} \rightarrow T \times C$ is an incidence function that associates a meaning edge to a relation and an attribute.
- $\nu = \{\nu_A, \nu_T, \nu_C\}$ is a set of node labeling functions such that $\nu_A : A \rightarrow L_A$ is a bijective function from actors to actors' labels; $\nu_T : T \rightarrow L_T$ is a bijective function from relations to relations' labels; and $\nu_C : C \rightarrow L_C$ is a function from attributes to attributes labels.
- $\varepsilon = \{\varepsilon_{AT}, \varepsilon_{AC}, \varepsilon_{TC}\}$ is a set of edge labeling functions such that $\varepsilon_{AT} : E_{AT} \rightarrow L_{AT}$ is a function from participation edges to participation edges' labels; and $\varepsilon_{AC} : E_{AC} \rightarrow L_{AC}$ and $\varepsilon_{TC} : E_{TC} \rightarrow L_{TC}$ are functions from meaning edges to meaning edges' labels.
- $f = \{f_A, f_T\}$ is a set of family labeling functions such that $f_A : \mathcal{A} \rightarrow L_{f_A}$ is a function from actors families to actors families' labels; and $f_T : \mathcal{T} \rightarrow L_{f_T}$ is a function from relations families to relation families' labels.
- The following condition holds for all edges between the same pair of actors and relations, for all e_1 and e_2 such that $\iota(e_1) = \iota(e_2) = (u, v)$ with $u \in A$ and $v \in T$, $e_1, e_2 \in E \Leftrightarrow \varepsilon(e_1) \neq \varepsilon(e_2)$.
- Each labeling function in ν , ε , and f , except ν_c , must be invertible.
- For a relation $r \in T$ between two actors $a_1, a_2 \in A$, such that there exist $e_1, e_2 \in E$, and $\iota(e_1) = (a_1, r)$, $\iota(e_2) = (a_2, r)$ with labels $\varepsilon(e_1) = p_1$, $\varepsilon(e_2) = p_2$. The direction of r can be specified by the ordered pair of participation labels, that is a direction (p_1, p_2) indicates that r starts in a_1 and ends in a_2 , the opposite direction is represented by (p_2, p_1)

Definition 21 (SNDM Data Structure) *The data structure of SNDM is a generalized social network.*

In the following sections we refer to *generalized social networks* interchangeably as SNDM social networks, or simply as social networks.

4.2.3 Graphical Syntax

We define for the SNDM data structure a graphical syntax to depict generalized social networks (see Figure 4.7). This syntax has four building blocks: actors with its family labels, relations with its family labels, participation roles of actors in relations, and attributes on actors and relations.

Actors are represented by circles with their id inside, and a grey label indicating their family. Accordingly, relations are represented by rectangles with their id inside, and a grey label indicating their family. Participation roles are represented by directed lines departing from the participating actor to the relation. Line labels differentiate participation roles. Attributes, represented as gray dots, are linked by arcs to the actor or relation they describe,

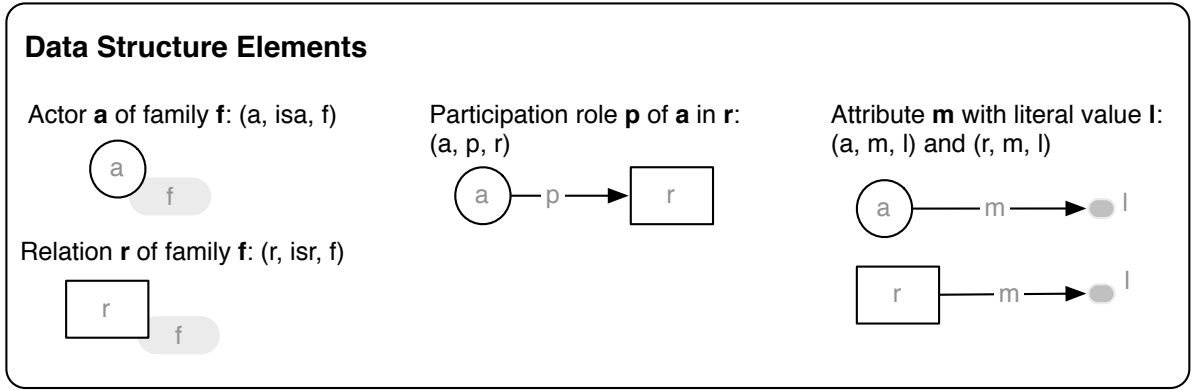


Figure 4.7: *Social Network Graphical Syntax*. From left to right, the four graphical building blocks of a social network: an actor (above) and its family label, a relation and its family label, a participation role of an actor in a relation, and attributes on an actor and a relation. For each block the equivalent triple representation is also shown.

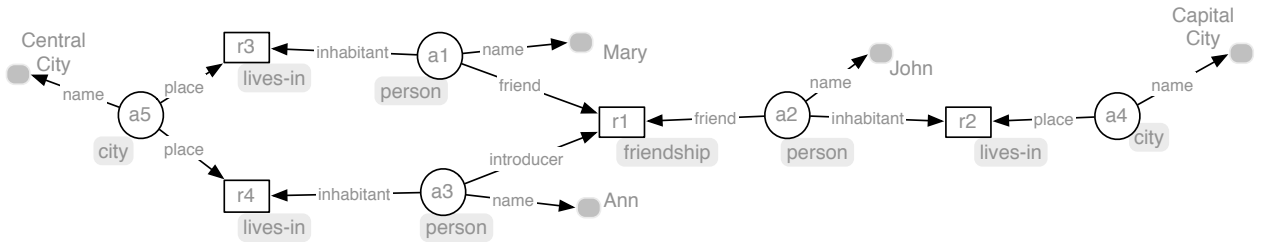


Figure 4.8: *Friendship Network*. A social network representing the friendship relation (square node) between *Mary* and *John*, who were introduced by *Ann* (actors as round nodes, and attribute values as grey dots). The cities of residence are also represented as actors.

and labeled with their literal values. Arc labels represent the meaning of the attribute.

Figure 4.8 depicts a small social network with two types of actors: cities and persons, both with attribute names. There are also two types of relations: lives-in linking persons and cities, and friendship linking persons. For instance, it could be read “*Mary* lives in *Central City*”, and “*Mary* and *John* participate as friends in *r1*, with *Ann* as ‘introducer’ ”.

4.3 Triples Based Syntax

There is a simple triple notation for the SNDM data structure (see Definition 21). We will prefer this triple notation for two reasons. First, it is better suited to the definition and study of the query and transformation language because each triple can be seen as a *Datalog* predicate. Second, triples provide a better framework for implementation than the theoretical graph definition. For instance, sets of triples can be directly mapped to three column relational tables, or an RDF model.

Definition 22 (Social Network Triple Representation) *Consider a generalized social network $G = (N, E, L_N, L_E, L_f, \iota, \nu, \varepsilon, f)$ (see Definition 20); and consider the vocabulary $\Sigma = L_A \cup$*

$L_T \cup L_C \cup L_{f_A} \cup L_{f_T} \cup \{isa, isr\} \cup L_{AT} \cup L_M$, where $L_M = L_{AC} \cup L_{TC}$ is the set of label of attributes (meanings). Then define the following sets of triples:

1. *Nodes and their Family Belonging:*

$$N \subseteq (L_A \times \{isa\} \times L_{f_A}) \cup (L_T \times \{isr\} \times L_{f_T})$$

2. *Participation roles of actors in relations:*

$$R \subseteq L_A \times L_{AT} \times L_T$$

3. *Meanings (attributes):*

$$M \subseteq (L_A \cup L_T) \times L_M \times L_C$$

And the referential integrity constraint: all actors and relations in R and M must have at least one family associated in N .

$$\pi_1(R) \subseteq \pi_1(N)$$

$$\pi_3(R) \subseteq \pi_1(N)$$

$$\pi_1(M) \subseteq \pi_1(N)$$

From the definitions above we can show that:

Lemma 1 *A social network $G = (N, E, L_N, L_E, L_f, \iota, \nu, \varepsilon, f)$ can be represented by three sets of triples (N, R, M) as described above, and vice versa.*

Proof.

Graph representation to triple representation. It is possible to translate a given social network G into a triple representation (N, R, M) with the following algorithm that produces a triple for each edge in G :

- For each actor $a \in A$, and for each family of actors A_i such that $a \in A_i$ add the triple $(\nu_A(a), isa, f_A(A_i))$ to N .
- For each relation $t \in T$, and for each family of relations T_i such that $t \in T_i$ add the triple $(\nu_T(t), isr, f_T(T_i))$ to N .
- For each edge $e \in E_{AT}$ with end nodes $u \in A$ and $v \in T$, add the triple $(\nu_A(u), \varepsilon_{AT}(e), \nu_T(v))$ to R .
- For each edge $e \in E_{AC}$ with end nodes $u \in A$ and $v \in C$, add the triple $(\nu_A(u), \varepsilon_{AC}(e), \nu_C(v))$ to M .
- For each edge $e \in E_{TC}$ with end nodes $u \in T$ and $v \in C$, add the triple $(\nu_T(u), \varepsilon_{TC}(e), \nu_C(v))$ to M .

Triple representation to graph representation. It is possible to translate a given social network represented as three sets of triples (N, R, M) in a generalized social network G with the following algorithm that produces a node or edge for each triple:

- For each triple $(\nu_A(a), isa, f_A(A_i))$ in N add an actor node a to A_i .

Table 4.1: Friendship network (Fig. 4.8) represented as sets of triples .

N: Typing			R: Roles			M: Attributes		
<i>a1</i>	isa	‘person’	<i>a1</i>	friend	<i>r1</i>	<i>a1</i>	name	‘Mary’
<i>a2</i>	isa	‘person’	<i>a2</i>	friend	<i>r1</i>	<i>a2</i>	name	‘John’
<i>a3</i>	isa	‘person’	<i>a3</i>	introducer	<i>r1</i>	<i>a3</i>	name	‘Ann’
<i>a4</i>	isa	‘city’	<i>a2</i>	inhabitant	<i>r2</i>	<i>a4</i>	name	‘Capital City’
<i>a5</i>	isa	‘city’	<i>a4</i>	place	<i>r2</i>			
<i>r1</i>	isr	‘friendship’	<i>a1</i>	inhabitant	<i>r3</i>	<i>a5</i>	name	‘Central City’
<i>r2</i>	isr	‘lives-in’	<i>a5</i>	place	<i>r3</i>			
<i>r3</i>	isr	‘lives-in’	<i>a3</i>	inhabitant	<i>r4</i>			
<i>r4</i>	isr	‘lives-in’	<i>a5</i>	place	<i>r4</i>			

- For each triple $(\nu_T(t), isr, f_T(T_i))$ in T
add a relation node t to T_i .
- For each triple $(\nu_A(u), \varepsilon_{AT}(e), \nu_T(v))$ in R
add a participation role edge e to E_{AT} with end nodes $u \in A$ and $v \in T$.
- For each triple $(\nu_A(u), \varepsilon_{AC}(e), \nu_C(v))$ in M
add an attribute node v to C , and a meaning edge e to E_{AC} with end nodes $u \in A$ and $v \in C$.
- For each triple $(\nu_T(u), \varepsilon_{TC}(e), \nu_C(v))$ in M
add an attribute node v to C , and a meaning edge e to E_{TC} with end nodes $u \in T$ and $v \in C$.

□

Table 4.1 shows the triple representation of friendship network depicted in Figure 4.8 (see equivalence between graphical syntax and triples in Figure 4.7).

4.4 GraphLog Based Syntax

GraphLog is a query language based on a graph representation of both databases and queries. *SNQL* is a query language for social networks that is also based on graphs patterns. A part of the *SNQL* queries – the extraction pattern *EP*– is directly based on *GraphLog*. The semantics of both languages are defined in terms of translations to *Datalog*: in both cases each query has an equivalent *Datalog* program. In this section we focus on the underlying data structures used by both languages.

Consens and Mendelzon [30] define both data instances and queries in *GraphLog* as directed labeled multigraphs, whose node labels are sequences of variables, and its edge labels are EDB literals. The base data structure of SNDM (see Definition 20) is also a directed labeled multigraph, to which we have added restrictions in the process of modeling the requirements of social networks (it is a tripartite graph, and the labels of nodes and arcs are single symbols). Thus, the SNDM data structure is actually a subset of the *GraphLog* data structure.

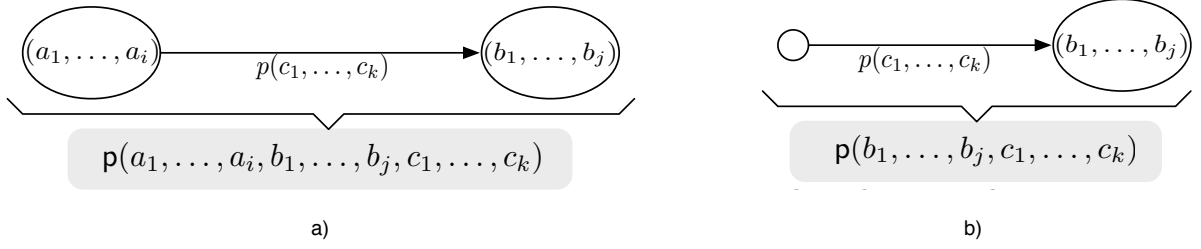


Figure 4.9: *Edge semantics in GraphLog*. The meaning of an edge in *GraphLog* is defined by its translation to a *Datalog* predicate.

The semantics of a graph in *GraphLog* is defined by a *logical translation* of each edge in the graph, including its two end nodes, to *Datalog*. In general, a tuple

$$p(a_1, \dots, a_i, b_1, \dots, b_j, c_1, \dots, c_k)$$

can be represented by an edge labeled with the literal $p(c_1, \dots, c_k)$ between two nodes labeled as (a_1, \dots, a_i) and (b_1, \dots, b_j) ; see Figure 4.9. The sequence of symbols in the source node, and the sequence of arguments of p , could be empty.

$$p(b_1, \dots, b_j, c_1, \dots, c_k)$$

The algorithm to produce the triple representation of the SNDM data structure can be slightly modified to show the equivalence of semantics for the subset covered by SNDM: instead of a triple an EDB predicate is produced in each case.

Lemma 2 (SNDM Data Structure to *GraphLog* EDB translation) *It is possible to translate a given social network G into a GraphLog EDB.*

Proof. It is sufficient to show the following algorithm that produces an EDB predicate for each edge in G :

- For each actor $a \in A$, and for each family of actors A_i such that $a \in A_i$, add the predicate $n(\nu_A(a), isa, f_A(A_i))$ to EDB.
- For each relation $t \in T$, and for each family of relations T_i such that $t \in T_i$, add the predicate $n(\nu_T(t), isr, f_T(T_i))$ to EDB.
- For each edge $e \in E_{AT}$ with end nodes $u \in A$ and $v \in T$, add the predicate $r(\nu_A(u), \varepsilon_{AT}(e), \nu_T(v))$ to EDB.
- For each edge $e \in E_{AC}$ with end nodes $u \in A$ and $v \in C$, add the predicate $m(\nu_A(u), \varepsilon_{AC}(e), \nu_C(v))$ to EDB.
- For each edge $e \in E_{TC}$ with end nodes $u \in T$ and $v \in C$, add the predicate $m(\nu_T(u), \varepsilon_{TC}(e), \nu_C(v))$ to EDB.

□

This algorithm is analogous to the *logical Translation* provided by Consens and Mendelzon [30] for *GraphLog*. The main differences are that –in the case of SNDM– the source graph is tripartite, and that node and edge labels contain only one id or value.

It is possible to define an inverse translation algorithm for any *GraphLog* graph instance that is also a generalized social network: it is tripartite, all its nodes and edges are labeled with single symbols, and provides family predicates.

4.5 Related Work

Given the triple representation of SNDM social networks it is natural to consider representing them in the Resource Description Framework¹ (RDF). RDF is a more general triple model, thus it is possible to represent SNDM social networks as follows:

- URIs are used to represent node (actors and relations) identifiers.
- RDF literals are used to represent labels and values.
- Blank nodes are not allowed.
- Network must be tripartite (actors, relations, and attributes), and sets of labels must be partitioned following Definition 20.
- Integrity constraints in Definition 22 must be satisfied.

We discuss other related data models and frameworks after presenting the social networks query language, in the next chapter.

¹<http://www.w3.org/RDF/>

Chapter 5

Social Networks Data Model II: Query and Transformation Language

In a data model, the query and transformation language is the set of data-manipulation operations designed to access and transform the information in a database defined under that model (that is, using the corresponding data structure) [43]. The expressiveness of the query language is the set of all queries that can be expressed in the language [2]. A practical query language must offer an appropriate trade-off between its expressiveness and a feasible query-evaluation complexity in order to solve the actual data management problem while scaling adequately. Additionally, to actually improve users productivity, the language should be understandable by and appealing to the users in the application domain.

Consequently, the query and transformation language of the SNDM must be able to query and transform social networks following the requirements from the social networks community of practice, while using elements and mechanisms familiar or accessible to the members of this community.

In the social networks domain, a pattern matching and production strategy fulfills these requirements. Users could easily correlate the graphical representation of graph patterns with social networks; a user could specify a basic query by drawing a pair of patterns: a motif to be searched, and a new pattern to be constructed each time the motif is found. We found that this basic strategy covers most of the basic data management needs in SNA [95, 96]. Abiteboul et al. [2] shows that, if negation is allowed in the patterns, this strategy is equivalent to relational algebra without aggregation, and consequently equivalent to an affordable subset of *Datalog*. However, there are query language requirements that are not expressible in terms of local patterns, and which may also require to produce summaries and new values, e.g. to count all transitively reachable friends of a given actor. To improve the expressiveness of the query language, while keeping its complexity under a practical bound (NLOGSPACE), we extended this basic approach with three elements: complex patterns (using boolean combination of patterns); functionally produced new values and object ids (using aggregate functions and second order tuple-generating dependencies); and inflationary patterns like the subnetwork reachable from a given actor (using transitive closure). This design decision gives a language that covers all practical SN requirements, leaving out only

very intricate cases, like maximal and recursive cohesive subgroups (e.g. k-clans and k-clubs [110]), whose identification algorithms push the complexity over the complexity bound stated above [19, 31]. The resulting Social Networks Query and Transformation Language, SNQL, has a textual and graphical syntax, and covers all the requirements fulfilled by queries computable under the given complexity bound.

In this chapter we present the definition of SNQL and study its properties. First, we briefly recall the main requirements for a query language for social networks (see Chapter 3). Then, an overview of the language and its properties is presented. We continue with the formal definition of the query and transformation language, its evaluation process, and the study of its expressiveness and complexity. Finally, we briefly discuss related work on query languages for social networks and other network structured data.

5.1 Query Language Requirements Summary

Three driving ideas summarize the design goals of SNQL: appropriate expressiveness to cover the required queries; feasible complexity to allow the actual implementation of the language; and the adequate level of abstraction to produce schemas that are easily correlated to the problem domain.

The following list enumerates the key requirements of SNQL:

- to identify the occurrences of basic network patterns: small fixed patterns that characterize a locality in terms of its structure (participants and organization of ties) and attributes values;
- to identify complex patterns defined as logical combination of simpler patterns using the logical operators: AND, OR, and AND-NOT;
- to produce aggregate measures from groups of objects and their attributes;
- to identify the objects that can be reached through the transitive closure of a given pattern;
- to produce new social networks transforming existing social networks, that is: to define new structures, and to calculate new values and object identifiers, using the information collected from source social networks;
- to combine and compare networks; if networks are represented as sets, these operations may correspond to set theoretical union and difference;
- to identify combinatorial paths, general path summarization, and general cohesive subgroups; and
- to define complex queries composing simpler queries (the query language should be closed over the set of social network instances).

5.2 Language Overview

What is a “good” query language for the requirements given above? Although nobody has yet proposed a set of primitives, flexible and expressive enough, to represent the full diversity of queries implied by these requirements [8], the good news for SN is that the required set of functionalities seems to be small, and most of them (not all) are computable by efficient graph algorithms (paths, connectedness, etc.).

From a social networks practice point of view, we distinguish two types of operations: *data management* operations (i.e. queries and transformations) that return social networks, and *structural measures* operations that return values or sets of values for structural properties, such as centrality, diameter, etc. SNQL concentrates on the first type: data management operations that produce networks from networks, and allows the composition of queries. As for the second group, there are at least two ways to implement the availability of structural measures from the query language: via import/export facilities in the DBMS from/to structural analysis tools, for instance well known SNA tools like *Pajek*, *network* and *sna R* packages, and *UCINET*; or by implementing an extension to the DBMS, a set of structural analysis functions, and making them available to the query language, without the need of expressing them in the language itself. The Social Networks Data Management System (SNDBMS) proof-of-concept prototype, discussed later (see Chapter 6), uses the import/export approach.

SNQL queries build instances of a target schema T from instances of a source schema S ; both schemas must describe SNDM social networks.

The evaluation of a query can be seen as a two step process: first the relevant data is identified in the source instance; then the resulting network is built using the identified data. The following issues regarding complexity and practical computational costs are worth noticing:

- The actual evaluation process does not need to proceed in two separate steps collecting first all relevant data in an intermediate structure and then producing the result. If a locality principle holds, it will be possible to produce a portion of the result from each selected portion of the source, thus avoiding the materialization of the intermediate data.
- Transforming data from one schema to another one has been extensively studied in databases, an area known as “data exchange”. In this context an appropriate formalism is *second-order tuple-generating dependencies (SO tgds)*, a class of second-order formulas with function symbols. Fagin et al. [45] showed that they can be considered the “right” language for composing schema mappings, and that the chase procedure can be extended to *SO tgds* so that it produces polynomial-time computable universal solutions in data exchange settings specified by *SO tgds*.
- The complexity of these queries also depends on the nature of the selected and produced structures. Consens and Mendelzon [31] show that extending *GraphLog* with operations to select and report whole paths, and to compute general path summarizations and aggregation, increases the complexity to exponential time.

Consequently, to achieve a practical complexity bound, we need to narrow the set of queries in SNQL, leaving outside the language queries defined in terms of whole paths or paths summarization, such as the selection of maximal cohesive subgroups defined in terms of minimum mutual distances, e.g. k-clubs [110].

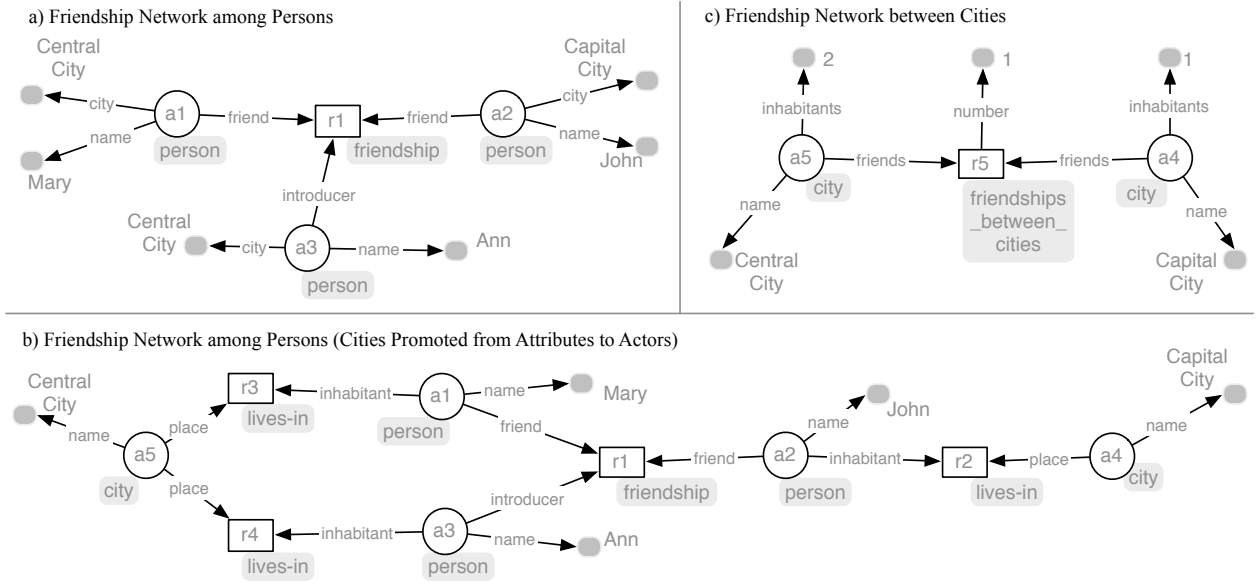


Figure 5.1: *Friendship Network and Simple Queries Results.* a) A social network representing the friendship relation (square node) between *Mary* and *John*, who were introduced by *Ann* (actors as round nodes, and attribute values as gray dots). b) The same social network after promoting *city* attributes to actors. c) The social network result after grouping persons by city and computing aggregate attributes: *inhabitants* of each city, and *number* of friendships between cities.

Example 3 introduces a friendship network along with two other networks that are defined by SNQL queries over the first. The actual queries are presented in detail later, after defining the syntax of the language.

Example 3 Consider a SN of friendship relations among people, including some other person, if any, who introduced them; this implies having relations of variable arity (solved by representing relations as nodes, see Chapter 4). People are described by the attributes ‘name’ and ‘city’ (see Fig. 5.1a). To study the relevance of city of residence to friendship might require promoting cities to actors and linking people and cities with a new type of relation, e.g. ‘lives-in’ (see Fig. 5.1b). Another type of transformation would be to group people by city of residence, thus defining a network of cities, where relations summarize friendships among residents of cities. Additionally, one might like to describe in the network the population (person count) of each city, and label the relations between them with the number of friendship-relations between people (see Fig. 5.1c).

5.3 Query and Transformation Language: Syntax

The transformation and query language should be friendly enough for both the lay-user and the programmer. For the first, a visual language close to the SN graphical representation is ideal: in the simpler cases, one extraction pattern and one construction pattern cover many use cases; in the general case, the extraction pattern should resemble the DAG of query graphs that exists in *GraphLog* [30]. For the programmer, an SQL-like language is familiar to developers and advanced database users (for searching text, writing, pasting, debugging, etc.). Thus, our language has both syntaxes.

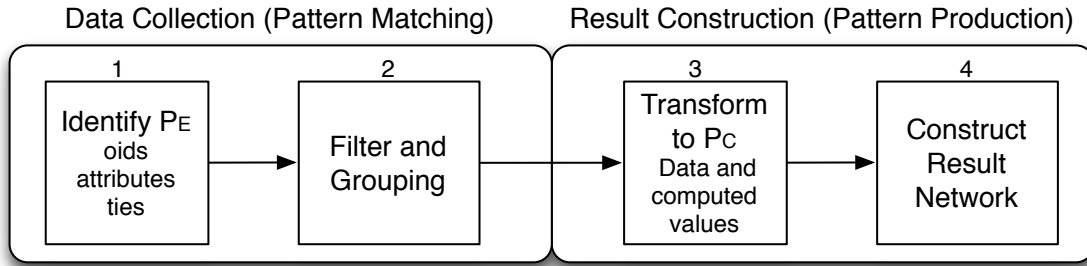


Figure 5.2: *Querying SNDM data.* The main components of a query are: a data extraction pattern P_E , and a construction pattern P_C . Each query proceeds as shown in the figure: (1) P_E is matched against the source network, (2) only the subnetworks that match are extracted, and grouped if needed for aggregation, (3) values bound to elements of P_E are used to populate elements of P_C and, possibly, to compute new values, and (4) finally the result network is constructed as the union of all instances of P_C .

5.3.1 Query Language Elements

Following the requirements and general issues summarized above we propose a query language composed by a single query operation that collects data from the source network via pattern matching, and that produces the result using the collected data and a pattern template. Queries can be composed to form all the needed variants.

Thus each query in the language is defined by two pattern elements (see Figure 5.2): an *extraction pattern* and a *construction pattern*. These two elements are based on a *basic pattern*.

Definition 23 (Basic Pattern) *A basic pattern is a connected SNDM social network (see Definitions 21 and 22), typically small, where the set of labels $L = L_N \cup L_E \cup L_f$ is extended to include the set of variables V .*

Note that in the textual syntax defined below we use the representation of social networks based on triples (Definition 22).

Definition 24 (Extraction Pattern) *An extraction pattern P_E is a single basic pattern, or an algebraic expression over basic patterns using logical operations between patterns (AND, OR, and AND-NOT), filtering, projection, aggregating functions (AGG), and transitive closure (TC).*

Constants restrict the matching of P_E , and variables are bound by a match to the corresponding values in the source network. Each match of P_E produces a tuple of variable bindings.

Definition 25 (Construction Pattern) *A construction pattern P_C is a basic pattern, plus possibly a list of equalities involving P_C variables and predefined functions. A P_C is a template designed to produce the building blocks (subnetworks) of the query result; equalities constraint the possible outputs.*

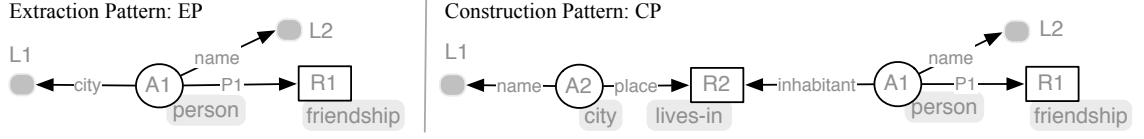


Figure 5.3: *Graphical representation of a simple SNQL query.* This is the graphical representation of P_E and P_C for the query that promotes city attributes to actors producing the network depicted in Figure 5.1b from the network depicted in Figure 5.1a. The correspondence between variables from P_E and P_C is indicated using the same names in both patterns. $A2$ and $R2$ represent new object identifiers created with functions: $A2 = g(L1)$ and $R2 = f(A1, A2)$.

Each subnetwork instance is produced using one tuple from the data collection process. P_C variables get their values directly from values bound to variables of P_E , or from functions that compute values or object identifiers using the values bound to variables in P_E and constants. Note that the set of variables used in P_C may include, beside the variables in P_E , new variables.

Definition 26 (SNQL Query) *An SNQL Query is a mapping $Q_{\langle P_E, P_C \rangle} : \mathcal{SN} \rightarrow \mathcal{SN}$ from the set of social networks into the set of social networks. We denote the evaluation of the query over the network instance D as $\llbracket Q_{\langle P_E, P_C \rangle} \rrbracket_D = D'$, where $D' \in \mathcal{SN}$ is the result of the query.*

The formal semantics of Q is defined by the *Datalog* program formed by the translations to *Datalog* of P_E and P_C (see Section 5.4).

Example 4 *Consider the SN of friendship relations among people presented in Example 3, and the query (graphically represented in Figure 5.3) to promote the city attribute of each person to a new type of actor. First, the P_E is matched against the friendship network: the structure and constants (attributes meanings, and family labels) must coincide with a subnetwork. Each match generates a tuple of values for variables $A1$, $R1$, $P1$, $L1$, and $L2$ (the corresponding values in the subnetwork). In turn each of these tuples is used to produce an instance defined by the P_C : each value is directly used where the same variable name is indicated. New values are functionally created for new variables: $A2 = g(L1)$ and $R2 = f(A1, A2)$. (These assignments are usually expressed in the text of the query, which is discussed below).*

5.3.2 SNQL: Use Cases and their Queries

To illustrate how the query language covers the requirements and use cases discussed in Chapter 3, we present in Table 5.1 a variation of the use cases and features table (see Table 3.2) where the column of use case description has been replaced by a column containing sketches of the SNQL queries that solve the respective use cases. In fact, two of the query sketches correspond to examples used in this chapter: 1. Promoting Attributes to Actors, and 4. Counting Binary Relations (see Examples 4 and 6, respectively).

Table 5.1: Selected social network data management use cases, their requirements, and the corresponding query sketches over source network S . See use cases descriptions in Table 3.2.

Use Case	Query Sketch	Required QL Features
1. Selecting Groups	Extraction: select all subnetwork instances from S that match P_E and satisfy filters. Construction: join the selected instances in the result.	Pattern matching, filtering by attributes values.
2. Promoting Attributes to Actors	Extraction: P_E and its filters identify the relevant actors A_i and at least one of their attributes m, l_i . Construction: P_C defines for each pair $A_i - l_i$ a new structure including a new relation $R_i = f(A_i, m)$ and a new actor $A_j = g(l_i)$. See Figure 5.3.	Pattern matching, functional creation of new object identifiers, pattern production.
3. Identifying Brokers	Extraction: $P_E = P_{pos}$ AND-NOT P_{neg} , the match requires that P_{pos} must exits and P_{neg} must not exists in S . Construction: P_C attach a constant label identifying the broker actor.	Pattern matching, negation, pattern production.
4. Counting Binary Relations	Extraction: P_E is an aggregation pattern, for each pair of nodes $A_i - A_j$ reports the nodes ids and the count of relations between them. Construction: A new relation $R_i = f(A_i, A_j)$ is created along with the count attribute. See Figure 5.10.	Pattern matching, aggregation, pattern production.
5. Ego-network selection	This requires two queries: Q_1 identifies all immediate neighbors of ego using a basic P_E pattern, and Q_2 induces all relations from S that have both of their participant actors identified by Q_1 .	Pattern matching, induced subgraphs.
6. Reachable neighborhood	This requires two queries: Q_1 identifies all reachable neighbors using TC (note that each TC pattern match adds to the output only the endpoint reached; TC proceeds recursively until no more reachable neighbors can be added), and Q_2 induces all relations from S that have their both participant actors identified by Q_1 .	Pattern matching, transitive closure, induced subgraphs.

5.3.3 SNQL Text Based Syntax

An SNQL query Q follows the standard `SELECT|CONSTRUCT – WHERE – FROM` structure of languages like SQL and SPARQL. It receives social networks as input (the `FROM` clause), extracts information using patterns (the `WHERE` clause), and outputs a new social network, possibly with newly calculated values, using the `CONSTRUCT` clause.

```
Q ::= CONSTRUCT <list-of-construct-patt>
      WHERE      <extract-patt>
      FROM       <list-of-social-networks>
```

Each *construction pattern* in `<list-of-construct-patt>` is a collection of `<list-of-patt-triples>` (set of triples including variables) possibly constrained by a list of equalities:

```
<construct-patt> ::= <list-of-patt-triples>
  [IF <expr> = <expr>[ AND <expr> = <expr>]*] [AS <sn-id>]
```

The `<extract-patt>` is either a *basic pattern* `<list-of-patt-triples>` (to be matched against one of the *social networks* listed in the `FROM` clause), or a complex pattern built using operations on patterns:

```
<extract-patt> ::= <list-of-patt-triples> [MATCH <sn-id>]
  | <extract-patt> AND <extract-patt>
  | <extract-patt> OR <extract-patt>
  | <extract-patt> AND-NOT <extract-patt>
  | <extract-patt> FILTER <condition>
  | TC(<startV>, <endV>, <extract-patt>) WITH <start-condition>
  | AGG(<group-vars>, <aggr-func>, <extract-patt>)
  | PROJECT(<var-list>, <extract-patt>)
```

where TC denotes transitive closure, AGG denotes aggregation, and PROJECT denotes projection. Finally `<list-of-social-networks>` is a list of sources (social networks) with aliases:

```
<list-of-social-networks> ::= <social-network> [AS <sn-id>
  [, <social-network> AS <sn-id>]*]
```

Let us recall that a social network is a collection of triples comprised of object identifiers and constant literals (see Chapter 4).

See the complete syntax in Figure 5.4.

Example 5 Consider again the SN from Example 3. In the network of persons that are friends depicted in Figure 5.1a, the possibly relevant fact that two persons live in the same city is not represented as part of the structure. It is possible to define a query that transforms cities from attributes to actors, and thus produces a network where all persons that live in the same city are connected to the same actor (which represents that city), see Fig. 5.1b. We call this transformation a promotion of an attribute to actor. This query creates a new kind of actor (*city*), and a new kind of relation (*lives-in*) to associate people with cities. The skeleton of the query that references the patterns in Figure 5.3 is the following:

<SN-expression>	::= <SN-query> <social-network>
<SN-query>	::= CONSTRUCT <list-of-construct-patt> WHERE <extract-patt> FROM <list-of-social-networks>
<list-of-construct-patt>	::= <construct-patt>[AS <sn-id> [, <construct-patt> AS <sn-id>]*]
<construct-patt>	::= <list-of-pattern-triples> [IF <list-of-equalities>]
<list-of-equalities>	::= <expr> = <expr> [AND <expr> = <expr>]*
<extract-patt>	::= <list-of-pattern-triples> [MATCH <sn-id> <extract-patt> AND <extract-patt> <extract-patt> OR <extract-patt> <extract-patt> AND-NOT <extract-patt> <extract-patt> FILTER <condition> TC(<startV>,<endV>,<extract-patt>) WITH <start-condition> AGG(<group-vars>,<aggr-func>,<extract-patt>) PROJECT(<var-list>, <extract-patt>)]
<list-of-pattern-triples>	::= {<pattern-triple> [, (<pattern-triple>)]*}
<pattern-triple>	::= (term, term, term)
<list-of-social-networks>	::= <social-network>[AS <sn-id> [, <social-network> AS <sn-id>]*]
<condition>	::= <logic-expression>
<term>	::= <variable> <constant> <expr> AS <variable>
<expr>	::= <variable> <constant> <function>(<expr>[, <expr>]*)

Figure 5.4: SNQL Syntax.

```

CONSTRUCT CP  IF R2 = f(A1, A2) AND A2 = g(L1)
WHERE      EP
FROM      FriendshipNetwork

```

FriendshipNetwork is the SN shown in Fig. 5.1a.

The full text of the query, including textual representation of P_E and P_C as list of triples, is the following:

```

CONSTRUCT {(A1, isa, person), (A2, isa, city), (R1, isr, friendship),
           (R2, isr, lives-in), (A1, inhabitant, R2), (A1, P1, R1),
           (A1, name, L2), (A2, place, R2), (A2, name, L1)}
IF R2=f(A1, A2) AND A2=g(L1)
WHERE      {(A1, isa, person), (R1, isr, friendship),
           (A1, city, L1), (A1, P1, R1), (A1, name, L2)}
FROM      FriendshipNetwork

```

From a SNA point of view new structural analysis options arise. The resulting cities become hubs that explicitly connect all people living in the same city; to achieve this key transformation, the new actors –the cities– require the functional creation of their identifiers from the collected data: the identifiers of cities are produced by applying a function g to the literal values bound to variable $L1$. Similarly, new relation identifiers for the ‘lives-in’ relation are created, one for each pair (person,city) matched by $(A1,A2)$.

The following should be noted with regard to the syntax in Figure 5.4:

- For FILTER, the <condition> is a Boolean expression over pattern variables and constants that further restricts the matches of <extract-patt>.
- For TC the variables <startV> and <endV> must appear in <extract-patt>. Then TC returns the transitive closure of the binary relation formed by all instantiations of <startV> and <endV> when matching <extract-patt>. TC allows the specification of an additional condition <start-condition> that is applied only to be able to initially locate the first match of the recurrent pattern.
- AGG returns a tuple of values comprising each distinct instantiation of the variables in <group-vars> along with the result of applying <aggr-func> to the remaining variables in <extract-patt>.
- PROJECT returns a tuple containing only the variables in <var-list>. It projects out all variables not in the given list. <var-list> must be a subset of the set of variables of <extract-patt>

Much like in SQL, the language allows to work with multiple source social networks which, in turn, may allow to express set theoretical operations with these queries (see Section 5.4.3).

It is worth mentioning a straightforward extension of SNQL: to produce tables instead of networks. In this case, each instance of the construction pattern in the output could be treated as the tuple comprised by all its variables; thus producing a table instead of a network as a result (as in SPARQL SELECT queries). This extension is not pursued in this work.

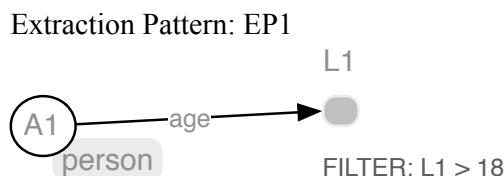


Figure 5.5: *FILTER operation graphical syntax*. The graphical syntax for *FILTER* operation simply adds the condition at the bottom of the pattern ($L1 > 18$).

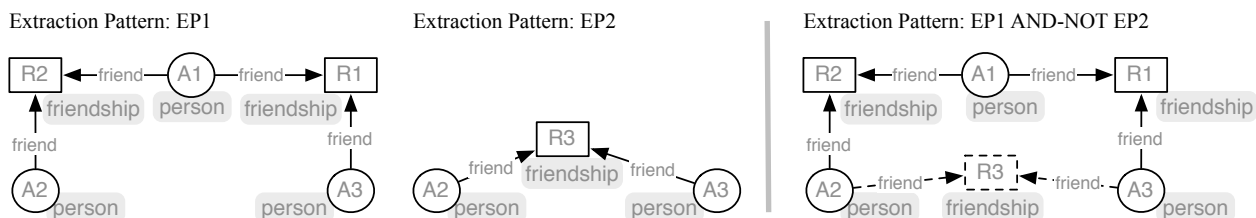


Figure 5.6: *AND-NOT operation graphical syntax*. The graphical syntax for *EP1 AND-NOT EP2* operation allows the combination of the two patterns in one diagram. Dashed lines identify *EP2*.

5.3.4 Graphical Syntax for Query Patterns

As in the case of the social networks data structure (see Section 4.2.3), a visual representation of query patterns would improve the usability of the language, specially to users with a SNA background. The basic pattern textual representation as a list of triples does not convey at a glance the structure of the pattern and the intended match, possibly obscuring the user interpretation of queries.

The graphical representation of basic patterns works exactly as in the case of graphical representation of social networks (see Section 4.2.3): actors as round nodes, relations as rectangles, attributes as grey dots, and so on. The only difference is in the labels: some of them represent variables. We already used this approach above in Figure 5.3.

The case of complex *extraction patterns* requires further analysis because, as discussed above, the syntax of *extraction patterns* also allows to combine them using several operations; however not all these operations have useful graphical representations. We define new graphical syntax elements only when they may improve the readability of the queries. In the following case by case discussion we assume that a graphical representation for each basic pattern (*EP1*, and *EP2*) has been provided.

- *EP1 AND EP2*, and *EP1 OR EP2*. We consider that in this case adding additional graphical elements does not improve readability over the existing text based syntax.
- *EP1 FILTER <condition>*. In this case, *<condition>* may be added at the bottom of the graphical representation of *EP1*. There is no need for additional graphical elements. For instance, if the *extraction pattern* is $\{n(A1, isa, "person"), m(A1, age, L1)\}$ *FILTER* $L1 > 18$, it could be graphically represented as shown in Figure 5.5.
- *EP1 AND-NOT EP2*. Provided that *EP1* and *EP2* are intended to match the same social network, it may improve the readability of the query to represent both patterns in the same diagram with *EP2* highlighted (dashed lines). This way it may be clearer which part must exist and which part must not exist in the source network. For instance, an extraction pattern *EP1 AND-NOT EP2* (see Figure 5.6) could specify that two actors are

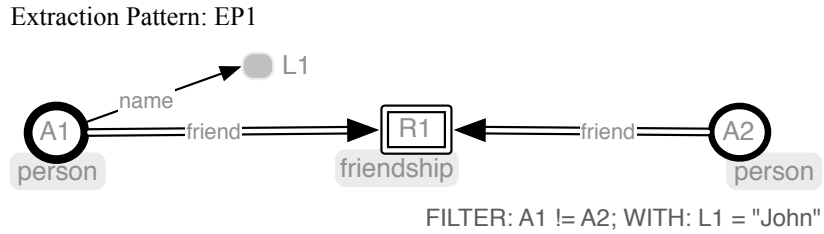


Figure 5.7: *TC operation graphical syntax*. The graphical syntax for $TC(A1, A2, EP1)$ WITH $L1 = \text{"John"}$ operation. Transitive patterns elements are drawn using double lines; start and end elements use bold lines. Conditions are included at the bottom of the pattern.

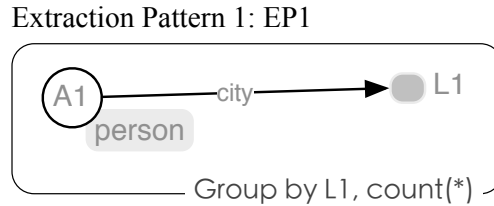


Figure 5.8: *AGG operation graphical syntax*. The graphical syntax for $AGG(L1, count(*), EP1)$ operation. The pattern is surrounded by a frame, including the grouping criteria and aggregated functions in its lower right corner. (This pattern is part of the query discussed in detail in Example 6).

both connected to a third actor (EP1) and that there does not exist a direct connection between them (EP2). Instead of representing EP1 and EP2 separately, they can be combined to emphasize the intended match.

- $TC(S, E, EP1)$ WITH $\langle \text{condition} \rangle$. In TC, for each pair (S, E) returned, the *extraction pattern* EP1 is transitively matched one or more times. To visually differentiate this behavior from a regular match, the graphical representation of EP1 is drawn with double lines, and S and E nodes are identified with a bold border. As in the case of FILTER, $\langle \text{condition} \rangle$ can be included at the bottom of the pattern. Figure 5.7 shows the following transitive closure *extraction pattern*: to find all reachable friends of persons named "John".

```
TC(A1, A2,
  {n(A1, isa, "person"), n(A2, isa, "person"),
   n(R1, isr, "friendship"),
   m(A1, name, L1),
   p(A1, friend, R1), p(A2, friend, R1)})
FILTER A1 != A2
WITH L1 = "John"
```

- $AGG(\langle \text{group-vars} \rangle, \langle \text{aggr-func} \rangle, EP1)$. For AGG operation the group of matches that share the same values for $\langle \text{group-vars} \rangle$ is consolidated in one result. This result includes the values bound to $\langle \text{group-vars} \rangle$ and the result of $\langle \text{aggr-func} \rangle$. To emphasize the grouping operation the whole pattern is framed, and the grouping criteria and aggregated functions are included at the bottom. Figure 5.8 depicts the *extraction pattern* to count all inhabitants of each city in the network.
- $PROJECT(\langle \text{var-list} \rangle, EP1)$. This operation is intended to select a subset of variables

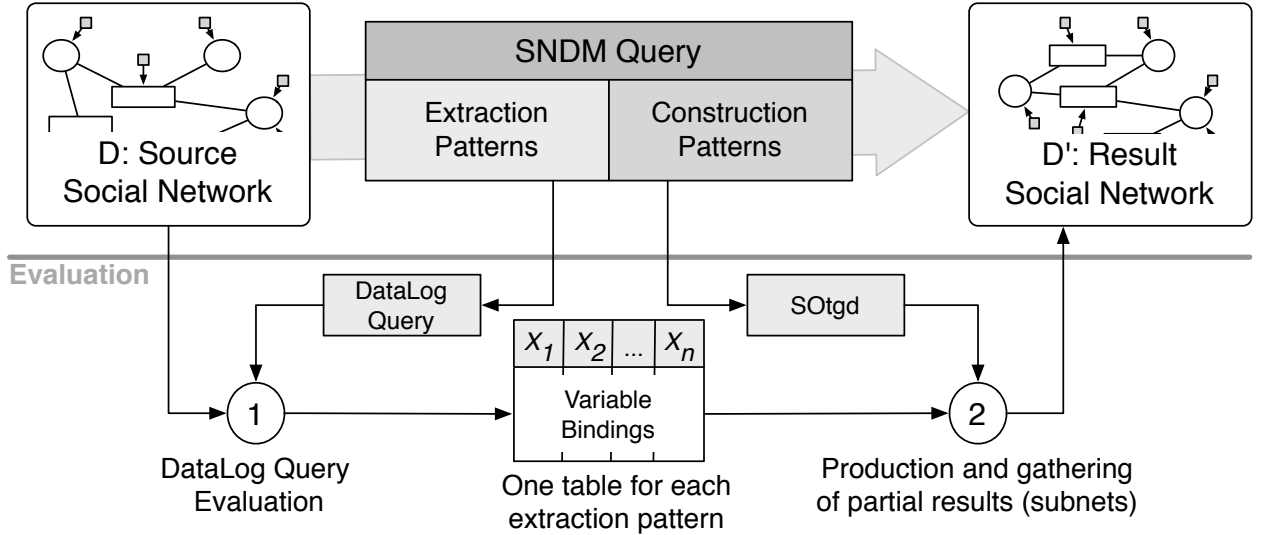


Figure 5.9: *The evaluation process of a SNQL query.* The evaluation process occurs in two stages: (1) a *Datalog* program equivalent to P_E is evaluated over the data source network D producing intermediate tables of variable bindings; (2) portions of the result are produced from the obtained tuples using the *SO tgd*s equivalent to P_C . All partial results are gathered in the final result network D' .

in the resulting tuples after the match. We do not consider a graphical syntax for projections on basic patterns. However, if it is required, it may be implemented simply highlighting the selected variables in the pattern.

5.4 Query and Transformation Language: Semantics

We define the semantics of SNQL following its two stages (see Figure 5.9). The collection of relevant data from the source networks is defined in terms of a *GraphLog* query [30], and the building of the output network is defined using *SO tgd*s transformations [45]. Both stages are expressed as one single *Datalog* [2] program using logical translations. This formal framework allows the study of the complexity and expressiveness of the SNQL.

In this section we present the semantics of SNQL in two parts. First, we define a translation of SNQL queries to a *Datalog* program P ; and second, we discuss the evaluation of the SNQL queries as a matching process.

5.4.1 Translation of Patterns to Datalog

In the spirit of the *logical translation* provided by Consens and Mendelzon [30] for *GraphLog query graphs*, we define the semantic of SNQL queries using a logical translation, that is, an equivalent *Datalog* program. We provide translations for basic patterns, extraction patterns, and construction patterns into *Datalog* rules. The semantics of a SNQL query is given by

the *Datalog* program P comprised by the set of rules produced.

The translation follows the definition of patterns in Section 5.3.1 (see Definitions 23, 24, and 25). Note that, in the following presentation, we use for each set of variables X the notation \bar{x} for the tuple constituted by all the variables in the set X .

Translation of Basic Patterns.

0. Translation of individual triples.

Each triple $t \in T$ of the form $\mathfrak{t}(A, B, C)$ is translated as a predicate $t(A, B, C)$, where t is n, r or m according to which set $-N, R, \text{ or } M-$ the triple \mathfrak{t} belongs to.

1. Translation of basic patterns (list of triples).

A basic pattern PATT, made up of the list of triples $\{ \mathfrak{t}_1, \dots, \mathfrak{t}_n \}$, is translated as the conjunction of the translations of each of its triples:

$$p(\bar{z}) \leftarrow \bigwedge_{t \in \text{PATT}} t(A_t, B_t, C_t) \quad (5.1)$$

Translation of Extraction Patterns.

An extraction pattern is recursively decomposed and translated to a *Datalog* program as follows: let PATT be the pattern to be translated to a predicate p , and assume that predicates p_1 and p_2 are the translations of patterns PATT1 and PATT2, respectively. Let \bar{z} , \bar{x} , and \bar{y} contain the projected variables of PATT, PATT1 and PATT2, respectively. The translation depends on the structure of PATT as follows:

1. Translation of conjunction of *extraction patterns* PATT1 AND PATT2.

$$p(\bar{z}) \leftarrow p_1(\bar{x}), p_2(\bar{y}). \quad (5.2)$$

2. Translation of disjunction of *extraction patterns* PATT1 OR PATT2.

$$\begin{aligned} p(\bar{z}) &\leftarrow p_1(\bar{x}). \\ p(\bar{z}) &\leftarrow p_2(\bar{y}). \end{aligned} \quad (5.3)$$

Note that disjunction requires that Z satisfy $Z \subseteq X \cap Y$.

3. Translation of PATT1 AND-NOT PATT2.

$$p(\bar{z}) \leftarrow p_1(\bar{x}), \neg p_2(\bar{y}), d(\bar{y}). \quad (5.4)$$

where predicate $d(\bar{y})$ indicates that \bar{y} is in the domain.

Safe negation is not enough to cover the requirements of the query language. Instead we rely on closed world assumption to ensure tractability. Hence the predicate d over the variables in negated predicate p_2 .

4. Translation of *extraction pattern* filtering PATT1 FILTER C.

$$p(\bar{z}) \leftarrow p_1(\bar{x}), c(\bar{x}). \quad (5.5)$$

where condition C is simulated by predicate c .

5. Translation of transitive closure of an *extraction pattern*
 TC (Vs, Vt, PATT1) WITH <start-condition>.

$$\begin{aligned} p(U, V) &\leftarrow p_1(\dots U \dots V \dots), \text{start_cond}(\dots U \dots V \dots) \\ p(U, V) &\leftarrow p_1(\dots U \dots W \dots), p(W, V) \end{aligned} \quad (5.6)$$

where variable Vs corresponds to variable U and Vt to variable V of $p_1(\bar{x})$.

6. Translation of the aggregate operator AGG(VList, AggF, PATT1) .

$$p(\bar{z}, \mathbb{A}(\bar{y})) \leftarrow p_1(\bar{x}) \quad (5.7)$$

where Vlist is the set of variables Z , and AggF is the aggregate function \mathbb{A} over the set of variables Y . Note that the following relation among the sets of variables must hold: $Y \cup Z \subseteq X$. (We assume non-recursive aggregation [31]).

7. Translation of projection operation PROJECT(Z, PATT1).

$$p(\bar{z}) \leftarrow p_1(\bar{x}) \quad (5.8)$$

where $Z \subseteq X$.

Translation of Construction Patterns.

The translation of the *construction pattern* is built using the predicate p , obtained from the *extraction pattern* PATT in the translation defined above, along with the list of triples and, if they exist, the corresponding lists of equalities in the *construction pattern*. The equalities are of two types. One type defines each variable: $v_i = \text{term}_i$, $1 \leq i \leq k$; the other is of the form $\text{term}_i = \text{term}_l$, where each term may contain variables (from p), constants and functions.

For a given CONSTRUCT trList IF eqList, the translation process takes the result of the extraction process, the $p(\bar{z})$ predicate, plus the list of equalities eqList translated as $\bigwedge_j \text{eq}_j$ to produce the following intermediate rule:

$$\text{construct}(v_1, \dots, v_k) \leftarrow p(\bar{z}) \wedge \bigwedge_j \text{eq}_j. \quad (5.9)$$

Finally, each triple \mathbf{t} in trList is translated as a new rule in the *Datalog* program as follows:

$$t(u_1, u_2, u_3) \leftarrow \text{construct}(\bar{v}). \quad (5.10)$$

such that $u_1 \in \bar{v}$, $u_2 \in \bar{v}$, and $u_3 \in \bar{v}$.

Example 6 (*Grouping and Aggregation*) The following SNQL query Q produces the network depicted in Fig. 5.1(c) from that in Fig. 5.1(a), grouping people by city and counting friendship relations between cities.

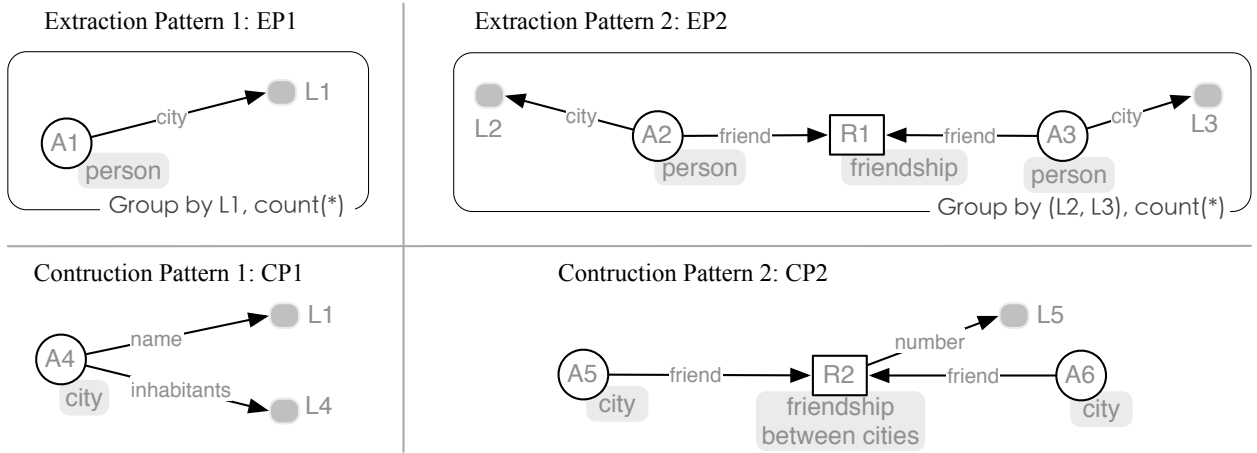


Figure 5.10: *Grouping and Aggregation*. Patterns EP1 and CP1 group people by ‘city’ and count the number of *inhabitants* in each city. Patterns EP2 and CP2 group and count friendship relations between pairs of cities.

```

CONSTRUCT CP1 IF A4 = f(L1) AS SN1
  WHERE AGG({L1}, COUNT AS L4, EP1)
  FROM FriendshipNetwork
UNION
CONSTRUCT CP2 IF A5 = f(L2) AND A6 = f(L3) AND R2 = g(A5, A6) AS SN2
  WHERE AGG({L2,L3}, COUNT AS L5, EP2 FILTER (L2 != L3))
  FROM FriendshipNetwork

```

Patterns EP1, EP2, CP1, and CP2 are depicted in Fig. 5.10. Note that each new group (actor) requires a new oid functionally produced from the value of attribute ‘city’. Also the number of inhabitants bound to L4 and the number of friendship-between-cities bound to L5 must be computed with the aggregate function COUNT. The first argument of AGG is the set of grouping variables, the second is the aggregation function required. and the third is an extraction pattern. The results of the two construct queries are combined using UNION to produce the desired result.

The translation of Q to Datalog is shown in Figure 5.11.

5.4.2 Query Evaluation Process

An SNQL query maps social networks to social networks using pattern matching (see Definition 26). As we said before, at the abstract level, the semantics of the SNQL query

$$\text{CONSTRUCT } \langle \text{CP} \rangle \text{ WHERE } \langle \text{EP} \rangle \text{ FROM } \langle \text{S} \rangle$$

can be considered as comprising two steps: (1) generate an intermediate table $Temp$ with the results (tuples) of matching the *extraction pattern* EP against the network S , and (2) construct a network, matching the basic pattern in CP to the values in the intermediate table $Temp$. Thus, the resulting social network SN is the set of instantiations of each triple t in the list of triples in the *construct pattern*.

```

output-n(A4,isa,city)      :- construct1(A4,L1,L4)
output-m(A4,name,L1)      :- construct1(A4,L1,L4)
output-m(A4,inhabitants,L4) :- construct1(A4,L1,L4)

output-n(A5,isa,city)      :- construct2(A5,R2,A6,L5)
output-n(R2,isr,
friendship-between-cities) :- construct2(A5,R2,A6,L5)
output-n(A6,isa,city)      :- construct2(A5,R2,A6,L5)
output-r(A5 friend,R2)     :- construct2(A5,R2,A6,L5)
output-r(A6,friend,R2)    :- construct2(A5,R2,A6,L5)
output-m(R2,number,L5)    :- construct2(A5,R2,A6,L5)

construct1(A4,L1,L4)       :- ag1(L1,N), A4=f(L1), L4=N
ag1(L1,count(A1))         :- ep1(A1,L1)
ep1(A1,L1)                :- n(A1,isa,person), m(A1,city,L1)

construct2(A5,R2,A6,L5)   :- ag2(L2,L3,M), A5=f(L2), A6=f(L3), R2=g(A5,A6),
                           L5=M
ag2(L2,L3,count(A2,R1,A3)) :- ep2(A2,A3,R1,L2,L3)
ep2(A2,A3,R1,L2,L3)      :- n(A2,isa,person), n(R1,isr,friendship),
                           n(A3,isa,person), r(A2,friend,R1),
                           r(A3,friend,R1), m(A2,city,L2), m(A3,city,L3),
                           L2 != L3

```

Figure 5.11: Translation of query in Example 6 to Datalog.

In practice, the evaluation process may proceed more efficiently by avoiding intermediate materialization: each match of the extraction pattern produces a collection of tuples corresponding to *Datalog* predicates, and this collection of tuples is further processed by the construction pattern to produce the output, one pattern instance per predicate.

The process to compute the evaluation $\llbracket Q_{\langle P_E, P_C \rangle} \rrbracket_D = D'$ of a SNQL query (Definition 26) is based on the matching of *extraction pattern* P_E over the network D , and then on the matching of the *construction pattern* over the table of intermediate results.

To define the semantics of matching we follow the approach used by Pérez et al. [89] to define the semantics of SPARQL, which is also based on pattern matching (they call it *graph pattern evaluation over D*). Pérez et al. define a *mapping* $\mu : V \rightarrow T$, where μ is a partial function from the set of variables V to the active domain T .

In the case of SNQL, to allow the mapping of patterns containing constants, the domain of μ must be extended to constants in the active domain: $\mu(c) = c$, thus we have $\mu : V \cup T \rightarrow T$.

We also extend the notation of the mapping to triples and patterns.

- For a given triple $t = (u_1, u_2, u_3)$, where u_i is a variable in V or a constant, the mapping over the triple is defined as

$$\mu(t) = (\mu(u_1), \mu(u_2), \mu(u_3))$$

Table 5.2: Table *Temp*: collection of extraction pattern mappings.

	X	Y	Z	...
μ_1	a	b	c	...
μ_2	e	f	g	...
μ_3	a	b	c	...
...

- For a basic pattern $P = t_1, \dots, t_n$, the mapping over the triple is defined as

$$\mu(P) = \{\mu(t_1), \dots, \mu(t_n)\}$$

Definition 27 (Pattern Matching) *Let D be a social network represented as triples over T , and P a pattern. A match occurs when there exists a μ such that $\text{dom}(\mu) = \text{var}(P)$ and $\mu(P) \in D$. Here $\text{dom}(\mu)$, the domain of μ , is the subset of V where μ is defined, and $\text{var}(P)$ is the set of variables occurring in pattern P .*

Note that for general patterns, it is possible to extend the definition of the mapping using the properties demonstrated for SPARQL by Pérez et al [89].

This definition produces, after the first stage of matching the *extraction pattern*, a table *Temp* of values for each variable in each mapping: a set of mappings (see Table 5.2).

For the second stage, the same notion of mapping is used. This time the *construction pattern* P_C is mapped against *Temp* producing the values for the result: the set of triples D' . Thus the resulting social network D' is the union of the mappings of the triples in P_C :

$$D' = \bigcup_j \{\mu(t_j) : t_j \in P_C\}. \quad (5.11)$$

5.4.3 Data Manipulation and Set Theoretical Operations

Up to this point, we have presented in detail SNQL from the perspective of extracting and transforming social networks from social networks, which is the main focus of this work. A practical database language for SN also requires data manipulation operations (i.e. insert, update, and delete), and set theoretical operations (see Section 5.1). SNQL is also capable of expressing data manipulation and set theoretical operations. In this section we discuss how this operations can be expressed. Nevertheless, it is worth noting that a complete presentation of these topics requires a well-defined set of integrity constraints for the model. Consequently here we restrict our presentation to show how SNQL expresses these operations and the main issues involved.

Data Manipulation Operations. SQL-style data manipulation operations for social networks: INSERT-INTO-WHERE, DELETE-FROM-WHERE, and UPDATE-WHERE, could be simulated in

SNQL. The *extraction pattern* identifies parts of the database that must be changed, and the *construction pattern* defines the changes to be applied, for example, a triple to be added in an insert operation. Here, we identify two main issues:

- Integrity of the result. As in relational databases, changes in the source network may render it inconsistent. To resolve this issue, we need a comprehensive list of integrity constraints, and methods to check, keep, and restore them.
- Source network updating. All previous discussion on SNQL assumes that the source network never changes by the action of a query; that is, all changes are produced in a new network. At a purely theoretical level, we could consider that updating operations actually create a new network that replaces the source network. At the practical level, this is not an option; the evaluation process should be redefined to reflect the changes directly in the source network, ensuring that they do not modify the final result.

Set theoretical operations. From the requisites listed earlier (see Section 5.1), it is clear that at least two set theoretical operations are needed: union to combine networks, and difference to compare them. In this case, it is preferable to consider alternate syntax and semantics for triples to avoid the need to define a custom union operation (for instance, as the pairwise union of the corresponding sets of two social networks). The alternate representation generalizes the current representation allowing the specification of triples without necessarily specifying to which set they belong. In a pattern this allows to match a triple in any of the three sets. It is possible to express the current representation by adding a filter to the pattern. Equation 5.12 shows the equivalence between the two representations for a triple t in T , with corresponding edge labels in L .

$$t(a, b, c) \equiv (a, b, c) \text{ FILTER } (b \in L) \quad (5.12)$$

To work as expected, this approach requires that the sets of labels allowed in each set of triples should be pairwise disjoint. For a social network $S = (N, R, M)$ we know, by the Definition 22, that the labels in triples in sets N , R , and M must come only from sets of labels $\{isa, isr\}$, L_{AT} , and $L_M = L_{AC} \cup L_{TC}$ respectively. Furthermore, Definition 20-bis states that L_{AT} , L_{AC} , and L_{TC} are pairwise disjoint. Finally, without losing generality, we assume that they also are pairwise disjoint with $\{isa, isr\}$.

Whenever it is possible we keep using the syntax $t(a, b, c)$ to indicate the type of the triple as a notation to aid the reader to distinguish triples.

Union of social networks $G_3 = G_1 \cup G_2$ could be expressed as

```

CONSTRUCT {(A, B, C)} as G_3
WHERE      {(A, B, C)} MATCHES G_1
           OR
           {(A, B, C)} MATCHES G_2
FROM      G_1, G_2

```

In the case of difference, it must be noted that this operation could produce social net-

works with integrity problems. It is convenient, to avoid this problem, to compute difference regarding a certain priority pattern, in the simplest cases actors or relations (as in node/link driven minus defined by Amer-Yahia et al. [5]). Let us consider actor-driven difference; it requires two SNQL queries, one to identify the base pattern (e.g. actors), and another to induce the structures that tie the base patterns together (e.g. binary relations). The first query $Q1$ is stated as follows:

```

Q1:
CONSTRUCT {n(V, isa, F)} AS G'
WHERE      {n(V, isa, F)} MATCH G_1
           AND-NOT
           {n(V, isa, F)} MATCH G_2
FROM      G_1, G_2

```

In the second query $Q2$ all relations between the identified actors are induced.

```

Q2:
CONSTRUCT {n(R, isr, F), m(R, M, L), r(U, P1, R1), r(V, P2, R),
           n(U, isa, F1), m(U, M1, L1),
           n(V, isa, F2), m(V, M2, L2)}
WHERE      ({n(U, isa, F1), n(V, isa, F2)} FILTER U!=V MATCH G')
           AND
           ({n(R, isr, F), m(R, M, L), r(U, P1, R), r(V, P2, R),
            n(U, isa, F1), m(U, M1, L1),
            n(V, isa, F2), m(V, M2, L2)} MATCH G_1)
FROM      G_1, G'

```

5.5 Expressiveness and Complexity

SNQL is composed of two modules: one for extracting information and another for constructing a new network. In the design, consideration has been given to providing the maximum expressiveness possible, while keeping the complexity of processing within reasonable bounds. First, for the extraction stage, we considered *GraphLog* (possibly with summarization functions), which is a graph query language designed to be simple, graphical, oriented to graphs, and be as expressive as possible while staying within the NLOGSPACE complexity bound [30]. Second, for the construction module, whose main purpose is to compute identifiers and values in the process of creating the new network, the language is modeled after second-order tuple-generating dependencies (*SO tgds*), which are known to be a family of transformations between tables of tuples with good expressiveness/complexity tradeoff [45].

5.5.1 Expressiveness

Knowing this, it comes as no surprise that SNQL covers all use cases we identified as being used in current practice by SN researchers. (There are still some queries defined theoretically by SN scientists which are not covered by SNQL; however, it can be proved that they fall out

of the scope of a reasonably efficient complexity bound. A typical example are the cohesive subgroups defined in terms of shortest paths lengths between members, for instance *n-clans*.)

Formally presented, this result can be stated as follows:

Claim: *SNQL solves all use cases presented from SN practice that fall in the NLOGSPACE complexity bound.*

A proof of this claim relies on the list of use cases in current practice. The column “Required QL Features” of Table 5.1 collects the features needed for the classical use cases from the SN community. All of them, except induced subgraph, are incorporated directly in the language.

The language is expressive enough to cover all SNA use cases, and still has a good complexity as shown by the following theorem.

Theorem 1 *The expressive power of the SNQL extraction module is contained by GraphLog.*

Proof. From the semantics of the translation to *Datalog* presented (see Section 5.4), it is clear that every `<extract-patt>` can be expressed in *GraphLog*. The semantics of a *GraphLog* query graph are also defined by a logical translation to *Datalog* [30], producing rules very similar to those defined in Section 5.4.

□

Remark. Note that the converse of Theorem 1 has problems. Even though both data structures are directed labeled multigraphs, the underlying data structure of extraction patterns has two additional restrictions:

- Basic patterns are tripartite.
- Only single symbol labels are allowed, whereas in *GraphLog* query graphs the labels can be tuples of symbols.

Hence, when trying to express *GraphLog* query graphs as extraction patterns, the reverse translation of a *GraphLog* logical translation may not be a valid extraction pattern.

Theorem 2 *The construction module can be specified by one SO tgds of the form:*

$$\exists f_1 \dots f_m (\forall \bar{x}_1 (\phi_1 \rightarrow \psi_1) \wedge \dots \wedge \forall \bar{x}_n (\phi_n \rightarrow \psi_n)),$$

where each $(\phi_i \rightarrow \psi_i)$ has the form:

$$(p(\bar{x}) \wedge \bigwedge_k eq_k) \rightarrow (t_1 \wedge \dots \wedge t_r), \quad (5.13)$$

where $p(\bar{x})$ and eq_k follow the notation of equation (5.9), that is, predicate $p(\bar{x})$ is the result of the processing of the extraction pattern, and the t_j and eq_k are predicates resulting

from the translation of the triples and equations in the *CONSTRUCT* clause, and each tuple \bar{x}_i includes all variables in p and in the eq_k 's.

Proof. Note that from their semantics it follows that the evaluation of each expression $\langle \text{list-of-triples} \rangle \text{ IF } (\langle \text{exp} \rangle = \langle \text{exp} \rangle)^*$ over the results of $\langle \text{extract-patt} \rangle$, can be simulated by a formula of the form $(\phi_i \rightarrow \psi_i)$ as in eq. (5.13), where $p(\bar{x})$ corresponds to the translation of the extraction pattern. Hence the conjunction of these formulas can simulate the whole *CONSTRUCT* expression. This is precisely a *SO tgds*. (Note that *SNQL* uses predefined functions in the *CONSTRUCT* clause, while *SO tgds* allow existential functions.) \square

5.5.2 Complexity

A naïve implementation of the semantics presented in Section 5.4 (see also Fig. 5.9) would materialize intermediate results. This can be avoided by using the algorithm in Figure 5.12.

(a) Translate extraction pattern *EP* into a *Datalog* program L that computes predicate $p(\bar{x})$. L is contained in the logical translations of *GraphLog*.

(b) For an expression

```
CONSTRUCT trList1 IF eqList1,
      ...
      trListn IF eqListn
```

translated in the form of eq. (5.13), define n clauses:

$$aux_j(\bar{x}_j) \leftarrow p(\bar{x}) \wedge \bigwedge_k eq_k, \quad j = 1, \dots, n.$$

(c) For each clause $\text{trList}_j \text{ IF } eqList_j$; and each triple (x, y, z) in trList_j , define a rule $t(x, y, z) \leftarrow aux_j(\bar{x}_j)$.

(d) Add the clauses generated by (a) and (b) to the *GraphLog* program L generated from the extract module.

(e) Obtain the values of the triples to be generated by running the new program.

Figure 5.12: Evaluation Algorithm.

Lemma 3 *The Evaluation Algorithm preserves the semantics.*

Proof. First, note that the resulting program is a *Datalog* program. (The clauses added are simply predicates plus equalities.) Hence, it follows that this evaluation procedure gives the correct result (given by the original semantics). This is only possible, though, because of a subtle point: each clause $\phi_j \rightarrow \psi_j$ in the *SO tgds* can be evaluated independently because we have fixed functions. (If not, the rules would not be independent. For example, assume $a = f(x)$ is in $eqList_1$ and $b = f(x)$ in $eqList_2$ for a given variable function f). Then, if a given value of x fires rule 1, then it should not fire rule 2. \square

We will show that, from a database perspective, the above evaluation computes queries efficiently. As is customary when studying the complexity of the evaluation problem for a query language [108], we consider its associated decision problem. We denote this problem by `EVALUATION` and define it as follows.

INPUT : A Social Network S , a query Q and a triple $t = (a, b, c)$.

QUESTION : Is $t \in \llbracket Q \rrbracket_D$?

Theorem 3 *The data complexity of `EVALUATION` is in `NLOGSPACE`.*

Proof. Note that adding the clauses in (b) and (c) of the Evaluation Algorithm (see Figure 5.12) to the *GraphLog* program L produces a *Datalog* program, and the size of that program stays in the bounds of the original query. Hence to check if a given triple t is in $\llbracket Q \rrbracket_D$, we just have to instantiate each t in (c), hence in aux_j . Then, check the recursive equations in (b). Now check if $p(\bar{x})$ is in the *GraphLog* program left (note that it has no functional equations now). It is known that *GraphLog* can be evaluated in `NLOGSPACE` [30].

□

5.6 Related Work

We found relevant related work in different domains:

Data Management and Databases. A review of the current support for social networks data management (see Section 3.5.3) shows that, while a data model for networks is necessary, it is still an open problem. Jagadish’s report on this topic supports this idea [62, 63, 102]. From the database point of view, there are several proposals of data models for general graphs (Angles provides a comprehensive survey [8]), but none of them provides a functional and complete solution, nor the appropriate level of abstraction.

As part of the foundations of our data model we use *GraphLog* [30], which is a general graph query language. It is based on *Datalog* and its deductive data model, which has been thoroughly studied and characterized. The relations between `SNQL` and *GraphLog* can be summarized as follows:

- The *GraphLog* data structure covers the `SNDM` data structure. Besides that, *GraphLog* graphs not necessarily satisfy `SNDM` social networks integrity constraints, and *GraphLog* supports multi-element labels in nodes and edges, which `SNDM` does not support.
- Following the relation between data structures we show that the `SNQL` extraction module is included in *GraphLog*. The rest of *GraphLog* does not satisfy the integrity constraints of `SNDM`.
- Social networks practice requires the flexibility to construct arbitrary new networks from the data collected by the extraction module: a mechanism is required to define

new structures and to compute new values. Computation of new values cannot be freely added in *GraphLog*. We found that *SO tgds* provide a good mechanism with a complexity under the required bound.

- The final program is a piece of *GraphLog* (the logical translation of it) plus a layer of rules from *SO tgds*.

SPARQL¹, the query language for RDF, has many similarities with SNQL. Both have CONSTRUCT queries based on similar pattern matching strategies. However, the current recommendation SPARQL 1.0² does not provide two critical features of SNQL: functional creation of values, and transitive closure. The current working draft SPARQL 1.1³ includes the missing features, but the mechanism that provides transitive closure does not fit in SNQL design goals. SPARQL 1.1 defines a form of path expressions, called *property paths*, to find routes through a graph between two nodes. The evaluation of property paths returns collections of paths, even in cyclic graphs, which push the evaluation cost beyond the SNQL upper bound.

In the domain of molecular and cell biology, Eckman and Brown [42] present an extension to a RDBMS data model and query language: Systems Biology Graph Extender (SBGE). Interestingly, the authors provide a list of operations in four categories that shows a considerable commonality with the requirements of social networks. The four categories are: extract graph properties, compare graphs, compute graphs from graphs, and decompose a graph into its component parts.

Data Mining. From the data mining point of view, Jensen and Neville [64] present Proximity⁴, a system for relational knowledge discovery over MonetDB⁵. Proximity defines a graph data structure, and a graphical query language, QGraph [15]. The main differences with our approach are that attributes are not an explicit part of the structure, a QGraph query returns a collection of subgraphs, and the system is focused on data mining and machine learning models. The last update available of the system was published in November 2007.

Query and Transformation Languages on the Web. In the late 1990's the recent success of the Web as a global information source gave place to an active research domain on the application of database techniques as an alternative to the available search engines.

Florescu et al. [49] present a survey of the proposed models and query languages, which they classify in two generations. The first generation comprises languages like W3QL [70], and WebSQL [80], which query information from the Web, but do not offer any restructuring functionalities. Second generation languages, on the other hand, do have restructuring capabilities. The authors review the following second generation models and languages with the capacity of restructuring the content and/or the link structure of a collection of web pages: Lorel [3], WebOQL [9], UnQL [22], StruQL [47], and *Araneus (Ulixes and Penelope)* [10]. We

¹<http://www.w3.org/standards/techs/sparql>

²<http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/>

³<http://www.w3.org/TR/2012/WD-sparql11-query-20120105/>

⁴<http://kdl.cs.umass.edu/software/about.html>

⁵<http://monetdb.cwi.nl/>

found also that the declarative language WebLog [71] is capable of producing a restructured web of pages. In all these languages it is possible to distinguish a data extraction stage, and a data production (restructuring) stage; thus they follow the same general design of SNQL.

For instance, Lakshmanan et al. [71] present the WebLog declarative language that defines an extraction pattern as a set of rules over the content values and structure of each page, and the structure of links among them. These rules use predicates that represent existing pages, and navigation patterns (paths of links between pages). The variables in the rules are bound to information being collected. The construction phase uses rules whose heads define the pages to be constructed from the collected and computed information. WebLog uses “built-in predicates” as abstractions of external functions to enrich the language, for instance: `newlink(<string>,<hlink-id>)`, where the unique `<hlink-id>` is created to correspond to `<string>`.

Atzeni et al. [10] define the *Araneus* model, which includes the *Ulixes* extraction language, and the *Penelope* restructuring language. In these hypertext networks the web pages are the nodes connected by their links. Each node represent the structure of an entire page. The system works in two stages: first it defines and extracts database tables from web pages, and second, it can build new web pages from the intermediate tables. Queries in each stage are expressed with the corresponding language.

All these languages present three main shortcomings from the social networks data management point of view:

- Their patterns are “navigational patterns”, devised to represent paths along the links between web pages. It may be difficult or impossible to define the more structured and expressive patterns that social networks extraction requires.
- Their syntax is designed in the context of redefining the structure of existing web pages, to define new web pages, or to create new connections among web pages. Even if it is possible to define social networks patterns, it would be difficult to define and read those queries.
- The underlying data models reflect the structure of hypertext webs, and at a syntax level they do not resemble the customary model for social networks. Furthermore, none of them offers a graphical syntax for their queries.

More recently, in the context of semantic web, several languages have been proposed that query and transform XML (e.g. XSLT and XQuery) and RDF (SPARQL) data. Furche et al. [52] thoroughly review these languages. In the case of XML, Kepser [68] proved that both XSLT and XQuery are Turing complete and therefore are actually more expressive than required. They certainly can be used to express SNQL queries; however, GraphLog achieves the same goal under a better complexity. The relation with RDF and SPARQL is discussed above.

Social Networks Data Management. The three most similar proposals, that we are aware of, oriented to social networks and with a related aim and scope are: BiQL [41], SocialScope [5], and SoQL [92].

BiQL is a SQL-like language design with a set of features that shares the motivation of providing database support for SN and interoperation with analysis tools; however the authors do not provide an implementation, nor a study of its complexity properties. From our point of view, a major drawback is the choice of the data structure. Although, at a low level, the underlying graph is represented with a set of tables similar to the ones used in SNDM; these low level structures are exposed through the query language to the users, which is avoided by design in SNDM. It is worth mentioning that this model allows the creation of links among links –in this proposal there is no distinction between nodes and edges–, which is intentionally forbidden in SNDM. BiQL query language works selecting data using path expressions, and building collections of nodes and links, called *domains* –one of the low level structures we refer to above. SNQL patterns are more general structures, consequently, SNQL can express all the BiQL examples and cases discussed by the authors. Given the stated similitudes, it is a reasonable assumption that BiQL complexity should be similar or lower than the complexity of SNQL.

SoQL is also a SQL-like query language design for SN; it is focused on identifying paths and groups. In this model a SN is composed by actors and binary relations. SoQL defines three types of queries SELECT FROM PATH, SELECT FROM GROUP, and CONNECT USING PATH/GROUP. In each case the key feature is the ability to find paths and groups that fulfill given predicates. The syntax supports multiple aggregation levels and returning whole paths and summaries, which does not even ensure the practical tractability of computing complete answers of queries [31]. Although the authors seem to be aware of the complexity problems posed by their design, they do not provide any complexity assessment, only suggesting the use of deployment parameters to ensure the practicability of SoQL query evaluation (e.g. the maximal time to be spent on a single query evaluation).

SocialScope is a logical architecture for discovering, integrating, and managing social information in social content sites; thus, its scope goes beyond data management for this particular context. It comprises three layers: content management, information discovery, and information presentation. The SocialScope framework includes an algebraic language for manipulation of social content graphs (which are very large social networks that include a rich variety of objects); its main objective is to enable uniform data manipulation of social content graphs. The algebraic language includes selection and set theoretical operations, along with binary operations to combine networks and aggregation.

We think that, from the three proposals discussed here, SocialScope is the most closely related to SNDM/SNQL. SocialScope is strongly founded on a practical problem (examples come from Y!Travel service), and provides –among other features– a query language for its data managing requirements. We produced a translation of SocialScope query language to SNQL, and we found that SNQL covers the query language of SocialScope; see the full translation in Appendix B.

Chapter 6

Social Networks Data Model III: Implementations

One of the goals of our research was to test the data model in practice; specifically, to have a GUI environment to define, issue, evaluate, and present results of SNQL queries. Instead of embarking in the development of a proprietary DBMS, we defined proof-of-concepts prototypes using existing DBMS as database services back-end.

In the following sections we present our general choices regarding these implementations. Afterwards, we describe the main proof-of-concept prototypes developed, as well as related software. Finally, a simple performance assessment and its results are discussed.

6.1 General Implementation Remarks

We think that a proper implementation for SNDM should promote the adoption of the model, and support the collaborative evolution and improvement of the software. To promote the adoption of the model, the software should not only be accessible but appealing to the target set of users. The corresponding development and improvement effort is a resource-consuming and long-term effort that is best served in an open environment.

6.1.1 Software Design Choices

In this context we applied the following main design and implementation guidelines.

Modular Architecture. All software developed was done in a layered and modular architecture (see Figure 6.1), using an object oriented language. This design choice supports the distribution of implementation work across different teams, and facilitates the extension and update of the systems.

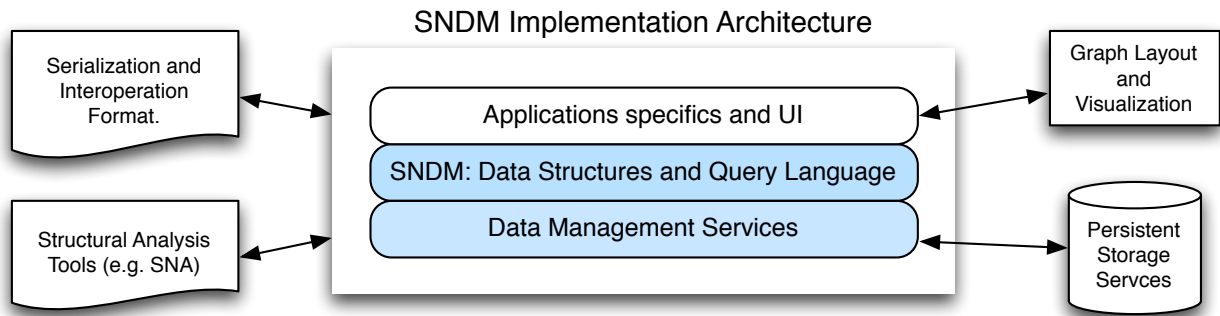


Figure 6.1: *A Modular Software Architecture for SNDM.* The figure shows the three layers that form the SNDM modular architecture, including service providers and libraries.

We defined three layers:

- **Application layer.** In this layer, user interface and all application specific functions should be implemented. For instance, a graphical social networks editor.
- **SNDM layer.** This layer implements the data model itself. Here, main memory data structures should be defined, along with the access to persistent data services. The query language must also be implemented here, providing the methods to evaluate SNQL queries against the networks database. In practice, this layer implements the SNDM interface between applications and data management services.
- **Data Management Services layer.** This layer implements the persistent data structures that store the networks and allow the evaluation of queries. Currently, it is implemented using other DBMS; however, in the future it should be implemented with the appropriate algorithms to evaluate SNQL queries.

Language. To guarantee free access to the source code and development tools, we preferred object oriented and freely available programming languages, mostly Java and PHP.

Libraries. Some functionalities, required to implement applications for social network data management, pose complex problems beyond the scope of the present work. For example, graph layout, visualization, and graphical manipulation of networks, which are provided by an external library (JUNG) in the prototypes implemented here. Each layer should be implemented to allow a flexible choice of supporting libraries and services.

6.1.2 Translations/Mappings to Existing DBMS

To keep the present work focused on the theoretical definition and study of the social networks data model, we postponed the implementation of the algorithms for network data management. Currently, proof-of-concept prototypes borrow the data management services from existing DBMS.

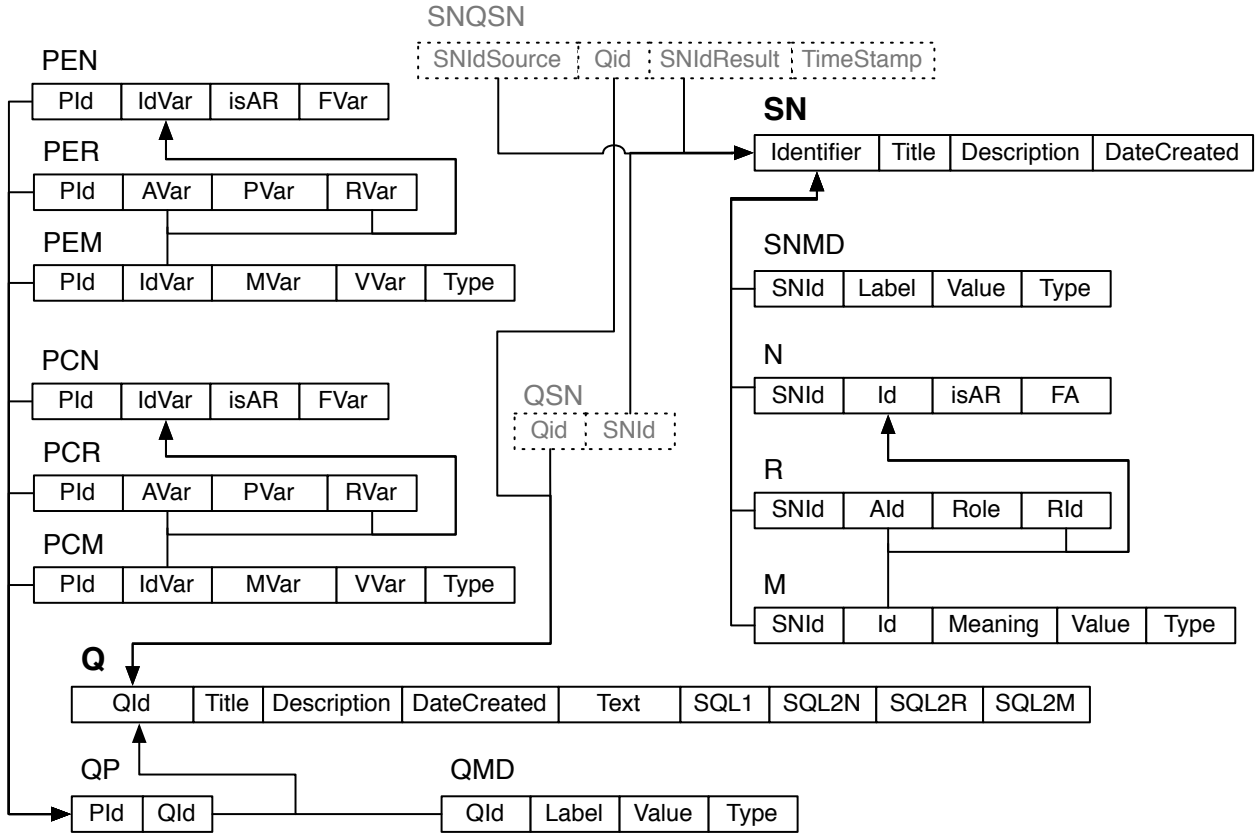


Figure 6.2: *SNDM Data Structure as a Relational Schema*. This schema implements the SNDM data structure (N , R , and M tables), and additional tables for queries and metadata.

Until now, we have tried two already implemented db-models and their query languages: relational db-model, and semi-structured RDF model.

RDBMS/SQL

Data Structure. The SNDM data structure can be implemented in a RDBMS as a collection of tables [94]: each set of triples in a social network is implemented as a separate table. Figure 6.2 shows an example of relational schema capable of representing social networks (tables N , R , and M). Additional tables are defined to contain network level metadata, and to allow the storage of queries.

Query Language. Each SNQL query is translated into four SQL queries (see Figure 6.3). The first stage of the evaluation process uses an SQL query to implement the extraction pattern PE . This stage produces a temporary table $Temp$. The second stage uses three SQL queries to produce the result network from $Temp$. Each of these queries produces the triples of construction pattern PC that correspond to one of the triple sets in the result. There are three issues with the translation to SQL: (1) the first query usually has many joins, (2) not all implementations of SQL99 allow recursion (feature required to support transitive closure), and (3) the materialization of table $Temp$ (its size can be exponential in the size of

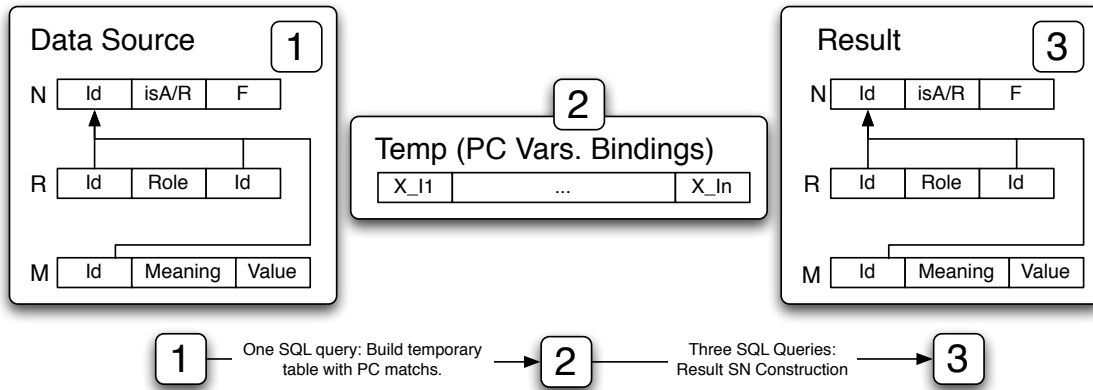


Figure 6.3: SNQL Query Evaluation Steps with SQL. It is possible to evaluate most SNQL queries in two steps. First, an SQL query (equivalent to PE) extract datas from the source network and produces a temporary table $Temp$, then three different SQL queries build each triple set in the result (PC).

the network). The translation of PE to SQL produces a query that has a number of joins proportional to the sum of node degrees in the pattern: $\sum(\text{degree}(u) - 1)$, where u is a node in the pattern.

RDF/SPARQL

Given its semi-structured graph model based on triples, RDF/SPARQL offers a very promising framework to implement SNDM. However, when we explored this approach, there were some missing features in available implementations, such as the functional creation of values, which was not supported in the standard [57].

Data Structure. Translation of the data structure is straightforward: each SNDM triple becomes an RDF triple. Network level metadata and multiple network storage can be provided using auxiliary triple stores. This structure is implemented using three sets of triples:

- A typing set N . Each triple $(oid, [isa|isr], family) \in N$ indicates that the actor or relation oid belongs to (is of type) $family$.
- A set R indicating roles. Each triple $(a_oid, role, r_oid) \in R$ indicates that the actor a_oid participates with $role$ in the relation r_oid .
- A set M describing attributes. Each triple $(oid, pred, v) \in M$ indicates that the actor or relation oid has the attribute $pred$ with value v .

It is possible to implement all sets in the same or separate triple stores. The convenience of using one approach or the other depends on the particular implementation of RDF/SPARQL used.

Query Language. **CONSTRUCT** queries in the SPARQL specification are very close to the requirements of SNQL, but they lack key functionalities: aggregate functions, transitive closure, and functional creation of values and ids. Note that blank nodes do not provide the required functionality, as long as they are all distinct in principle, and there is no programmatic control over their ids. At the time we tried a SPARQL implementation of SNQL,

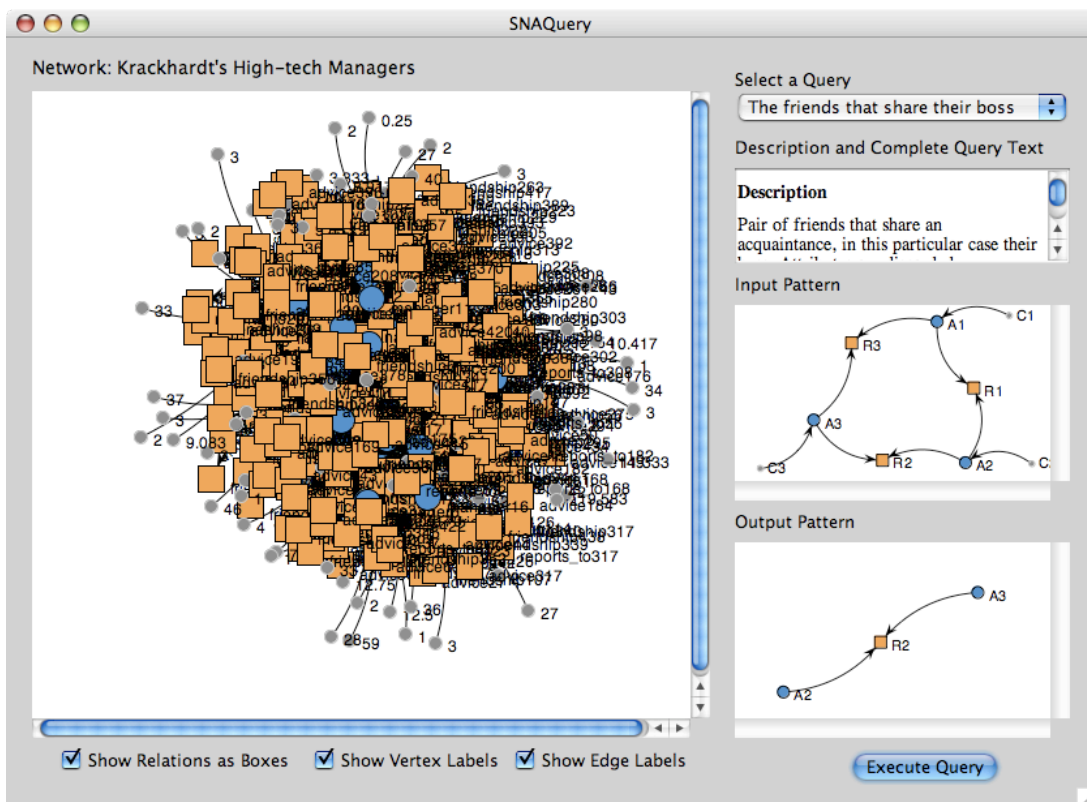


Figure 6.4: *SNAQuery Prototype Main Window*. The main window shows the complete network and the query to select the friends that have the same boss.

the current SPARQL specification included aggregation and partially functional creation of values (from a SNQL perspective), and the other missing features were only under consideration. Given this situation, we postponed a full implementation of SNQL over SPARQL, and we tested only the queries that were implementable at the moment [95].

6.2 Software

In this section we present the proof of concept prototypes developed during the project. These prototypes and the related datasets are available for download at <http://sn.dcc.uchile.cl/sndm/>.

6.2.1 SNAQuery: A First Prototype

Our first prototype consisted of a set of eight fixed queries over the Krackhardt's High Tech Managers (KHTM) social network, and a graphical user interface (see Figure 6.4). This prototype was devised to illustrate the key ideas and benefits of the SNDM. The queries were manually translated into SQL.

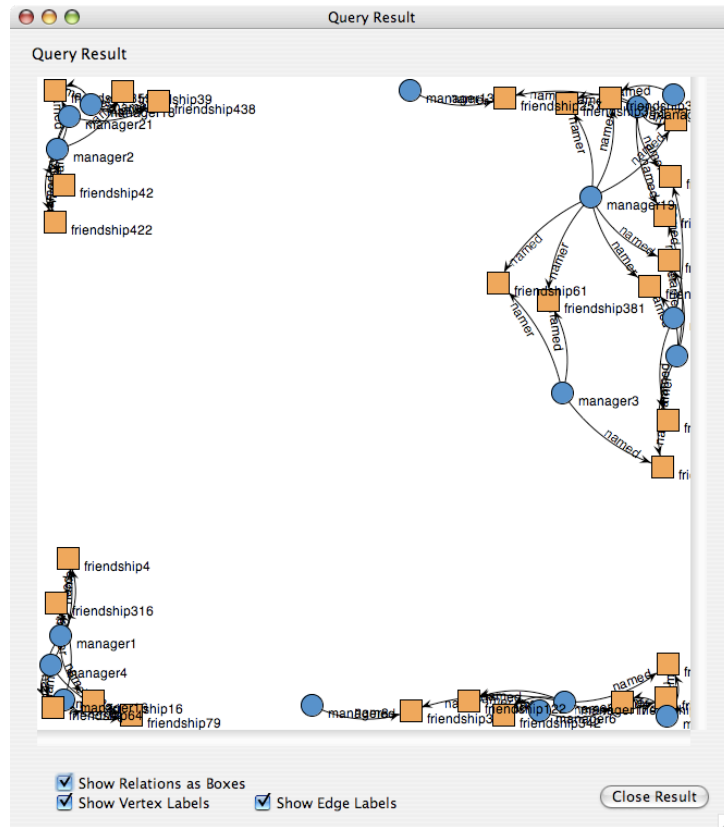


Figure 6.5: *SNAQuery Result Window*. This window shows the result network after selecting the friends that have the same boss.

This prototype implements the following queries: ego-network of a manager, all relations between a given pair of managers, the network under a given relation family, the friends that share their boss, mentor and boss (parallel relations grouping), friend of a friend (path contraction), grouping by boss (binary to n-ary relations), and relations between departments (actor and relation grouping). These queries were translated manually from an early version of SNQL to SQL.

The KHTM network was imported before to a RDBMS from a text file containing the data set as published by Wasserman [110]. The network is stored in the RDBMS as three tables for the N , R , and M triple sets.

All networks –source network, patterns and result networks (see Figure 6.5)– are presented graphically. For each query, a description and its full text representation are provided. This prototype is a Java application that requires the JVM 5 to run.

6.2.2 SNDMS

The most comprehensive proof-of-concept prototype built to date is a DBMS for the social network data model: SNDMS. This prototype has two versions, one maps the conceptual

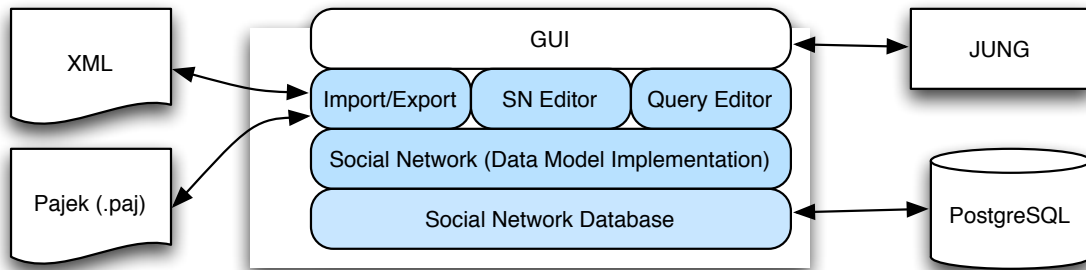


Figure 6.6: *SNDMS Architecture*. The figure shows the packages and libraries that form the SNDMS.

model to the relational model, using as data management back-end an RDBMS; and another –restricted by the underlying framework– that maps the model to RDF/SPARQL. The first is described in this section.

The SNDMS prototype is a Java application that implements a social network data managing environment as defined by the SNDM, using database services provided by PostgreSQL. Due to its modular architecture, other data management providers can be used. It is organized around the package that implements all data structures of the model, including social networks and queries, in main memory. The supporting packages are:

- **ie** (import and export). This package handles all interaction with external formats. The currently implemented two way translators include XML and Pajek. Social networks and queries can be imported and exported from and to XML. Only social networks expressible in Pajek format can be exported to it. Most Pajek networks can be imported.
- **visualization**. Social network layout, drawing, and graphical manipulation processes are implemented in this package using the JUNG library (<http://jung.sourceforge.net/>).
- **sne** (social networks editor). This package allows editing social networks graphically, and with an alternate interface based on tables.
- **qe** (query editor). This package allows editing queries graphically, and to execute them on an existing social network.
- **sndb** (social networks repository). In this version of the prototype, this package handles the interaction with the RDBMS that provides persistence for **sndm** objects. These module takes the SNQL queries translate them into SQL, send this SQL query to the database, receive and transform the result into a SNDM social network.

In the prototype version discussed here all social network data is primarily stored in the relational database –PostgreSQL– and on execution each SNDM query is translated into a sequence of five SQL queries that perform the actual query evaluation inside the RDBMS.

Example 7 (KHTM Analysis Data Workflow) *In this example, the KHTM network will be imported from XML to SNDMS and, once there, the network will be queried and analyzed using Pajek.*

1. *Import KHTM network from XML to SNDM: the KHTM network initially stored as an*

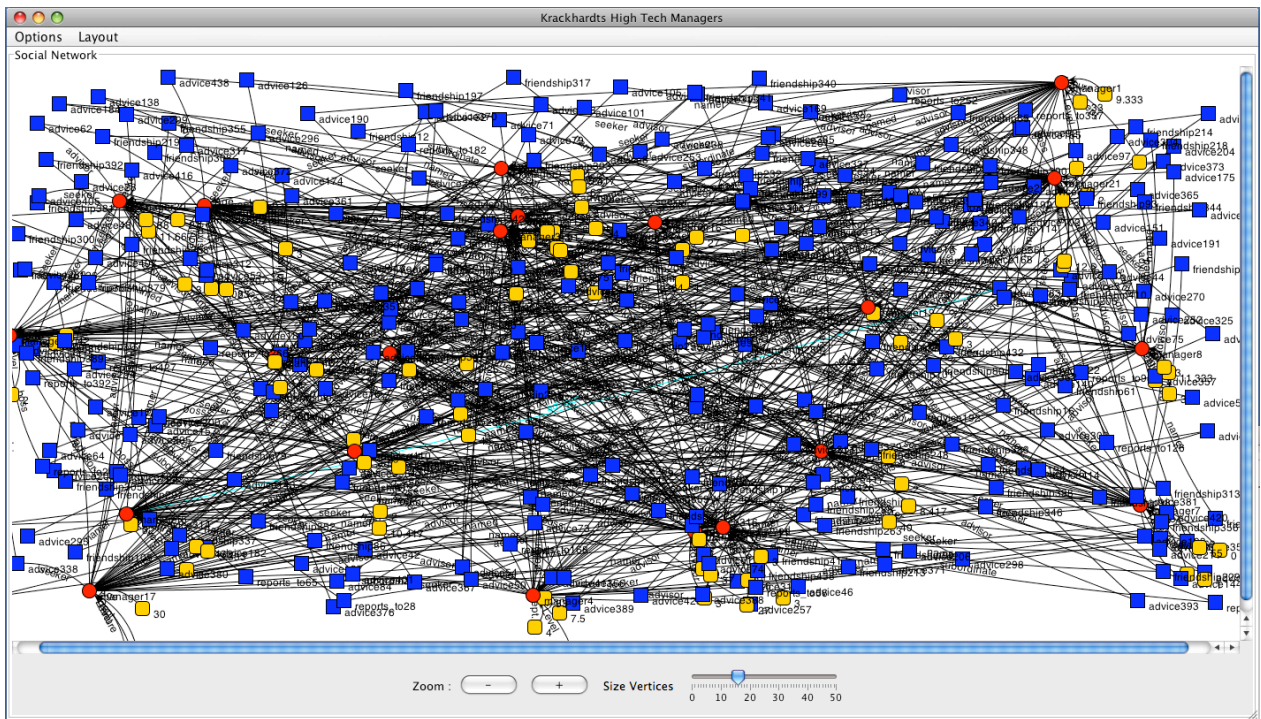


Figure 6.7: Graphical representation of KHTM network imported to SNDMS.

XML file is imported and stored in the database of the SNDMS (see Figure 6.7).

2. *Extract subnetwork defined by the “advice” relationship: with a SNQM query (see Figure 6.8) executed from the SNDMS, the network induced by the “advice” relationship is extracted without actors’ attributes, because they are not relevant in the computation of centrality. The relevant data is copied to a new network; the source network is not altered.*

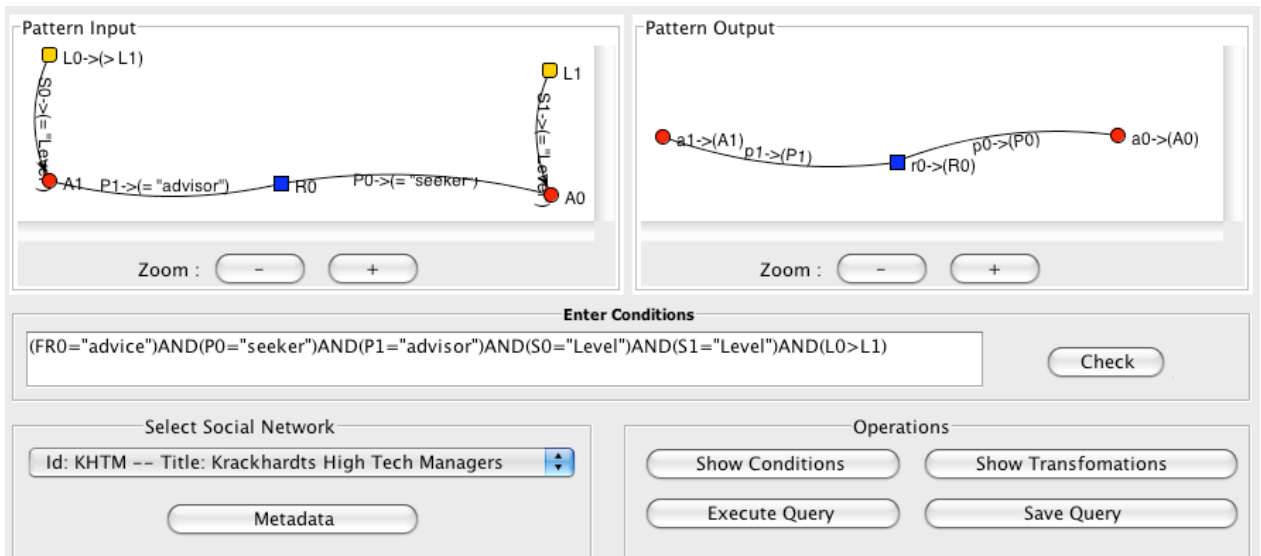


Figure 6.8: Query editor with an SNQM query that selects only advice relation.

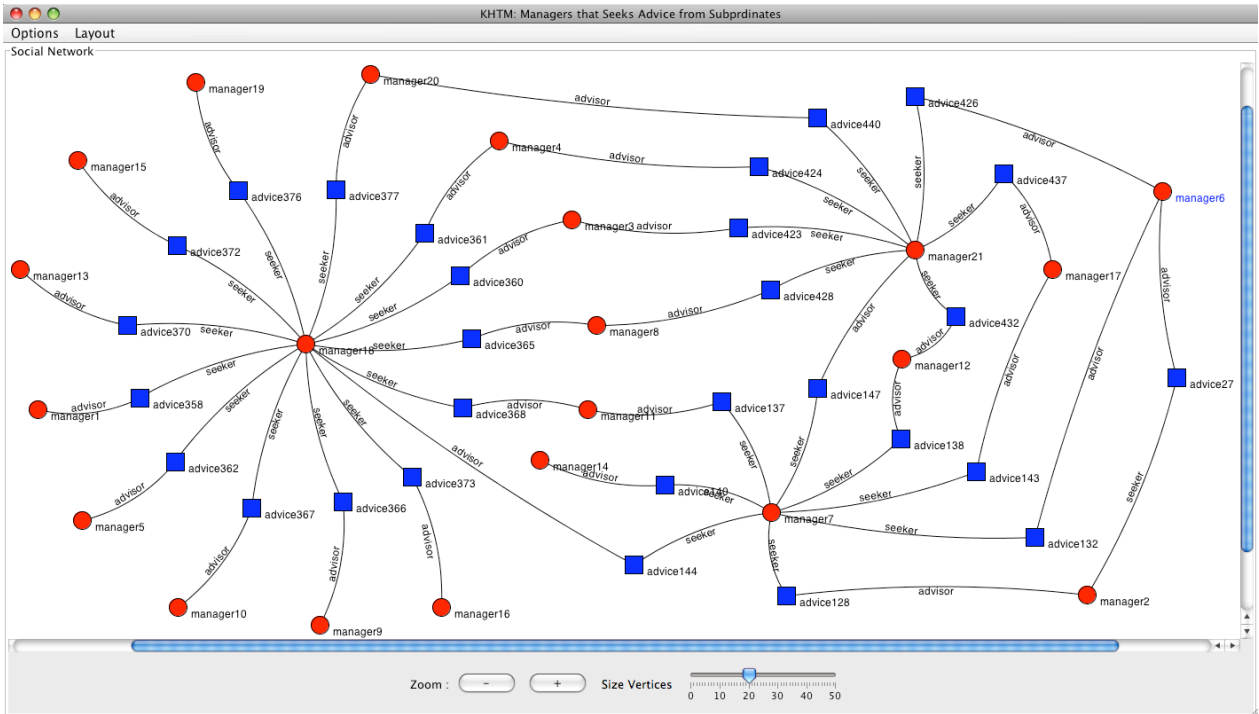


Figure 6.9: Result social network from query in Figure 6.8 over source social network in Figure 6.7.

3. *Compute betweenness centrality for actors in subnetwork advice: This step involves exporting the subnetwork to Pajek, computing centrality in Pajek, and importing back the network including the new data. The Advice subnetwork in Pajek, with betweenness centrality represented as the size of actors, is depicted if Figure 6.10. It is worth noting that when interacting with Pajek it is critical to keep track of actor identifiers because it changes them each time the actors in the network change. Actor 12 has the maximum centrality (0.234); it corresponds to manager18 in the original advice subnetwork.*

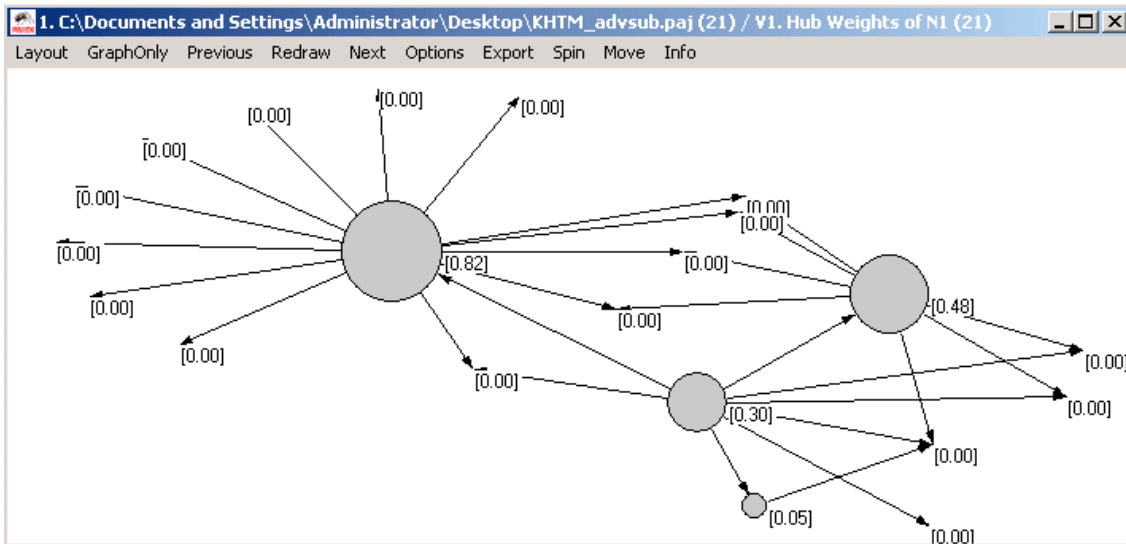


Figure 6.10: Advice subnetwork in Pajek.

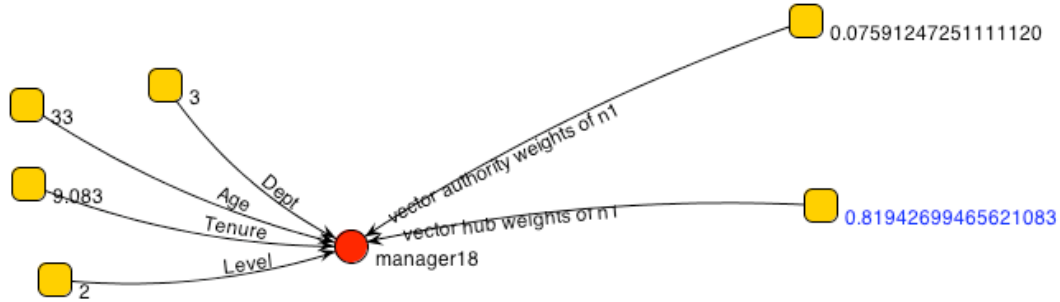


Figure 6.11: Detail of *manager18* in the KHTM network enriched with centrality values computed in Pajek .

4. *Merge advice network with complete network: now actors have an additional attribute corresponding to their centrality in the advice network. Actually, there is another attribute, that could be discarded with a query that recorded the Pajek id that the actor had. See Figure 6.11.*

These use cases stress the fact that, at this scale, the key aspects are: query language expressivity (filters and transformations), visual presentation and editing of networks and queries.

6.2.3 SiRSL

Note that this project did not implement the SNDM; instead, SNDM and SNQL were used as a high-level modeling tools to define queries that later were hardcoded in SQL.

We illustrate the practical use of the data model and its query and transformation language by means of SiRSL¹, a system to search and navigate the catalog of the University’s library. SiRSL integrates in one social network the bibliographical data in the library’s catalog with a record of user activities, in this case, borrowing books.

From this social network, SiRSL produces several user navigable network views rendered on demand from search terms. The main social network is modeled with the SNDM data model and views are produced using SNQL, with translations made to the relational model and SQL. Each data source is mapped to the social network model as straightforwardly as possible: each record produces an isolated actor surrounded by its attributes. The system is organized in modules as follows:

¹<http://sn.dcc.uchile.cl/sirsl>



Figure 6.12: *SiRSL Snapshot*. Main result window of SiRSL showing the result of searching for the term *history* in publication *titles*. All matching titles are shown as a list in the text view (left), and as a graph in the graphical view (right). Double-clicking on a title in the graph adds to the view all neighbors: titles that have been borrowed by a user that also borrowed that title. (Some labels have been translated from Spanish)

Bibliographical Catalog Acquisition

The library catalog is encoded in the Marc21 standard², and is managed with a proprietary system. For the purposes of these experiments, 20% of the catalog was translated to the model: one actor for each publication record, including the existing attributes (title, authors, subjects, etc.). The network, obtained after translation, was then transformed into a multimode network by promoting authors and subjects to actors.

Lending History

For each registered user, one year of lending history was translated to the model: one actor for each user. Attributes include description of the user and her/his list of borrowed publications. Then, publication attributes are promoted to actors with their ids corresponding to the ids of the publication actors in the catalog.

Integrated Social Network Model

Both networks defined above are combined using set theoretical union.

²<http://www.loc.gov/marc/>

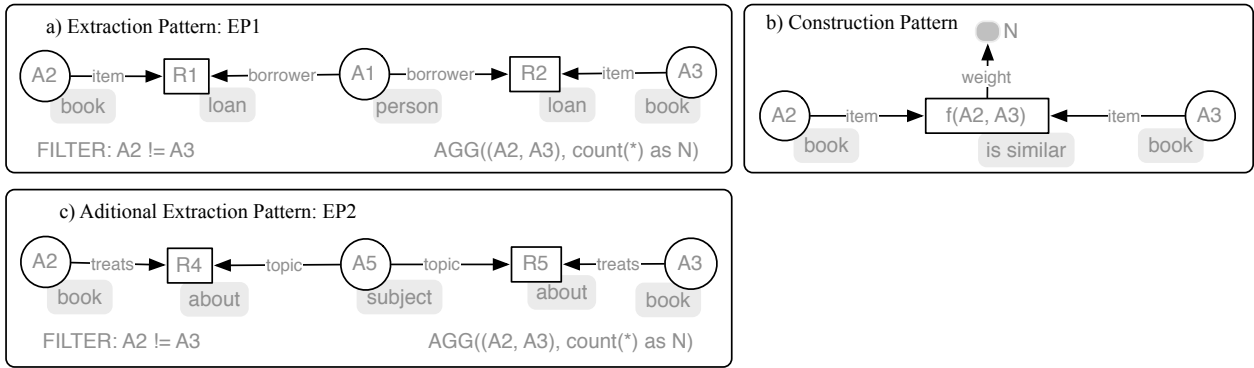


Figure 6.13: *Social Network Schema for Library: Books View*. Extraction pattern (a) matches pairs of books that have been borrowed by the same user. This result is used by the construction pattern (b) to produce a network of related books. A more restrictive view requiring, besides at least one user in common, that two books have a subject in common, can be obtained with the extraction pattern EP1 AND EP2, the conjunction of the patterns (a) and (c).

Network Views

Several materialized network views are built from the network defined above to support different navigational options. We discuss only one: publications are related if they have been borrowed by the same user. This view corresponds to a transformation of the network that makes a path (from one publication to another via a user) to coalesce into a new relation between these two publications. Additionally, the transformation counts the number of different users and attaches the count as an attribute of the relation. This attribute is interpreted as the strength of the relation (see Figure 6.13(a) and (b)).

Noisy relations may appear when, for instance, a student borrows a book on behalf of a friend from another department, thus potentially producing a link between two disparate books. To exclude this relation, there exists an alternate view that takes into account only cases where the two books are also connected through at least one subject (see Fig. 6.13(c)).

Presentation

This module manages the interaction with the user. It is implemented as a web application. The search starts with a term to be found as a title, author, or subject. If the term matches exactly an item in the database, this item and its immediate neighborhood is presented to the user as a graph, and the corresponding details, as text. When the term does not match an item, the term itself is presented in the center of the graph, and all possible matches are presented as its neighbors. In both cases, clicking on a neighbor expands its own neighborhood. The strength of the links included can be controlled by a slider in the upper right corner of the GUI (see Fig. 6.12).

6.3 Preliminary Performance Assessment

To evaluate the performance of the implementation over a RDBMS, we ran some experiments that imported, queried, and transformed a medium size social network: the online bibliographic database on computer science –DBLP.

6.3.1 Experimental Setup

Data

The Digital Bibliography & Library Project (DBLP, <http://dblp.uni-trier.de/>), created by Michael Ley and hosted at Trier University, is a publicly available bibliographic database dedicated to computer science publications. DBLP is organized as one record per publication, where record size may vary depending on the amount of data available for the corresponding publication. DBLP is accessible on-line through several web interfaces, and also as an XML file –`dblp.xml`– that is updated periodically [75].

The version of `dblp.xml` used here (february 2011) weights 822 MBs, and it contains 2.464.449 records.

The first step of the experiment creates 98 incremental subsets from `dblp.xml`, ranging in size from 25.000 to 2.450.000 publication elements, and 16.781.191 attributes, in increments of 25.000.

Each additional batch produces new social networks that are stored in the database for further use (in subsequent batches) and analysis.

Environment

- Hardware: 1 CPUs Intel Xeon E3430 QuadCore 2.4GHz, 8GB RAM, one SATA system disk 500GB, and 3 SATA 500GB disks dedicated to database, two of them as an LVM partition.
- System Software: Ubuntu server 10.10 64bits³, and JVM 1.6.

As data service provider, the experiment reported here uses the RDBMS PostgreSQL version 8.4⁴, under a standard installation with a separated tablespace for datasets residing in a separate set of disks, and `fsync` disabled. This approach requires that:

1. Each social network queried must be previously stored in the RDBMS as three tables: n to declare actors, relations, and their membership to families, r to represent participation roles, and m to represent attributes and their values. To avoid as much as

³<http://www.ubuntu.com/>

⁴<http://www.postgresql.org/>

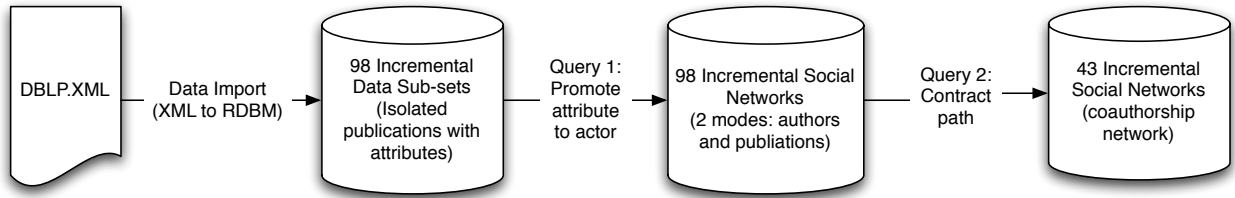


Figure 6.14: *Experiment Sequence*. Data import, attribute promotion query (authors from attributes to actors), path contraction (coauthorship network).

possible interferences between runs, each social network is stored in its own separate set of tables.

2. Each SNDM query executed must be translated to SQL prior to its actual execution inside the RDBMS. We use the automated translation procedure, implemented for SNDMS, to translate each SNDM query to four SQL queries that calculate the resulting social network, and store it into the RDBMS.

Batches

This experiment is comprised by three consecutive batches, where each batch uses data produced by the previous one. See Figure 6.14.

1. Data Import: This stage produced datasets –as groups of three relational tables– in increments of 25.000 publications, from 25.000 to 2.450.000, involving a proportional number of triples. Each data set was translated to the SNDM and imported to a PostgreSQL database; after this stage the only additional data structure in the database are the primary key indexes. Note that, after importing, each record and its attributes remains isolated (R set is empty).
2. Network Connection (Q1 - Author promotion): This batch applies Q1 (see Figure 6.15) to each of the results from the previous batch. Each time, it produced the corresponding network where author attributes were transformed to actors.
3. Path contraction(Q2 - Coauthorship Network): This batch applied Q2 to each of the results from previous batch (Q1). Each time, it produced the corresponding network where publications are replaced by new direct relations between coauthors. Each new relation is labeled with the number of coauthored publications that it represents.

6.3.2 Results

The following are the times for the largest data set processed by each batch:

- Incremental data-sets import: The largest data set inserted had 19.231.191 tuples in the database and took 6.554.469 *ms* (about 109 minutes). Examining the complete series, the tendency is, as expected, clearly linear on the number of tuples. Without indexes the average insertion cost is constant.

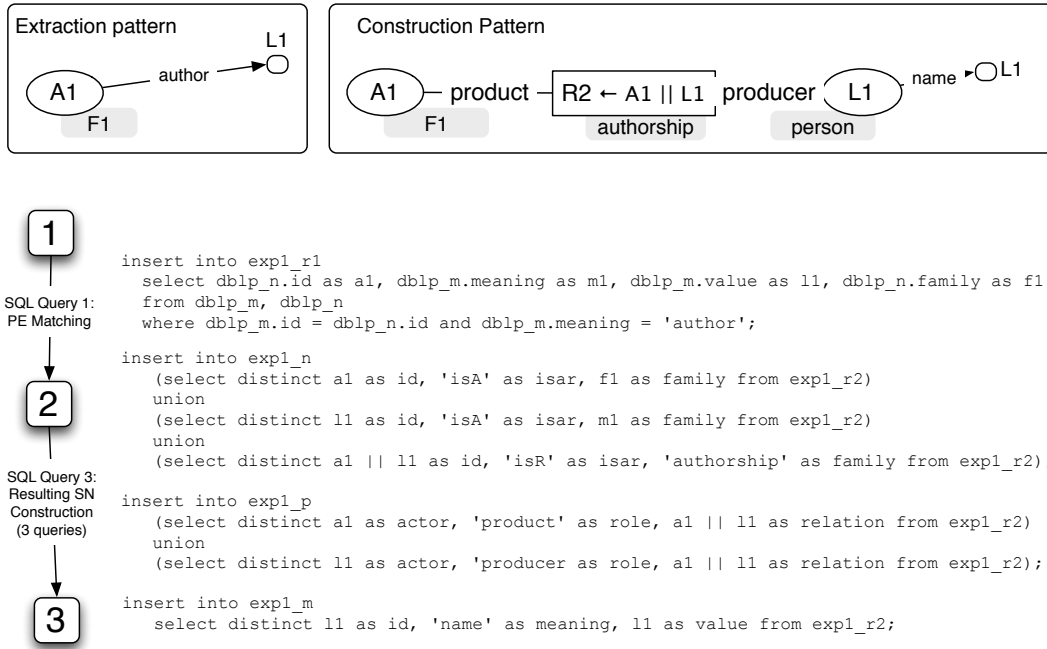


Figure 6.15: *Q1 - Authors Promotion from Attributes to Actors*. Promoting authors to actors connect publications that shares the same author. Figure depicts the patterns in the SNQL query and its four equivalent SQL queries.

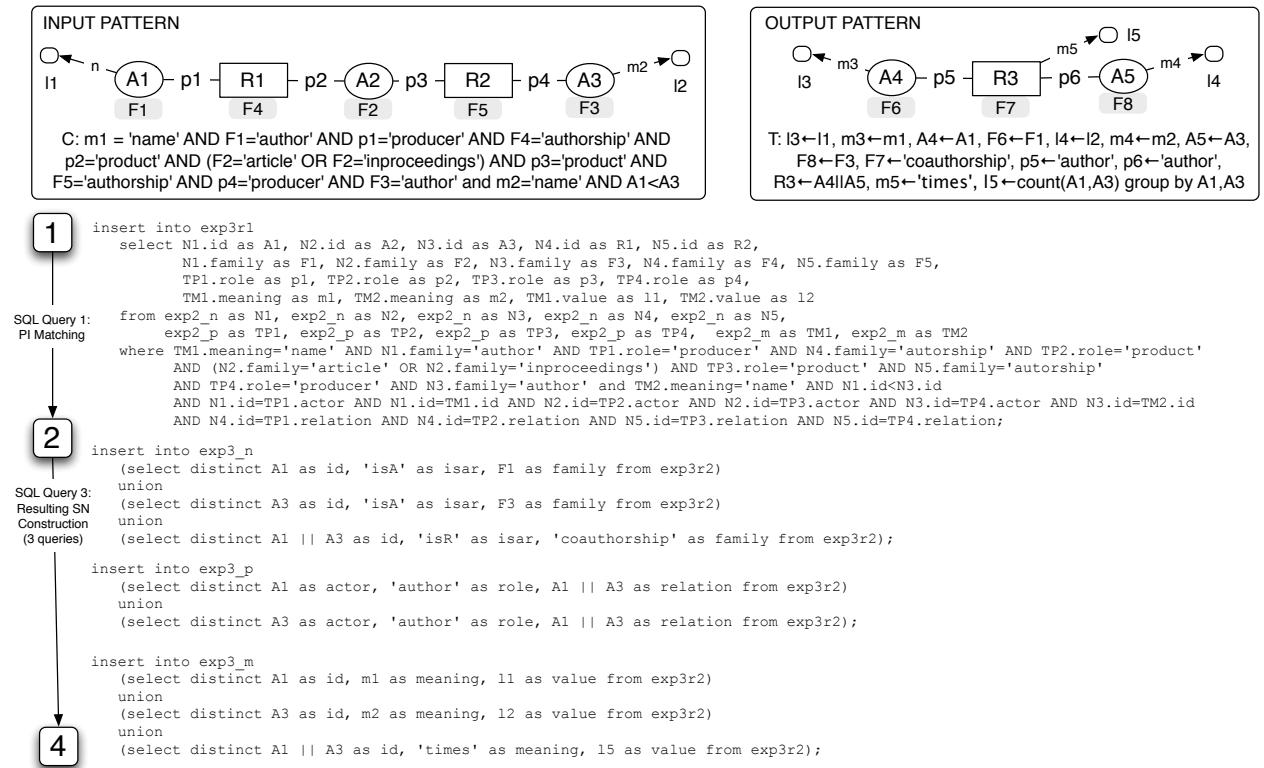


Figure 6.16: *Q2 - Building the coauthorship network*. This query works on the result of the previous one. It contracts paths of length 2 between authors, where there is a coauthored publication. The figure depicts the patterns in the SNQL query and its four equivalent SQL queries.

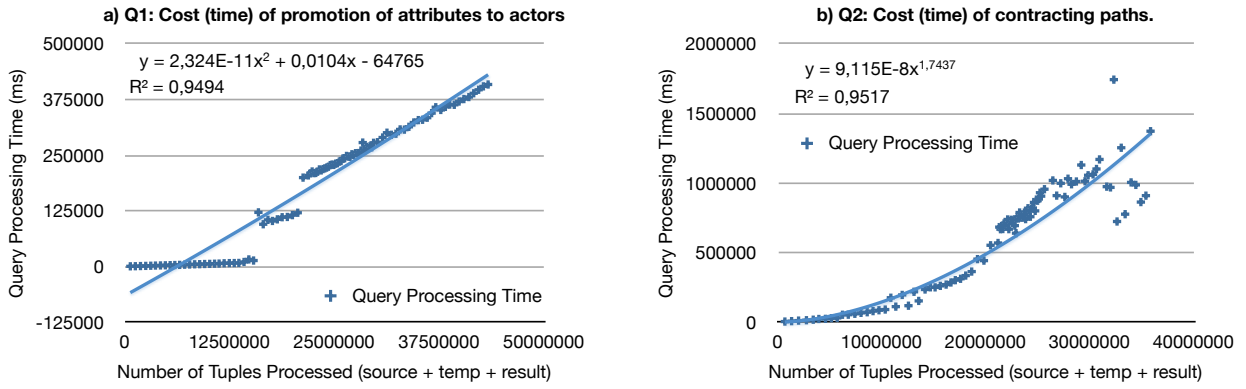


Figure 6.17: *Q1 and Q2: Query Evaluation Times.* Evaluation times in ms for Q1 (a) and Q2 (b) for each subset, per number of processed tuples (input + temp + output).

- Promoting attributes to actors to connect the network (Q1, see Figure 6.15): When applied to the largest network, Q1 processed 43.257.688 of tuples, including intermediate results, and took 408.688 ms (about 7 minutes), see Figure 6.17. The figure shows a polynomial trend.
- Building the coauthorship network (Q2, see Figure 6.16): When applied to the largest network, Q2 processed 35.801.159 of tuples, including intermediate results, and took 1.372.185 ms (about 20 minutes), see Figure 6.17. In this case, even though the total number of tuples processed is smaller, the query itself is more complex: a bigger pattern implies a greater number of joins, and hence a bigger evaluation cost.

6.3.3 Discussion

This experiment assesses the actual performance of the system as it is currently implemented. We think that it shows an acceptable performance in the small size range. Arguably, even at a medium scale (the size of the experiments) the system may offer an alternative to coding each data extraction process. If not in terms of raw performance, the query language may offer advantages: standardization and portability of data sets definitions.

Additionally, it is reasonable to expect sizable performance improvements by implementing the system with specialized algorithms, and not over a different data model.

Conclusions and Future Work

To some extent, the three sections describing the SNDM are the conclusion of our work, since they present its product, discuss its adequacy, and relate it to existing tools and models. On the other hand, our Introduction already listed the publications and graduating projects that have been produced in the context of our work. The concluding remarks in this section, therefore, are mostly written with an eye on the initial thesis proposal, so that the original objectives, expected benefits and hypothesis can be compared to the actual results. Afterwards we discuss the most promising lines for future work, both in the theoretical and in the implementation aspects.

Conclusions

Survey of social network analysis practices. The first task we embarked on was a comprehensive study of SNA data management practices from reference books [50,51,98,110], and from current (at the time) articles published in the Social Networks journal (see Appendix A), and database and data mining conferences (see Section 3.5.3). The results of this study, along with the benefits of data management, were presented to SNA community at the XXVI International Sunbelt conference, the most important forum in the field. There was little awareness of the need and benefits of a well developed data managing infrastructure. Since that date, when still most of the community was working with rather small and simple datasets, the situation has changed to some extent; the number of studies that use big networks has been steadily increasing, and there is a growing activity on data models and data management frameworks for social networks data. This year, for example, the first workshop on databases for social networks, SocialDB, will take place at SIGMOD.

Social networks data management requisites. There are two important use cases that describe data management needs of social networks: data sharing and reuse of big datasets, and data preparation for analysis by a layperson. The first calls for auto-descriptive graph data models, and the second for a standard and scalable language, able to extract and transform social networks but also accessible to users without a specific background in databases.

A model for managing social network data. The main objective of this thesis was to produce a model for the data management of social networks. From the requirements gathered, we produced a data model specification consisting of a graph data structure based on a triple model, and a query and transformation language based on pattern matching and production. Both data instances and query patterns have a text and an equivalent graphical syntax. Another indication of the increasing awareness of the need for a social networks data model is the recent publication of several model proposals [5, 41, 92].

A feasible query language. To be practical, the complexity of the query language must be kept under feasible bounds. This constraint imposes a design trade-off between expressiveness and complexity. We show that our language achieves the needed expressivity with an evaluation cost in NLOGSPACE.

Data management vs. structural analysis. While defining the query language, it became apparent that it was necessary to establish a divide between data management operations and structural analysis operations. This divide is similar to, for instance, the relation between SQL and data mining. A borderline group of operations was the identification of cohesive subgroups, where the subgroup is maximal and it is defined recursively. We finally opted to leave these operations outside of the language, due to their evaluation cost, and also because most SNA tools implement them.

A proof of concept social networks DBMS. From the data model specification, we implemented a proof-of-concept prototype of a data management system for social networks. In this prototype, it is possible to graphically define and issue queries to the database. The result can be visualized and edited. It also implements functions to import and export social networks to and from XML and Pajek.

Generalization of the model. The data structure is capable of representing not only social networks but also a wide range of graphs; thus, it could be used in other domains that require representing graph structures. However, if the structure is not a graph or if the elements are complex objects, other models may be needed, e.g., object oriented databases. The same occurs with the query language. For instance, SNQL would cover only part of the graph operations required for molecular and cell biology, as presented by Eckman and Brown [42].

Hypotheses discussion. Recall the initial hypothesis on which this work is based:

- Social networks are general enough to justify a data model. By “general enough” we mean that they are present as many instances in different domains, representing a big amount of data that needs automatic management methods.
- A specific data model, with a graph database point of view, will improve the support for the automation of data managing in the social networks field. As a result it is

expected that productivity will improve, as it did in the well documented case of the relational data model and its application domain. (The argument is that a data model will let users and programmers access the data through an abstraction layer similar to that used in the application domain, with all the low level data managing hidden by the database management system [28].)

- The common data structure characteristics, and the well defined set of operations over social networks, give design advantages over a generic graph database model, allowing, for instance, the use of algorithms which would not be practical in a more general context.

The first hypothesis states that social networks constitute a proper domain for a specialized database model. Our findings support this hypothesis: social networks are present in varied contexts and are complex data objects. The availability of huge and complex data sets poses the typical data management challenges of extracting and transforming information.

Regarding the second hypothesis, our intuition tells us that it may hold, since the model was designed according to the special needs collected from the application domain. However, only when the model and its implementations mature, and are adopted by a community of users, will we know for sure what the actual benefits are.

With regards to the last hypothesis, the benefits of narrowing the focus of the data model on the social networks domain are manifold. For instance: the modeling is done with a set of data elements that directly represent the objects in the domain, and the operations correspond directly to the tasks performed in the domain. However, without a proprietary implementation of a SN DBMS, it is not yet possible to evaluate the use of customized algorithms.

We feel confident that this thesis reached its original research agenda. We surveyed and characterized the application domain (data management for social networks). Based on these concrete requirements, we build a feasible and appropriate data model. We demonstrate the feasibility of the latter at the theoretical level, studying its complexity, and at the practical level through several implementations. These findings have been published for the social networks analysis and database communities.

At the same time, this research has opened several interesting research questions that we address in the next section.

Future Work

In this section we briefly describe a number of topics relevant for future research. They are organized in two groups: the first is theoretically oriented, while the second focuses on updating and extending the implementations.

Model Improvements

The current specification of the model includes the basic features addressing social networks data management requirements, and serves as a basis for future developments.

Insert, Update and Delete. In this work we did not address the theoretical study of operations to insert, update, and delete elements of a social network. To some extent, it is possible to use the current operations to modify a network, for instance, to change an attribute value or to remove elements. In the implementations we use a naive approach. However, further study is necessary to discern the interactions with the integrity constraints.

Integrity Constraints. In the current version of the model, only the most basic integrity constraints are considered: actor and relations must be in the N set to appear in the R and/or the M set. However, it would be useful to extend and study other plausible constraints, for instance: relation minimum and maximum arity of relations, type of literals, required attributes (not NULL), required participation roles, etc.

Longitudinal Network Data. With continuous data collection, naturally the networks can be seen as a stream of time stamped data. How could the model be extended to provide better support for the analysis of this kind of data? It would be useful, for instance, to facilitate the production of sequences of time slices that properly depict the dynamics of a network.

Additional translations. A good way of characterizing and improving SNDM is through translations to and from other models. For example, the translation to SQL informs us about the relative expressiveness of the model, and the translation to the SocialScope algebraic language shows how well suited SNQL may be to the context of social content sites.

Extend the model to other type of networks. The extension of the model implies the modification of the data structure and/or the query language. For instance: which features of SNQL are useless in biosciences? Which operations should be added to SNQL for it to be useful in biosciences? Would it be possible to keep the language under the same complexity bounds?

Implementations

The current implementation of the SNDMS is functional and serves its purposes in the context of this work. There are, however, many aspects that can be further developed, or added to the model. In fact, the implementation framework was designed to support the continued

extension and improvement of the SNDMS. We list below some interesting features that are not yet implemented.

Performance assessment and practical scalability. The experiments presented in this work show that the implementation behaves within the expected boundaries with a medium sized network. Further experiments are nevertheless needed along at least two axis: variety of queries and networks, and sizes of networks.

Implementation over Datalog/Prolog and a column oriented model. During the course of this work, we developed and tested implementations over the predominant relational data model and over RDF, which is also based on triples. However, there exist at least two other data models worth testing: Datalog and a column oriented model. We use Datalog to provide the extraction semantics of SNQL; thus, the theoretical translation is already defined but the construction must be implemented. Column oriented models are gaining momentum as an alternative to the relational model, as they can offer a great flexibility and are capable of managing triples.

Proprietary database implementation. In the same line of the previous idea, it would be interesting, even if expensive, to implement a proprietary DBMS for the model instead of using a translation into other data models, which taxes every query.

Social networks repository. It could be useful to implement and administer a public SNDM repository of social networks data. The client that remotely issues the queries to extract networks from the repository could be implemented via the Web. A clear issue still to be resolved is the transport strategies of big results from the repository to the clients.

Bibliography

- [1] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufman Ed., 1999.
- [2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [3] Serge Abiteboul, Dallan Quass, Jason Mchugh, Jennifer Widom, and Janet Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1:68–88, 1997.
- [4] Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25:211–230, 2003.
- [5] Sihem Amer-Yahia, Laks V. S. Lakshmanan, and Cong Yu. Socialscope: Enabling information discovery on social content sites. In *CIDR*. www.crdrrdb.org, 2009.
- [6] Sihem Amer-Yahia, Volker Markl, Alon Y. Halevy, AnHai Doan, Gustavo Alonso, Donald Kossmann, and Gerhard Weikum. Databases and web 2.0 panel at vldb 2007. *SIGMOD Record*, 37(1):49–52, 2008.
- [7] Renzo Angles and Claudio Gutiérrez. Querying RDF data from a graph database perspective. In *ESWC*, pages 346–360, 2005.
- [8] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, 2008.
- [9] Gustavo O. Arocena and Alberto O. Mendelzon. Weboql: restructuring documents, databases, and webs. *Theor. Pract. Object Syst.*, 5(3):127–141, August 1999.
- [10] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. Semistructured und structured data in the web: Going back and forth. *SIGMOD Record*, 26(4):16–23, 1997.
- [11] Chris Baerveldt, Marijtje A.J. van Duijn, Lotte Vermeij, and Diane A. van Hemert. Ethnic boundaries and personal choice. assessing the influence of individual inclinations to choose intra-ethnic relationships on pupils’ networks. *Social Networks*, 26:55–74, 2004.
- [12] Albert-László Barabási and Eric Bonabeau. Scale free networks. *Scientific American*,

pages 50–59, May 2003.

- [13] Vladimir Batagelj and A. Mrvar. *Pajek - Analysis and Visualization of Large Networks*, pages 77–103. Springer, 2003.
- [14] Simona Bignami. Network stability in longitudinal data: A case study from rural malawi. *Social Networks*, 27:231–247, 2005.
- [15] H. Blau, N. Immerman, and D. Jensen. A visual language for querying and updating graphs. Computer Science Technical Report 2002-037, University of Massachusetts, Amherst, 2002.
- [16] Phillip Bonacich, Annie Cody Holdren, and Michael Johnston. Hyper-edges and multidimensional centrality. *Social Networks*, 26:189–203, 2004.
- [17] Phillip Bonacich and Paulette Lloyd. Calculating status with negative relations. *Social Networks*, 26:331–338, 2004.
- [18] Katy Börner. Plug-and-play macroscopes. *Commun. ACM*, 54:60–69, March 2011.
- [19] Ulrik Brandes and Thomas Erlebach, editors. *Network Analysis: Methodological Foundations [outcome of a Dagstuhl seminar, 13-16 April 2004]*, volume 3418 of *Lecture Notes in Computer Science*. Springer, 2005.
- [20] Ronald Breiger, Kathleen Carley, and Philippa Pattison, editors. *Dynamic Social Network Modeling and Analysis*. The National Academies Press, 2003.
- [21] Ronald L. Breiger. The analysis of social networks. In Melissa Hardy and Alan Bryman, editors, *Handbook of Data Analysis*, pages 505–526. Sage Publications, 2004.
- [22] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. *SIGMOD Rec.*, 25(2):505–516, June 1996.
- [23] Carter T. Butts. The complexity of social networks: theoretical and empirical findings. *Social Networks*, 23:31–71, 2001.
- [24] Carter T. Butts. network: A package for managing relational data in R. *Journal of Statistical Software*, 24(2):1–36, 2 2008.
- [25] Kathleen M. Carley. Linking capabilities to needs. In Ronald Breiger, Kathleen M. Carley, and Philippa Pattison, editors, *Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers*, pages 363–370. The National Academies Press, 2003.
- [26] Peter J. Carrington, John Scott, and Stanley Wasserman, editors. *Models and Methods in Social Network Analysis*, volume 27 of *Structural Analysis in the Social Sciences*. Cambridge, 2005.
- [27] E. F. Codd. Data models in database management. In *Workshop on Data abstraction*,

- databases and conceptual modeling*, pages 112–115, 1980.
- [28] E. F. Codd. Relational database: A practical foundation for productivity. *Communications of the ACM*, 25(2):109–117, February 1982.
- [29] Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science and Engineering*, pages 29–41, July/August 2009.
- [30] Mariano P. Consens and Alberto O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *PODS*, pages 404–416. ACM Press, 1990.
- [31] Mariano P. Consens and Alberto O. Mendelzon. Low complexity aggregation in graphlog and datalog. In Serge Abiteboul and Paris C. Kanellakis, editors, *ICDT*, volume 470 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 1990.
- [32] Elizabeth Costenbader and Thomas W. Valente. The stability of centrality measures when networks are sampled. *Social Networks*, 25:283–307, 2003.
- [33] Jonathon N. Cummings and Rob Cross. Structural properties of work groups and their consequences for performance. *Social Networks*, 25:197–210, 2003.
- [34] Jonathon N. Cummings and Monica C. Higgins. Relational instability at the network core: Support dynamics in developmental networks. *Social Networks*, 28:38–55, 2006.
- [35] Wouter de Nooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory Social Network Analysis with Pajek*. Cambridge University Press, 2005.
- [36] Alain Degenne and Marie-Odile Lebeaux. The dynamics of personal networks at the time of entry into adult life. *Social Networks*, 27:337–358, 2005.
- [37] Peter Sheridan Dodds, Roby Muhamad, and Duncan J. Watts. An experimental study of search in global social networks. *Science*, 301:827–829, August 2003.
- [38] Patrick Doreian, Vladimir Batagelj, and Anuska Ferligoj. An experimental study of search in global social networks. *Social Networks*, 26:29–53, January 2004.
- [39] Patrick Doreian, Vladimir Batagelj, and Anuska Ferligoj. Generalized blockmodeling of two-mode network data. *Social Networks*, 26:29–53, 2004.
- [40] S. N. Dorogovtsev and J. F. F. Mendes. *Evolution of Networks: From Biological Nets to the Internet and WWW*. Oxford University Press, 2003.
- [41] Anton Dries, Siegfried Nijssen, and Luc De Raedt. A query language for analyzing networks. In David Wai-Lok Cheung, Il-Yeol Song, Wesley W. Chu, Xiaohua Hu, and Jimmy J. Lin, editors, *CIKM*, pages 485–494. ACM, 2009.
- [42] B. A. Eckman and P. G. Brown. Graph data management for molecular and cell biology. *IBM J. Res. Dev.*, 50:545–560, November 2006.
- [43] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, 3rd*

Edition. Addison Wesley Longman, 2000.

- [44] Guillaume Erétéo et al. A state of the art on social network analysis and its applications on a semantic web. In *SDoW2008 at ISWC 2008*, 2008.
- [45] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.
- [46] Katherine Faust, Karin E. Willert, David D. Rowlee, and John Skvoretz. Scaling and statistical models for affiliation networks: patterns of participation among soviet politicians during the brezhnev era. *Social Networks*, 24:231–259, 2002.
- [47] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for a web-site management system. *SIGMOD Rec.*, 26(3):4–11, September 1997.
- [48] Tim Finin, Li Ding, and Lina Zou. Social networking on the semantic web. *Learning Organization Journal*, Ubiquitous Business Intelligence, 2005.
- [49] Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database techniques for the world-wide web: a survey. *SIGMOD Rec.*, 27(3):59–74, September 1998.
- [50] Linton C. Freeman. *The Development of Social Network Analysis*. Empirical Press, 2004.
- [51] Linton C. Freeman, A. Kimball Romney, and eds. Douglas R. White. *Research Methods in Social Network Analysis*. Transaction Publishers, 1992.
- [52] Tim Furche, François Bry, Sebastian Schaffert, Renzo Orsini, Ian Horrocks, Michael Krauss, and Oliver Bolzer. Survey over Existing Query and Transformation Languages, 2004.
- [53] Jim Gray, David T. Liu, María A. Nieto-Santisteban, Alex Szalay, David J. DeWitt, and Gerd Heber. Scientific data management in the coming decade. *SIGMOD Record*, 34(4):34–41, 2005.
- [54] R. Guting. Graphdb: modeling and querying graphs in databases. In *20th VLDB Conference*, pages 297–308, 1994.
- [55] Mark S. Handcock, David R. Hunter, Carter T. Butts, Steven M. Goodreau, and Martina Morris. statnet: Software tools for the representation, visualization, analysis and simulation of network data. *Journal of Statistical Software*, 24(1):1–11, 2 2008.
- [56] Robert A. Hanneman and Mark Riddle. *Introduction to social network methods*. University of California, Riverside, 1994.
- [57] Steve Harris and Andy Seaborne. Sparql 1.1 query language, w3c working draft 1 june 2010. *W3C website*, 2010.

- [58] Tony Hey, Stewart Tansley, and Kristin Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 1.1 edition, 2009.
- [59] Mark Huisman and Marijtje A. J. van Duijn. Software for social network analysis. *Models and Methods in Social Network Analysis*, March 2005.
- [60] Mark Huisman and Marijtje A. J. van Duijn. Software for statistical analysis of social network. In *Sixth International Conference on Logic and Methodology*, 2005.
- [61] Takashi Ito, Tomoko Chiba, Ritsuko Ozawa, Mikio Yoshida, Masahira Hattori, and Yoshiyuki Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *PNAS*, 98(8):4569–4574, April 2001.
- [62] H. V. Jagadish and Frank Olken, editors. *Data Management for the Biosciences: Report of the NSF/NLM Workshop on Data Management for Molecular and Cell Biology at the National Library of Medicine February 2-3, 2003.*, November 2003.
- [63] H. V. Jagadish and Frank Olken. Database Management for Life Science Research: Summary Report of the Workshop on Data Management for Molecular and Cell Biology at the National Library of Medicine, Bethesda, Maryland, February 2-3, 2003. *OMICS*, 7(1):131–137, 2003.
- [64] David Jensen and Jeniffer Neville. Data mining in social networks. In *Dynamic Social Network Modeling and Analysis*, pages 289–302, 2003.
- [65] Emily M. Jin, Michelle Girvan, and M. E. J. Newman. Collective dynamics of ‘small-world’ networks. *Nature*, 393:440–442, June 1998.
- [66] Jason J. Jung and Jérôme Euzenat. Towards semantic social networks. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *ESWC*, volume 4519 of *LNCS*, pages 267–280. Springer, 2007.
- [67] Irit Katriel and Ulrich Meyer. Elementary graph algorithms in external memory. In *Algorithms for Memory Hierarchies*, pages 62–84, 2002.
- [68] Stephan Kepser. A Simple Proof for the Turing-Completeness of XSLT and XQuery. In *Proceedings of the Extreme Markup Languages 2004 Conference*, Montréal, Quebec, Canada, 2004.
- [69] Peter D. Killworth, Christopher McCarty, H. Russel Bernard, Eugene C. Johnsen, John Domini, and Gene A. Shelley. Two interpretations of reports of knowledge of subpopulation sizes. *Social Networks*, 25:141–160, 2003.
- [70] David Konopnicki and Oded Shmueli. W3qs: A query system for the world-wide web. In *Proceedings of the 21th International Conference on Very Large Data Bases, VLDB ’95*, pages 54–65, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
- [71] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. Schemasql: An extension to sql for multidatabase interoperability. *ACM Trans. Database Syst.*,

26(4):476–519, December 2001.

- [72] Rance P. L. Lee, Danching Ruan, and Gina Lai. Social structure and support networks in beijing and hong kong. *Social Networks*, 27:249–274, 2005.
- [73] Mark Levene and George Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer-Verlag, London, UK, 1999.
- [74] Kevin Lewis, Jason Kaufman, Marco Gonzalez, Andreas Wimmer, and Nicholas Christakis. Tastes, ties, and time: A new social network dataset using facebook.com. *Social Networks*, 30(4):330 – 342, 2008.
- [75] Michael Ley. The DBLP computer science bibliography: Evolution, research issues, perspectives. In Alberto H. F. Laender and Arlindo L. Oliveira, editors, *SPIRE*, volume 2476 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002.
- [76] William S. Lovejoy and Christoph H. Loch. Minimal and maximal characteristic path lengths in connected sociomatrices. *Social Networks*, 25:333–347, 2003.
- [77] Alexandra Marin. Are respondents more likely to list alters with certain characteristics?: Implications for name generator data. *Social Networks*, 26:289–307, 2004.
- [78] Peter V. Marsden. Interviewer effects in measuring network size using a single name generator. *Social Networks*, 25:1–16, 2003.
- [79] Uwe Matzat. Academic communication and internet discussion groups: transfer of information or creation of social contacts? *Social Networks*, 26:221–255, 2004.
- [80] A.O. Mendelzon, G.A. Mihaila, and T. Milo. Querying the world wide web. In *Parallel and Distributed Information Systems, 1996., Fourth International Conference on*, pages 80 –91, dec 1996.
- [81] Peter Mika. *Social Networks and the Semantic Web*, volume 5 of *Semantic Web And Beyond Computing for Human Experience*. Springer, 2007.
- [82] M. E. J. Newman. The structure of scientific collaboration networks. *PNAS*, 98(2):404–409, January 2001.
- [83] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.
- [84] M. E. J. Newman and Juyong Park. Why social networks are different from other types of networks. *arXiv*, 393, May 2003.
- [85] M.E.J. Newman. Ego-centered networks and the ripple effect. *Social Networks*, 25:83–95, 2003.
- [86] M.E.J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27:39–54, 2005.

- [87] Jukka Nyblom, Steve Borgatti, Juha Roslakka, and Mikko A. Salo. Statistical analysis of network data –an application to diffusion of innovation. *Social Networks*, 25:175–195, 2003.
- [88] Rasmus Pagh. Basic external memory data structures. In *Algorithms for Memory Hierarchies*, pages 14–35, 2002.
- [89] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3), 2009.
- [90] María Guadalupe Ramírez Ortiz, José Ramiro Caballero Hoyos, and María Guadalupe Ramírez López. The social networks of academic performance in a student context of poverty in mexico. *Social Networks*, 26:175–188, 2004.
- [91] H. Rheingold. *The virtual community: homesteading on the electronic frontier*. Number no. 28 in *The Virtual Community: Homesteading on the Electronic Frontier*. MIT Press, 2000.
- [92] Royi Ronen and Oded Shmueli. Soql: A language for querying and creating data in social networks. In *ICDE*, pages 1595–1602. IEEE, 2009.
- [93] Nicolas Ruffin, Helmar Burkhart, and Sven Rizzotti. Social-data storage-systems. In Denilson Barbosa, Gerome Miklau, and Cong Yu, editors, *DBSocial*, pages 7–12. ACM, 2011.
- [94] Mauro San Martín and Claudio Gutierrez. Personal management of social networks data. In *CSE (4)*, pages 765–770. IEEE Computer Society, 2009.
- [95] Mauro San Martín and Claudio Gutierrez. Representing, querying and transforming social networks with RDF/SPARQL. *ESCW2009*, 2009.
- [96] Mauro San Martín, Claudio Gutierrez, and Peter T. Wood. SNQL: A social networks query and transformation language. In Pablo Barceló and Val Tannen, editors, *AMW*, volume 749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
- [97] Peter Sanders. Memory hierarchies - models and lower bounds. In *Algorithms for Memory Hierarchies*, pages 1–13, 2002.
- [98] John Scott. *Social Network Analysis*. SAGE Publications, second edition, 2000.
- [99] Lisa Singh, Mitchell Beard, Lise Getoor, and M. Brian Blake. Visual mining of multi-modal social networks at different abstraction levels. In *IV*, pages 672–679. IEEE Computer Society, 2007.
- [100] Susie M. Stephens, Johan Rung, and Xavier Lopez. Graph Data Representation in Oracle Database 10g: Case Studies in Life Sciences. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2004.
- [101] Michael Szell and Stefan Thurner. Measuring social dynamics in a massive multiplayer

- online game. *Social Networks*, 32(4):313 – 329, 2010.
- [102] Thodoros Topaloglou, Susan B. Davidson, H. V. Jagadish, Victor M. Markowitz, Evan W. Steeg, and Mike Tyers. Biological data management: Research, practice and opportunities. In *VLDB*, pages 1233–1236, 2004.
- [103] Maksim Tsvetovat, Jana Diesner, and Kathleen M. Carley. Netintel: A database for manipulation of rich social network data. *CMU-ISRI-04-135*, March 2005.
- [104] Maksim Tsvetovat, Jeff Reminga, and Kathleen M. Carley. Dynetml: Interchange format for rich social network data. *CMU-ISRI-04-105*, 2004.
- [105] Sergi Valverde and Ricard V. Solé. Network motifs in computational graphs: A case study in software architecture. *Santa Fe Institute Working Paper 2005-04-008*, August 2005.
- [106] IJ. Hetty van Emmerick. Gender differences in the creation of different types of social capital: A multilevel study. *Social Networks*, 28:24–37, 2006.
- [107] Frits van Merode, Anna Nieboer, Hans Maarse, and Harm Lieverdink. Analyzing the dynamics in multilateral negotiations. *Social Networks*, 26:141–154, 2004.
- [108] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146. ACM, 1982.
- [109] A. Vázquez, R. Dobrin, D. Sergi, J.-P. Eckmann, Z. N. Oltvai, and A.-L. Barabási. The topological relationship between the large-scale attributes and local interaction patterns of complex networks. *PNAS*, 101(52):17940–17945, December 2004.
- [110] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, first edition, 1994.
- [111] Duncan J. Watts, Peter Sheridan Dodds, and M. E. J. Newman. Identity and search in social networks. *Science*, 296:1302–1305, May 2002.
- [112] Scott White and Padhraic Smyth. Algorithms for estimating relative importance in networks. In *ACM SIG Knowledge Discovery and Data Mining proceedings*, pages 266–275, August 2003.
- [113] Barbara Zemljic and Valentina Hlebec. Reliability of measures of centrality and prominence. *Social Networks*, 27:73–88, 2005.

Appendix A

SNA Current Practices Study: Summary Tables

Table A.1: List of articles included in our study sample and their activities for each SNA data workflow stage.

ID	Ref.	General Data Description	Data Collection	Storage and Manipulation	Analysis
1	[106]	Network of faculty members, career and social capital related	Mail survey	Ego-centered network, 4 dimensions, free recall, max 5	Gender differences in social capital creation
2	[34]	Network of MBA students, career related	Group meeting survey and email survey	Ego-centered networks, unbounded number of alters	Relational stability
3	[36]	Networks around french young people	Longitudinal survey	Egocentered networks in three periods, unbounded number of alters	Personal networks in the transition to adult life
4	[14]	Egocentric informal conversational networks in rural Malawi	Household panel survey (name generator question), with reinterview, on informal conversational networks on family planning and AIDS.	Ego-centered conversational networks for 2 topics and role-relations.	Network Stability
5	[72]	Social support networks in two cities in China	Fixed choice survey of person's choices of social support over two samples (one from each city)	Ego-centered networks, 5 support dimensions and 8 categories of role-relations.	Bivariate porcentaje table, logistic regression analyses
6	[86]	Four datasets from previous research	One synthetic data set, three Existent Datasets: Potterat (2002), Padgett and Ansell (1993), and Newman and Park (2003).	One-mode networks	Random walk centrality compared to others

Continued on next page

Table A.1 – continued from previous page

ID	Ref.	General Data Description	Data Collection	Storage and Manipulation	Analysis
7	[113]	Social support network of high school students	Interviews	Ego-centered network, 4 support dimensions, recognition and free recall	Reliability of measures of centrality and prominence
8	[77]	Egocentric conversational network based on the 1985 General Social Survey (GSS)	Web-administered survey based on the name generator of the 1985 GSS, enhanced to promote recalling.	Ego-centered conversational networks, 1 topic, unbounded number of alters	Likelihood of alters to be recalled, given their properties.
9	[17]	Relations between monks in a monastery (Sampson 1968)	Existent data set	SAMPLK3 and SAM-PDLK (Sampson 1968) data sets contained in UCINET.Liking and disliking networks.	Centrality measure that take into account positive and negative relationships.
10	[16]	The data describe 56 attacks on European settlements by the Caribe in 29 different years	Existent data set	Hypergraph (incidence matrix: events x islands and years)	Supra-dyadic centrality.
11	[79]	Internet Discussion Groups (IDG) involving a sample of dutch and english researchers	Postal questionnaire sent to a random sample of English and Dutch university researchers on 8 disciplines	IDG use by researchers and communication between them.	Multiple linear regression, multiple logistic regression. (To test hypothesis regarding IDG benefits)
12	[107]	The process of determining prices for specialized medical care in The Netherlands	From a previous study (Lieverdink, 1999)	Two-mode network: agents and events	Dynamic quantitative analysis

Continued on next page

Table A.1 – continued from previous page

ID	Ref.	General Data Description	Data Collection	Storage and Manipulation	Analysis
13	[90]	Network of junior high school from a low socioeconomic status	Self-administered sociometric questionnaire	Ego-centered conversational networks, 1 topic , unbounded number of alters	Association between centrality measurements and academic performance
14	[39]	Supreme Court Voting, Southern Women, Journal-to-Journal citation	Existent data sets (Doriean 2003, Davis 1941, Baker 1992)	Two and one-mode networks	Generalized blockmodelling
15	[11]	Inter-ethnic networks of pupils in schools	Existent data set (Dutch Social Behavior Study)	One-mode multinet-work of students	Impact of ethnicity in relation development
16	[32]	Eight datasets from previous research	Existent data sets	Sociometric networks that differ in their size, the number and type of questions, and the number of nominations allowed	Eleven centrality measures
17	[76]	Star and clique-plus-paths synthetic networks	Synthetic networks	Random networks generation	Lower and upper bound for characteristic path lengths
18	[33]	Workgroups in a Fortune 500 telecommunications firm	Email survey	One-mode multinet-works	Impact of workgroup structure on performance
19	[4]	Networks of individuals' homepages	Automatically gathered information from homepage In-links, Out-links and Text, and from Mailing lists.	Automated construction of a one-mode network	N.A.

Continued on next page

Table A.1 – continued from previous page

ID	Ref.	General Data Description	Data Collection	Storage and Manipulation	Analysis
20	[69]	Three datasets from telephone surveys about subpopulations	Existent data set	Ego-centered networks	Subpopulations size
21	[87]	Diffusion of organic farming	Existent data set	N.R.	Confounding covariates for adoption, adoption process over time
22	[78]	Egocentric network from the 1998 General Social Survey (GSS)	Existent data set	Ego-centered network	Name generator effect on network size
23	[85]	Coauthorship networks in mathematics and biomedicine	Existent data sets	2 co-authorship networks	Number of second neighbors of the average vertex
24	[46]	Coattendance of soviet politicians to official and social events during 8 years of Brezhnev era	From CIA publication (1973-1980)	Affiliation network (politicians and events). Only events attended by at least two politicians.	Scaling models and random graphs models.

Table A.2: Summary of the study of the current SNA data workflow.

ID	Data Collection/Storage							Analysis		
	#Nets	Total Nodes	Total Rels	Network Types	Unit of Observation	Data Reuse	Time	Network Types	Unit of Analysis	Longitudinal
1	838	N.R.	N.R.	Ego-centered	Faculty members	No	N.R.	Ego-centered	Ego-networks	No
2	77	1054	977	Ego-centered	MBA students	No	3 periods	Ego-centered	Ego-networks	Yes
3	66	3152	3086	Ego-centered	Persons	No	6 years	Ego-centered	Persons	Yes

Continued on next page

Table A.2 – continued from previous page

ID	Data Collection/Storage							Analysis		
	#Nets	T. N.	T. R.	N. Types	U. of Obs	Reuse	Time	N. Types	U. of A.	Long.
4	3	2588	N.R.	Ego-centered	Persons	Yes	3 periods	Ego-centered	Persons	Yes
5	2	2124	N.R.	Ego-centered	Persons	No	1 year	Ego-centered	Persons	No
6	4	N.A.	N.R.	One-mode	Dataset dependent	Yes	N.A.	One-mode	Dataset dependent	No
7	8	N.R.	N.R.	Ego-centered	Students	No	1 month	Ego-centered, One-mode	Students	No
8	1	332	N.R.	Ego-centered	Students	No	N.R.	Ego-centered	Students	No
9	1	18	N.R.	One-mode	Monks	Yes	1 period	One-mode	Monks	No
10	1	51	56	Two-Mode	Attacks	Yes	91 years	Two-Mode	Attacks, years	Yes (*)
11	1	1063	N.R.	N.R.	Researchers	No	6 months	N.R.	N.R.	No
12	1	463	N.R.	Two-mode	Agents, Events	Yes	6 years	Two-mode, One-mode	Agents, events	No
13	12	523	N.R.	Ego-centered	Students	No	3 months	One-mode	Students, Classes, Whole net	No
14	3	85	N.R.	Two-mode, one-mode	Justices and Decisions, Women and Events, Journals	Yes	1 year, N.R., N.R.	Two-mode, one-mode	3 two mode nets	No
15	20	1317	N.R.	Ego-centered	Students	Yes	N.R.	One-mode, Ego-centered	Students, groups	No

Continued on next page

Table A.2 – continued from previous page

ID	Data Collection/Storage							Analysis		
	#Nets	T. N.	T. R.	N. Types	U. of Obs	Reuse	Time	N. Types	U. of A.	Long.
16	59	N.A.	N.R.	Ego-centered	Dataset dependent	Yes	Dataset dependent	One-mode	Dataset dependent	No
17	4×10^6	N.A.	5 - 35	One-mode	N.A.	No	N.R.	One-mode	Networks	No
18	182	957	N.R.	Ego-centered	Engineers	No	2 years	Ego-centered	Workgroups, Events	No
19	2	9775	N.R.	One-mode	Students	No	N.R.	N.A.	N.A.	No
20	3	1521	N.R.	Ego-centered	Persons	Yes	N.R.	Ego-centered	Subpopulations	No
21	1	N.R.	N.R.	Ego-centered	Farms	Yes	10 years	Ego-centered, One-mode	Farms	Yes
22	1	1445	N.R.	Ego-centered	Persons	Yes	N.R.	Ego-centered	Networks	No
23	2	1773590	N.R.	One-mode	Author	Yes	10 years	Ego-centered	Coauthor of Coathor	No
24	1	1883	14900	Two-mode non-dyadic affiliation	Politicians and events	Yes	8 years	Two-mode affiliation, one-mode	Politicians and events	Yes

Appendix B

SocialScope Translation to SNQL

In this chapter we provide a translation to SNQL for each SocialScope [5] query. Each translation provides an equivalent result by giving a straightforward mapping between data structures.

We first present the mapping between data structures, and then we provide the translation for each operation.

B.1 Data Structures

In SocialScope the social content graph is represented as an abstract graph where actors are nodes and relations are links (arcs). Both types of objects –nodes and links– have identity, attributes, and keywords. Attributes may have multiple values. The keywords set is a list of values associated to a given the object.

B.1.1 Comparison of Data Elements

First, it is worth mentioning that the SocialScope’s data structure is defined as abstract, consequently authors do not address practical lower-level details or implementation issues. For instance, it is not defined how attributes are stored and related to actors and relations.

Actors. In both models actors have identity and they are modeled as nodes in the graph.

Relations. In both models relations have identity. However, in SocialScope, relations are modeled as arcs in the social content graph, while in SNDM relations are modeled as nodes. This is the main difference between the data structures: in SocialScope all relations are considered in principle binary, while SNDM does not assume a fixed arity for relationships.

Furthermore, in SNDM the arity of relations may change. It is worth noting that, provided that relations have their own id in SocialScope, it would be straightforward to represent relations with bigger arities.

Attributes. In both models, actors and relations can have attributes: pairs of labels and collections of values. Each attribute may have one or more values. In SocialScope there exists a special collection of values called *keywords*. In SNDM these *keywords* can be implemented as another multivalued attribute. For each object (actor or relations), its attributes are identified with a label and a set of associated values.

B.1.2 Mapping between Data Structures

The translations presented below assume the following mapping between a SNDM social network S and a SocialScope social content graph G :

Actors. Actor nodes in G are mapped to actors nodes in S . SNDM supports having attribute triples $-m(a, b, c_i), i : 1..k-$ with the same meaning labels b and different values c_i for the same object a ; consequently a single attribute meaning may be associated to several values. Attributes are pairwise mapped, and actor keywords in G are mapped to a single keyword attribute called *keywords* for each corresponding actor in S . In terms of triples an actor $u \in nodes(G)$, with attributes $u.att_i, i : 1..k$, and keywords, is represented in S as:

```
{
  n(u.id, isa, type1), ..., n(u.id, isa, typeN),
  m(u.id, u.att1, value11), ..., m(u.id, u.att1, value1j_1),
  ... ,
  m(u.id, u.attk, valuek1), ..., m(u.id, u.attk, valuekj_k),
  m(u.id, u.keywords, kw1), ..., m(u.id, u.keywords, kwm)
}
```

Relations. Relation arcs (links) in G are mapped to relation nodes in S . Their participating actors are associated to these relation nodes through participation roles (arcs with labels source or target), attributes are mapped to SNDM attributes, and all keywords are mapped to a single SNDM attribute called *keywords*. In terms of triples, a relation $l \in links(G)$, connecting actor u to actor v ($u, v \in nodes(G)$), with attributes $l.att_i, i : 1..k$, and keywords, is represented in SNDM as:

```
{
  n(l.id, isr, type1), ..., n(l.id, isr, typeN),
  m(l.id, l.att1, value11), ..., m(l.id, l.att1, value1J_1),
  ... ,
  m(l.id, l.attK, valueK1), ..., m(l.id, l.attk, valueKJ_k),
  m(l.id, l.keywords, kw1), ..., m(l.id, l.keywords, kwM),
  r(u.id, source, l.id), r(v.id, target, l.id)
}
```

B.2 Translations to SNQL

B.2.1 Unary Operators

There are two unary operators in SocialScope: node selection ($\Sigma_{\langle C, S \rangle}^N$), and link selection ($\Sigma_{\langle C, S \rangle}^L$). In both cases, an object –node (actor) or link (binary relation)– appears in the result if and only if it satisfies condition C , which consists of a list of pairs of attribute names and values, and an optional set of keywords. An object o satisfies C when, for each condition $att = val_1, \dots, val_k$, the set of o 's values for att is a superset of values $\{val_1, \dots, val_k\}$. Scoring function S is used to compute a score for each object based on how well its content matches the keywords provided in C . If no scoring function is provided, but C provides keywords, a default function is used.

In SNQL there are currently two options to refer to a set/group of values associated to a given attribute: through grouping plus aggregation, and –if the set of values is discrete and finite (which should always be the case given the definition of C)– through patterns in a subquery. We use the latter in the translations below.

Node Selection

Node selection outputs a null graph consisting of nodes but no links, including their attributes and the calculated score.

Definition 28 (Node Selection) $\Sigma_{\langle C, S \rangle}^N(G) = \{v, v.score = S(v) | v \in nodes(G) \wedge v \text{ satisfies } C\}$

To avoid confusion, in the following translation the selected objects are called actors instead of nodes.

Translation to SNQL. The expected result can be achieved with two queries: one to identify all the actors that have all required values for each of the named attributes (Q1), and one to induce the complete result (each actor, its attributes, and score) from the set of identified actors (Q2).

Note that each SocialScope's structural condition is translated as a local pattern of attributes around an actor V : a collection of m -triples of the form (V, att_i, x_{ij_i}) , where variable V matches a node that for each att_i has every value x_{ij_i} , with $i : 1..n$ and each $j_i : 1..k_i$.

```
Q1:
CONSTRUCT {n(V, isa, F)} AS G'
WHERE      {n(V, isa, F),
           m(V, att_1, x_11), ... , m(V, att1, x_1k_1),
           ...
           m(V, att_n, X_n1), ... , m(V, attk, x_nk_n)}
FROM      G
```

In the second query Q2, all actors identified by Q1 and their attributes are collected. Keywords (translated as an additional multivalued attribute) are collected and used to compute S .

Q2:

```

CONSTRUCT {n(V,isa,F), m(V, M, L)}
WHERE      {n(V, isa, F)} MATCH G'
           AND
           {m(V, M, L)} MATCH G
FROM       G, G'

UNION

CONSTRUCT {m(V, score, L)}
WHERE      {n(V, isa, F)} MATCH G'
           AND
           AGG((V, keyword), S(X) as L, {m(V, keyword, X)}) MATCH G
FROM       G, G'

```

Link Selection

Link selection outputs a subgraph induced by those links that satisfy C , including their incident nodes, attributes, and the score computed from the keywords.

Definition 29 (Link Selection) $\Sigma_{\langle C, S \rangle}^L(G) = \{l, l.score = S(l) | l \in links(G) \wedge l \text{ satisfies } C\}$

To avoid confusion, in the following translation the selected objects are called relations instead of links.

Translation to SNQL. Given that in SNDM relations are –like actors– also nodes, the translation is similar to the actor (node) selection case, except for the induced pattern which is more complex.

The expected result can be achieved with two queries: one to identify all relations that have all required values for each of the named attributes (Q1), and one to induce the complete result (each relation, its two incident actors, its attributes, and score) from the set of previously identified relations (Q2).

Note that each SocialScope’s structural condition is translated as a local pattern of attributes around a relation V : a collection of m -triples of the form $(V, att_i, x_{i j_i})$, where variable V match a node (a relation in this case) that for each att_i has every value $x_{i j_i}$, with $i : 1..n$ and each $j_i : 1..k_i$.

```

Q1:
CONSTRUCT {n(V, isr, F)} AS G'
WHERE      {n(V, isr, F),
            m(V, att_1, x_11), ... , m(V, att1, x_1k_1),
            ...
            m(V, att_n, X_n1), ... , m(V, attk, x_nk_n)}
FROM      G

```

In the second query Q2 all relations identified by Q1, their incident actors and their attributes are collected. Keywords (translated as an additional multivalued attribute) are collected and used to compute S .

```

Q2:
CONSTRUCT {n(V, isr, F), m(V, M, L), r(A1, P1, V), r(A2, P2, V),
            n(A1, isa, F1), m(A1, M1, L1),
            n(A2, isa, F2), m(A2, M2, L2)}
WHERE      {n(V, isr, F)} MATCH G'
AND
            {n(V, isr, F), m(V, M, L), r(A1, P1, V), r(A2, P2, V),
            n(A1, isa, F1), m(A1, M1, L1),
            n(A2, isa, F2), m(A2, M2, L2)} MATCH G
FROM      G, G'

UNION

CONSTRUCT {m(V, score, L)}
WHERE      {n(V, isa, F)} MATCH G'
AND
            AGG((V, keyword), S(X) as L, {m(V, keyword, X)}) MATCH G
FROM      G, G'

```

B.2.2 Basic Binary Operators

In this category, SocialScope defines three set-theoretic operators: union, intersection, and set difference (link- and node-driven).

Set-Theoretic Operators

Definition 30 (SocialScope Set-Theoretic Operators) *Let G_1 and G_2 be two social content graphs originated from the same social content site. $G_1 \cup G_2$, $G_1 \cap G_2$, and $G_1 \setminus G_2$ are defined as: $nodes(G_1 \oplus G_2) = nodes(G_1) \oplus nodes(G_2)$ and $links(G_1 \oplus G_2) = links(G_1) \oplus links(G_2)$, where \oplus is one of \cup , \cap , \setminus , and nodes and links with the same id are consolidated in the output graph.*

SocialScope models social networks, representing actors as nodes and relations as links, a modeling choice that in principle restricts to two the arity of relations. Attributes are

encapsulated in the object that they describe –they are not part of the explicit structure of the graph– and authors do not specify what happens to them under these operations. For the sake of clarity, we assume here the simplest scenario, where for objects present in both G_1 and G_2 their attributes are identical (the set of attributes and the set of values for each one). However, it is worth mentioning that we are also interested in the general case where this restriction does not hold, for instance when consolidating or comparing two different moments of a given network.

Even though nodes and links have their own id, it seems reasonable to assume, in the context given by SocialScope, that the following holds:

- The scope of ids is universal.
- For each relation(link) l between nodes u and v , there exists a f.d. $\text{id}(l) \rightarrow \text{id}(u), \text{id}(v)$.
- For all actors $u \in G_1$ and $v \in G_2$ if $\text{id}(u) = \text{id}(v) \Rightarrow u \equiv v$ (which includes their attributes and keywords)
- For all relations $u \in G_1$ and $v \in G_2$ if $\text{id}(u) = \text{id}(v) \Rightarrow u \equiv v$ (which includes their attributes and keywords)

Note that we use the notation for triples $t(a, b, c), t \in \{n, r, m\}$ only to improve readability of SNQL queries, but it does not make different types of triples incompatible under set-theoretic operators: the actual distinction among types of triples comes from the partition of the set of predicate labels b . An alternate notation that stresses this point is $(a, t:b, c), t \in \{n, r, m\}$.

Translations to SNQL.

Each set-theoretic operation requires a different SNQL translation.

Definition 31 (SNQL Set-Theoretic Union) *Let $G_1 = (N_1, R_1, M_1)$ and $G_2 = (N_2, R_2, M_2)$ be two social networks originated from the same social content site (this is not a requirement of the union itself). The union $G_1 \cup G_2 = (N_1 \cup N_2, R_1 \cup R_2, M_1 \cup M_2)$.*

```

CONSTRUCT {(A, B, C)}
WHERE      {(A, B, C)} MATCHES G_1
           OR
           {(A, B, C)} MATCHES G_2
FROM       G_1, G_2

```

The notation $G_1 \text{ UNION } G_2$ refers to this definition.

Definition 32 (SNQL Set-Theoretic Union) *Let $G_1 = (N_1, R_1, M_1)$ and $G_2 = (N_2, R_2, M_2)$ be two social networks originated from the same social content site (this is not a requirement of the intersection itself). The intersection $G_1 \cap G_2 = (N_1 \cap N_2, R_1 \cap R_2, M_1 \cap M_2)$.*

```

CONSTRUCT {(A, B, C)}
WHERE      {(A, B, C)} MATCHES G_1
           AND
           {(A, B, C)} MATCHES G_2
FROM       G_1, G_2

```


Note that in the general case the resulting social network may not hold the same constraints that holds for G_1 and G_2 , such as the minimum arity of a family of relations.

In SocialScope there are two forms of set theoretic-minus: node-driven minus and link-driven minus. In both cases, the usual difference is computed between one of the sets (nodes or links), and the other is induced from the difference and the first operand.

Definition 33 (SocialScope Set-Theoretic Node-Driven Minus) *Let G_1 and G_2 be two social content graphs originated from the same social content site. $G_1 \setminus G_2$ is defined as: $\text{nodes}(G_1 \setminus G_2) = \text{nodes}(G_1) \setminus \text{nodes}(G_2)$, and $\text{links}(G_1 \setminus G_2)$ consists precisely of those links from G_1 induced by the nodes in $\text{nodes}(G_1 \setminus G_2)$.*

Definition 34 (SNQL Set-Theoretic Node-Driven Minus) *The result can be produced with two queries: one to identify the actors in G_1 and not in G_2 (Q1), and another to induce the complete result (each pair of distinct actors, their attributes, and the relations between them) from the set of previously identified actors (Q2).*

```
Q1:
CONSTRUCT {n(V, isa, F)} AS G'
WHERE      {n(V, isa, F)} MATCH G_1
           AND-NOT
           {n(V, isa, F)} MATCH G_2
FROM      G_1, G_2
```

In the second query Q2 all relations between the identified actors are induced.

```
Q2:
CONSTRUCT {n(R, isr, F), m(R, M, L), r(U, P1, R1), r(V, P2, R),
           n(U, isa, F1), m(U, M1, L1),
           n(V, isa, F2), m(V, M2, L2)}
WHERE     ({n(U, isa, F1), n(V, isa, F2)} FILTER U!=V MATCH G')
           AND
           ({n(R, isr, F), m(R, M, L), r(U, P1, R), r(V, P2, R),
           n(U, isa, F1), m(U, M1, L1),
           n(V, isa, F2), m(V, M2, L2)} MATCH G_1)
FROM      G_1, G'
```

Definition 35 (SocialScope Set-Theoretic Link-Driven Minus) *Let G_1 and G_2 be two social content graphs originated from the same social content site. $G_1 \setminus G_2$ is defined as: $\text{links}(G_1 \setminus G_2) = \text{links}(G_1) \setminus \text{links}(G_2)$, and $\text{nodes}(G_1 \setminus G_2)$ consists precisely of those nodes from G_1 induced by the links in $\text{links}(G_1 \setminus G_2)$.*

Definition 36 (SNQL Set-Theoretic Link-Driven Minus) *The result can be produced with two queries: one to identify the relations in G_1 and not in G_2 (Q1), and another to induce the complete result (each pair of distinct actors, their attributes, and the relations between them) from the set of previously identified relations (Q2).*

```

Q1:
CONSTRUCT {n(R, isr, F)} AS G'
WHERE      {n(R, isr, F)} MATCH G_1
           AND-NOT
           {n(R, isr, F)} MATCH G_2
FROM      G_1, G_2

```

In the second query Q2 all the actors participating in the identified relations are induced.

```

Q2:
CONSTRUCT {n(R,isr,F), m(R, M, L), r(U,P1,R), r(V, P2,R),
           n(U,isa,F1), m(U, M1, L1),
           n(V,isa,F2), m(V, M2, L2)}
WHERE      ({n(R, isr, F)} MATCH G')
           AND
           ({n(R,isr,F), m(R, M, L), r(U,P1,R), r(V, P2,R),
            n(U,isa,F1), m(U, M1, L1),
            n(V,isa,F2), m(V, M2, L2)} FILTER U!=V MATCH G_1)
FROM      G_1, G'

```

B.2.3 Advanced Binary Operators

SocialScope defines two additional binary operations that allow merging and filtering of social content graphs: composition and semi-join.

Both of these operations use a directional conditional δ to determine in which direction a link is matched. The directional condition $\delta = (d_1, d_2)$ consists of two link directions: $d_i = src|tgt, i : 1..2$. Consider, for instance, the composition of two links $l_1 \in G_1$ and $l_2 \in G_2$ and the directional condition (src, tgt) , l_1 is composed with l_2 if and only if the source node of l_1 matches target node of l_2 (that is $l_1.src.id = l_2.tgt.id$). Note that $\delta_{\bar{d}_i}$ indicates the opposite direction of δ_{d_i} .

Under the stated data structures mapping (see section B.1.2) the arc between a relation node and a participating actor node is labeled according to the direction of the link in SocialScope: from **source** to **target**. These participation role labels may be used to implement directional conditions.

Composition

The composition operation use the function \mathcal{F} to produce new links merging those links from two input social content graphs that match δ .

Function \mathcal{F} must be in class **CF**:

- A composition function must accept as input two groups of attributes (and their values) corresponding to the two input links. These attributes may be link attributes or node

attributes.

- A composition function must produce as output a group of uniquely named attributes (and their values) to be associated with the output link.

Definition 37 (Composition) *Let G_1 and G_2 be two social content graphs originated from the same social content site. $G = G_1 \odot_{\langle \delta, \mathcal{F} \rangle} G_2$, where \mathcal{F} is a function in the class **CF**, is defined as:*

- $\forall u, v, l [u, v \in nodes(G), l \in links(G) \iff \exists l_1 \in links(G_1), l_2 \in links(G_2) \text{ s.t.}$
 $u = l_1.\delta_{d_1} \wedge v = l_2.\delta_{d_2} \wedge l_1.\delta_{d_1} = l_2.\delta_{d_2} \wedge l.src = u \wedge l.tgt = v].$
- $l.\{att1, att2, \dots\} = \mathcal{F}(l_1, l_2)$

Translation to SNQL. The SNQL version of composition requires definitions for δ and \mathcal{F} . Given a directional condition $\delta = (d_1, d_2)$, and the relations $l_1 \in G_1$ and $l_2 \in G_2$, the following extraction pattern is equivalent to δ :

```
{r(U, -d1, L1), r(W, d1, L1)} MATCH G_1
```

AND

```
{r(V, -d2, L2), r(W, d2, L2)} MATCH G_2
```

On the other hand, to emulate \mathcal{F} is more complex. In fact, it is not possible to compute all outputs (each value of each attribute produced for the new relation) in a single SNQL query. Instead, \mathcal{F} is modeled as a collection of aggregate functions f_i . For each $f_i \in \mathcal{F}$ a SNQL query Q_i is defined, and the final result is: $\bigcup_i Q_i$.

To keep the query as simple as possible u and v attributes are not included. They could be added to the query itself, or later by means of another query plus set union.

Q_i :

```
CONSTRUCT {r(U,src,L), r(V,tgt,L), (L,isr,F), (L,M,A)}
           IF L=g(L1,L2) AND F=f(F1,F2), AND M=h(M1,M2)
WHERE     AGG((U,V,L1,L2), f_i(M1,M2,A1,A2,...) as A),
           {n(U,isa,F3),r(U, -d1,L1),r(W,d1,L1),n(L1,isr,F1),m(L1,M1,A1)} MATCH G_1
           AND
           {n(V,isa,F4),r(V, -d2,L2),r(W,d2,L2),n(L2,isr,F2),m(L2,M2,A2)} MATCH G_2
           )
FROM      G_1, G_2
```

Semi-Join

Semi-Join operator produces a subgraph of G_1 induced by the links in G_1 that match links in G_2 , where matching is defined as be incident in the same node under a directional condition.

Definition 38 (Semi-Join) *Let G_1 and G_2 be two social content graphs originated from the*

same social content site. $G = G_1 \times_{\delta} G_2$, where δ is a directional condition (see composition), is defined as:

- $\forall l[l_{src}, l_{tgt} \in nodes(G), l \in links(G)] \iff l \in links(G_1) \wedge \exists l_2 \in links(G_2) s.t. l \cdot \delta_{d1} = l_2 \cdot \delta_{d2}$.

Translation to SNQL. For this operation it is required to select each relation l and its two participating actors in G_1 such that there exists in G_2 a relation that shares an actor w with l given δ .

This result can be achieved with two queries. The first one to identify all relations l that satisfy the condition:

```
Q1:
CONSTRUCT {n(L, isr, F)} AS G'
WHERE      {n(L, isr, F), r(U, -d1, L), r(W, d1, L)} MATCH G_1
          AND
          {n(L2, isr, F), r(V, -d2, L2), r(W, d2, L2)} MATCH G_2
FROM      G_1, G_2
```

And the second query to induce the actors and attributes from G_1 for each l in G' .

```
Q2:
CONSTRUCT {n(L, isr, F), m(L, M, A), r(U, P1, L), r(V, P2, L),
          n(U, isa, F1), m(U, M1, A1),
          n(V, isa, F2), m(V, M2, A2)}
WHERE      {n(L, isr, F)} MATCH G'
          AND
          {n(L, isr, F), m(L, M, A), r(U, P1, L), r(V, P2, L),
          n(U, isa, F1), m(U, M1, A1),
          n(V, isa, F2), m(V, M2, A2)} MATCH G
FROM      G, G'
```

B.2.4 Aggregation Operators

SocialScope defines two aggregation operators (node based aggregation and link aggregation), and two related classes of aggregation functions (set and numerical aggregation) to create and store new aggregated data into social content graphs.

Aggregation Functions

An aggregation function takes as input a collection of links (and their associated attributes and values) and produces as output a value to be associated with the attribute `att`. Set aggregation functions map a set of links to a set of scalars, and numerical aggregation functions map a set of links to a numerical scalar value. In the following definitions, $\$x$ denotes

a variable. When **att** is a set-valued attribute of link l , the expression $l.att = \$x$ binds $\$x$ to one value of $l.att$ at a time.

Definition 39 (Set Aggregate Functions) *Let L be a set of links. An aggregation function \mathcal{A} is in class **SAF** (set aggregate functions) if and only if it is of the form $\{\$x | l \in L \text{ and } l.att = \$x\}$, which extracts values of the attribute **att** from every link in the input set L and forms an output set of scalar values.*

Translation to SNQL. Set aggregate functions collect in a new attribute all values of the given set of attributes. For instance, for a set of relations **tagging** between a user and all products he or she has tagged, where each relations has an attribute **tag**. A set aggregate function can collect all the values of each tag attribute in a single new attribute. In SNDM, multivalued attributes are represented as multiple m-triples with the same meaning and different values. Consequently, a SNQL query Q just need to produce for each value collected an m-triple with the label of the new attribute. If only a subset of the attributes of the link should be included, a condition over attributes labels (**M**) should be included using the **FILTER** option.

```
Q:
CONSTRUCT { ... , m(L, new_att, A), ... }
WHERE      {{..., m(L, M, A), ...} FILTER (C(M))} MATCH G
FROM      G
```

Definition 40 (Numerical Aggregate Functions) *The class **NAF** of aggregation functions is defined as follows:*

- *Every arithmetic operation $+$, $-$, \times , \div is in **NAF**.*
- *The constant functions **0** and **1** which map every scalar input to be constant 0 and 1 respectively are in **NAF**.*
- *Summation over a collection, i.e., $\sum_{x \in X} f(x)$, where X is a collection and f is a function in **NAF**, is in turn in **NAF**.*
- *Product over a collection, i.e., $\prod_{x \in X} f(x)$, where X is a collection and f is a function in **NAF**, is in turn in **NAF**.*
- ***NAF** is closed under composition.*

Translation to SNQL. Numerical aggregate functions (on a single level of aggregation) are equivalent to the aggregate functions \mathcal{A} used in SNQL extraction patterns.

```
...
WHERE      ...
            AGG(GroupingVariables, A(...), {Pattern})
...
```

Node Aggregation

Node aggregation actually aggregates links around nodes and produces a new attribute for each node whose value (or set of values) are computed with a function in **NAF** \cup **SAF**.

Definition 41 (Node Aggregation) *Let G be a social content graph, $\gamma_{\langle C, d, \text{textttatt}, A \rangle}^N(G)$ produces a social content graph G' that is isomorphic to G and $\forall v \in G' \exists l \in G \wedge l$ satisfies $C \wedge l.d = v$, then $v.\text{att} = \mathcal{A}(\{l_i \in \text{links}(G) | l_i \text{ satisfies } C \wedge l_i.d = v\})$.*

The directionality parameter d acts as a group-by attribute, in that all outgoing links from a node (or all incoming links to a node) are grouped together and aggregated. Condition C constraints which links are actually aggregated, for instance **type** = 'friend'.

Translation to SNQL. As discussed above, strategies for set aggregation and numerical aggregation are completely different. In the case of set aggregation, a new attribute is produced where all values collected are assigned.

For the clarity of presentation the construction pattern in the following query **Q** includes only the triples for the new attribute produced, which can be merged in the graph G later using set theoretical union.

```
Q:
CONSTRUCT {n(V, isa, F), m(V, new_att, A)}
WHERE      {n(V, isa, F), r{V, P, L}, m(L, M, A)} FILTER (P=d AND C(M))} MATCH G
FROM       G
```

In the case of numerical aggregation the AGG extraction pattern is used.

```
Q:
CONSTRUCT {n(V, isa, F), m(V, new_att, A)}
WHERE      AGG({V, F}, f(X) as A,
               {n(V, isa, F), r{V, P, L}, m(L, M, X)} FILTER (P=d AND C(M)))}
FROM       G
```

Link Aggregation

Link aggregation summarizes links between each pair of nodes, and replaces them with a new link with attributes functionally generated from the attributes of all involved links.

Definition 42 (Link Aggregation) *Let G be a social content graph, $\gamma_{\langle C, \text{textttatt}, A \rangle}^L(G)$ produces a social content graph G' as follows:*

1. Partition $\{l | l \in \text{links}(G) \wedge l \text{ satisfies } C\}$ on $l.\text{src}$ and $l.\text{tgt}$;
2. For each set of links $\mathcal{L}_{s,t}$ sharing the same source node s and the same target node t , replace $\mathcal{L}_{s,t}$ with a new link $l_{s,t}$;

3. Attach an attribute `att` with value $l_{s,t}$, with its value computed as $\mathcal{A}(\mathcal{L}_{s,t})$.

Translation to SNQL. The following query Q produces a new relation and a new attribute for the relation containing all the values of all attributes of selected relations.

```
Q:
CONSTRUCT {n(U, isa, F1), n(V, isa, F2), n(L1, isr, F), m(L1, new_att, A)}
           r{U,P1,L1}, r{V,P2,L1}} IF L1=g(U,V)
WHERE     {n(U, isa, F1), n(V, isa, F2), n(L, isr, F),
           r{U,P1,L}, r{V,P2,L), m(L, M, A)} FILTER (P1=d AND C(M))} MATCH G
FROM      G
```

In the case of numerical aggregation the AGG extraction pattern is used.

```
Q:
CONSTRUCT {n(U, isa, F1), n(V, isa, F2), n(L1, isr, F), m(L1, new_att, A)}
           r{U,P1,L1}, r{V,P2,L1}} IF L1=g(U,V)
WHERE     AGG({L, F, U, F1, V, F2}, f(X) as A,
           {n(U, isa, F1), n(V, isa, F2), n(L, isr, F),
           r{U,P1,L}, r{V,P2,L), m(L, M, X)} FILTER (P1=d AND C(M))} MATCH G
FROM      G
```