A Semantic Graph Model

Liu Chen(✉), Ting Yu, and Mengchi Liu

State Key Lab of Software Engineering, School of Computer,
Wuhan University, Wuhan, China
{dollychan,yuting}@whu.edu.cn, mengchi@scs.carleton.ca

Abstract. Graph models do excel where data have an element of uncertainty or unpredictability and the relationships are data's main features. However, existing graph models neglect the semantics of node and relationship type.

To capture as much semantics as possible, we extend the nodes in graph model with some object-oriented features and edges with multiple semantic information, and propose a Semantic Graph Model (SGM). SGM is a schema-less model and supports dynamic data structures as well as extra semantics. Although the class definition is unknown at the beginning, the schema can be extracted from the semi-structured and semantic data. The excavated domain model can help further data analysis and data fusion, and it is also important for graph query optimization.

We have proposed graph create statements to represent data in SGM and have implemented a conversion layer to store, manage and query the graph upon the graph database system, Neo4j.

1 Introduction

Due to the increasing requirement of query and analysis on linked data, such as social networking, master data management, geospatial, recommendations, web RDF data, etc., graph databases have attracted a lot of interests for their ability to represent connections [1,2]. A graph consists of a set of nodes and a set of edges (or relationships) that connect them. Real world entities are represented as nodes in a graph and the ways in which those entities are related in the world as relationships. This general-purpose, expressive structure allows graph model to represent all kind of scenarios.

A consensus schema is hard to define when integrating data from the Web for data expressions as structures are uncertain and unpredictable [3,4]. For example, from the Wikipedia the structures of countries are quite different, China makes up of provinces, USA comprises states and UK includes four countries. This is an example that data from one source may have different structures, as well as data from different sources and describing different aspects of information.

This work is supported by National Natural Science Funds of China under grand numbers 61202100 and 61272110.

C. Debruyne et al. (Eds.): OTM 2015 Conferences, LNCS 9415, pp. 378–396, 2015.
DOI: 10.1007/978-3-319-26148-5_25

Schema-less graph models are suitable here to represent data from different sources with various structures.

Semantics are important for analysis and schema-less applications, and people try to capture as much semantics as possible. To our knowledge, there are two kinds of semantic data, one is the well-defined schema or pre-defined conceptual knowledge base established before instance data are created; the other is semantics lies in instance data. For example, "consists_of" is a containment relationship, which is the former kind; China includes Province Hubei and this connection has containment meaning, which is the latter kind.

The former semantic data need lots of work to conclude and integrate. Existing semantic graph models, such as RDF, SKOS and OWL, belong to this kind[5–7]. RDF can represent conceptual and reasoning rules in a triple form, while it doesn't consider the relationships between relationships. Simple Knowledge Organization System (SKOS) is an RDF vocabulary and supports semantic relationships.

For the second kind of semantic data, the extra semantics can be captured, represented and used with extracted data. Some semantics of heterogeneous data are left over and are not captured and leveraged via existing graph models, which is the task we aim at. These semantics are mainly types of nodes and relationships, and there is no way to identify and classify important entities and relationships in a graph. In the country example given above, China, USA and UK are countries, and the relationships between provinces and China, states and USA, and countries and UK are containment connections.

The more semantics we capture, the more knowledge we can use [8–11]. Additionally, the semantics of nodes and relationships are necessary for graph query optimization [12–14]. Hence, we extend the nodes in graph model with some object-oriented features and edges with multiple semantic information, and we call this extension the Semantic Graph Model (SGM).

In SGM, every node is treated as an object and objects are classified into classes. Objects have various relationships with each other. Based on the data we extract and integrate from the Web, we attach three kinds of semantics to relationships: relationship type (including containment, role and user-defined type), relationship hierarchy and inverse relationship. Some relationships are unidirectional, some are bidirectional, and some directed relationships are pairwise. Pairwise relationships have a strong dependency on each other and they are fundamentally one relationships between two objects with two different labels for two directions.

In this way, SGM can represent as much semantics as possible in a schema-less manner. What's more, due to the semantics captured and semi-structure of data, the structures of classes can be extracted from data and the domain model is obtained, and from the model more knowledge can be excavated and applies to further integrate and fuse heterogeneous data. Also, the semantics and structures of the graph are applied to query optimization.

This paper is organized as follows: Section 2 gives an overview of SGM, Section 3 and Section 4 respectively elaborate the extension of nodes and relationships, Section 5 introduces the implementation of SGM upon Neo4j [15] and finally Section 6 concludes the paper.

2 Overview

2.1 Our Model

SGM is extended based on property graph model, in which every node and relationship can have properties. Nodes are objects and relationships are connections between objects.

Every node (object) is an instance of one or multiple classes. If the class information of an object is unknown, it can be omitted. In order to represent the complex properties of real-world entities, properties of objects in SGM can have multiple value (array) and aggregation of properties. Relationships are named and directed, and always have a start and end node. In SGM, we introduce three features to relationships: relationship type (containment, role, user-defined type), relationship hierarchy and inverse relationship. Hence, an extended node EN is the expansion of a graph node N and a semantic edge SE is the extension of graph edge E in graph model G = (N, E).

1. Every relationship has a label indicates its natural meaning that is the relationship name, such as friend, parent and etc. Relationship type refers to the extra semantics of the connection, like the two pre-defined types in SGM, containment and role.
   Relationships with composition semantics can be represented as containment relationships, such as relationship "consist_of", "has_a" and etc. Relationships of a type share common inference rules. General connections omit the interdependent relationship that objects to some degree depend on the object containing them. Furthermore, containment relationship is transitive such that if A contains B, B contains C then A contains C.
   Real world entities have various natural and complex properties and relationships with each other and via these relationships, objects play various roles that form their context, and then have the corresponding context-dependent properties. Even though relationships are first-class citizens in graph models, existing ones oversimplify and ignore the complex relationships and context-dependent properties. To model the dynamic aspect of an object, we introduce role relationships in SGM. An object can play the same role in different organizations or circumstances, as well as different roles in the same context. In specified context and as particular role, entity can have context-dependent properties and relationships.
   Users can classify relationships into ad hoc relationship types for different applications.
2. In some cases, the relationships between objects are exhibited in a hierarchical way from a general relationship to a more specific one. Hence, in SGM, the hierarchies of a relationships are kept.
3. In addition, there are uniliteral and mutual relationships, and as in graph relationships are directed, to represent mutual relationships, relationships in two directions are defined respectively in existing models, while in this way, the correspondence information between this two relationships is lost.

SGM supports the definition of mutual relationships in pairs to maintain their correspondence.

SGM is a schema-less and semi-structured model just like graph model; that is, there is no need to predefine schema for nodes and relationships in advance. However, with data represented in SGM, we can excavate the class definition from the object data, and generate the model graph. The extracted model summarizes and displays the structure and semantics of the object data and meaningful knowledge can be mined from it.

Also, with the knowledge of the data, graph query optimization can proceed.

2.2 Introduction of Neo4j

To manage and store data represented by SGM in a database system, we select Neo4j as the database system, transform the SGM data into structures that Neo4j can process, and at the same time keep the extra semantic information we capture in SGM. We manage and analyze data mainly extracted from the Web by taking advantage of Neo4j.

As in [16], the latest version of Neo4j released in May 2015, Neo4j is a labeled property graph making up of nodes, relationships, properties and labels. In Neo4j, Nodes contain properties. Nodes store properties in the form of arbitrary key-value pairs. In Neo4j, the keys are strings and the values are the Java string and primitive data types, plus arrays of these types. Nodes can be tagged with one or more labels. Labels group nodes together, and indicate the roles they play within the dataset. Relationships connect nodes and structure the graph. A relationship always has a direction, a single name, and a start node and an end node - there are no dangling relationships. Together, a relationship's direction and name add semantic clarity to the structuring of nodes. Like nodes, relationships can also have properties. The ability to add properties to relationships is particularly useful for providing additional metadata for graph algorithms, adding additional semantics to relationships (including quality and weight), and for constraining queries at runtime.

Cypher, the language Neo4j uses to create and query graphs, is concise and easy to use. Neo4j is a full-fledged native graph database system and achieves a good data access efficiency. We take use of features of Neo4j to keep and store data as well as their semantics. The semantics SGM capture is information of nodes and relationships, and a easy way to maintain the information is taking advantage of properties of nodes and relationships.

3 Extension of Nodes

As SGM is a schema-less model, and the properties and relationships in an object are not predefined and vary from objects to objects of a class. Hence, except the object-oriented features related to definitions of classes, like encapsulation and polymorphism, SGM extends graph model with objects and classes, and composition and inheritance.

Although the structure of a class is not known in SGM, illustrating the class an object belongs to needs not defining upfront. Additionally, the schema-less model can be used to represent the relationships between classes, like the class hierarchies.

The data collected from various sources are instances and structures extracted from the instances are schemas. Hence, there are two kinds of extended nodes (EN) in SGM: object nodes (ON) and class nodes (CN).

Definition 1. ON = (L, P), where

$L = \{l_1, ..., l_n\}$ is an optional set of class names (or labels) that the object belongs to, where $l_i \in \mathcal{L}$ with $1 \leq i \leq n$ is a class name. As an example from Fig. 1, the label of node China is Country. In SGM, the labels of an object can be omitted if the classification information is unknown.

$P = \{p_1, ..., p_n\}$ is a set of properties, where each p_i is an attribute-value and the value can be a array or a aggregated one. The value of a property can also be a node in SGM. If the value of a node property is a node, it should be represented as a relationship. Hence, the value of a node property is better not a node. A relationship property whose value is a node indicates relationship-based relationships, which will be illustrated in Section 4.1.

Definition 2. CN = (R, EP), where

$R = \{r_1, ..., r_n\}$ is a set of class hierarchical relationships as it shown in Example 1.

$EP = \{ep_1, ..., ep_n\}$ is a set of property definitions extracted from object data.

In this section, we explain the object-oriented features SGM extends and how the class information and complex property are represented in SGM.

3.1 Objects and Classes

The label of every object node makes it possible to extract the definition of every type, otherwise what we can get are the property and relationship names without classifications. Hence, a node in SGM is an object which is an instance of a particular class or multiple classes. As SGM is schema-less and there are no definitions of attributes and relationships for a given type or class of an object, and objects are not required to match a schema. The classes an object belongs to do no impact on the object and just indicates the type of the object. In some cases, the knowledge of an object's classification maybe remain unknown, and the class of an object can be a default one *Object* that is the superclass of any classes.

Although the definition of a class is not required at the beginning, the classes and structures can be extracted from the object data. The structure and content of a class may be huge because different sources depicts different aspects of data in the same domain and language expressions vary. Hence, the extracted structure is a complete representation of objects in a class and a collection of

language statement of all properties of a class, in which we can find the synonym sets under the context of a class.

As the latest release of Neo4j supports nodes with labels, which can be treated as classifications of objects, we directly follow the usage of label to denote objects and classes.

3.2 Composition and Inheritance

Objects can contain other objects; this is known as object composition. As this feature is related to relationships between objects, it will be elaborated with containment relationship in Section 4.1.

With introducing classes into SGM, it's natural to support subclass relationships between classes. This is predefined conceptual knowledge and can be used for data analysis after the classes are extracted from object data. In SGM, the inheritance mainly displays a hierarchy that represents "is-a-type-of" relationships among classes. What is different from usual inheritance is that subclasses will not inherit the properties of the superclass as the definition of a superclass is unknown.

Schema information classifies objects and the class hierarchies will make sense when querying and analyzing.

In SGM, we can define classes and their inheritance relationships via class definition statements, which is new from existing graph models. Neo4j supports definition of domain model, that is the schema of the graph. However, there is no metadata of the class hierarchies. Hence, we bring a keyword *class* to create statement to indicate the class node definition.

Example 1. The following are two statements representing that Country, Province and City are subclasses of Region.

```
create class (Region), class (Country), class (Province),
   class (City), (Region) - [:subsume] -> (Country),
   (Region) - [:subsume] -> (Province), (City) - [:isa] -> (Region);
```

3.3 Complex Properties

SGM is a property graph model and every node and relationship can have properties. To directly represent multiple value and nested attributes, we support the array and aggregation of properties in SGM.

Example 2. The following is the node creation statement for Person Alice who is female, has two phone numbers (+1(613)520-2525 and +1(613)520-4049), and the address is an aggregation of four values, which are addressline "1125 Colonel By Drive", city Ottawa, country Canada and address type "Company address".

```
create (Alice:Person { gender:female,
   phone_no.:[+1(613)520-2525,+1(613)520-4049]
   address:{AddressLine1:"1125 Colonel By Drive",
      City:Ottawa, Country:Canada, Type:"Company address"}});
```

4 Extension of Relationships

In this section, we introduce semantics attached to relationships (SE): relationship type (including containment relationship, role relationship and user-defined-type relationship), relationship hierarchies and inverse relationship.

Definition 3. SE = (t, P, h, i), where

t is the type of a relationship, and the default relationship is a general one without special semantics. t can be any value such as containment, role or other user-defined ones.

$P = \{p_1, ..., p_n\}$ is a set of properties as it is defined in Definiton 1. The value of a relationship property can be not only text and can be other nodes, which indicates the interdependency between relationships.

h indicates the hierarchical structure of a relationship.

i is the inverse relationship which establishes the connection between mutual relationships.

4.1 Relationship Type

Containment Relationships. As it is mentioned in Section 3.2, Object composition is used to represent "has-a" relationships: every employee has an address, so every Employee object has a place to store an Address object. This is a feature of object-oriented model.

The address of a person is a detailed geographic location that can be represented as aggregated properties as shown in Section 3.3 and can be also represented as an entity node with a connection to the person. The description of a geographic location always contains information of country, state/province, city and street that is from a larger scale to smaller ones. That a country is comprised of states/provinces and that a state/province consists of cities are special relationships with containment semantics.

There are two reasoning rules of containment relationship. Firstly, included entities depends on larger-scale entities. Once an entity is deleted, the entities it contains will also be deleted. Secondly, the containment relationship is transitive; that is, if A contains B, and B contains C, then A contains C.

Example 3. In Figure 1, it displays the containment relationships between Country China with Province Hubei and Guangzhou, and with Municipality Beijing, as well as the containment relationships between Province Hubei and City Wuhan and Yichang. As it shows, included nodes are inside larger-scale nodes.

```
create (China:Country {name:"China"}), (Beijing:City {name:"Beijing"}),
   (Wuhan:City {name:"Wuhan"}), (Guangzhou:Province {name:"Guangzhou"}),
   (Hubei:Province {name:"Hubei"}), (Yichang:City {name:"Yichang"}),
   (China) -[containment:municipality] -> (Beijing),
```

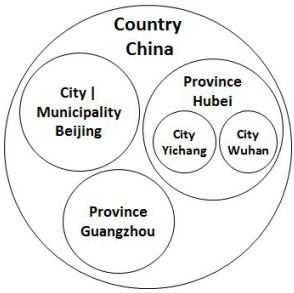


Fig. 1. A graph of containing objects.

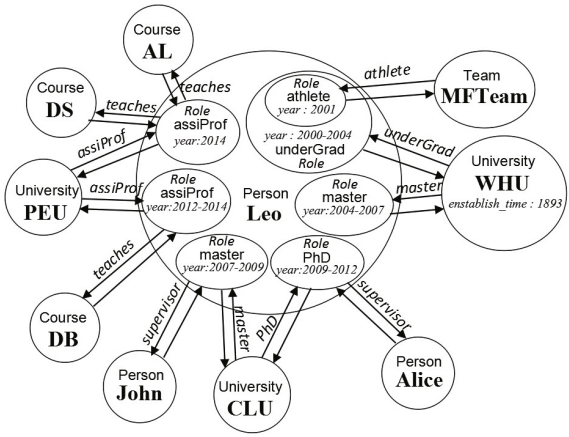


Fig. 2. A graph of Leo's resume.

```
(China) -[containment:provinces] -> (Hubei),
(China) -[containment:provinces] -> (Guangzhou),
(Hubei) -[containment:city] -> (Wuhan),
(Hubei) -[containment:city] -> (Yichang);
```

Note that keyword *containment* is introduced to indicate the containment semantic of the connection.

Role Relationships. An object has static aspects as well as dynamic aspects. Real world entities have various natural and complex relationships with each other and via these relationships, objects play various roles that form their contexts, and then have the corresponding context-dependent properties. Object properties are often based on their contexts, and contexts can be nested to form complex context dependent information.

For example, a student may have studied at one or more universities as an undergraduate, master, or Ph.D student, and thus may have several distinct properties with the same name (e.g. year). Each university involved is the context. Undergraduate, master, and Ph.D are also nested contexts for the properties. Similarly, a faculty member may have worked at one or more universities as an assistant professor, associate professor, or full professor and thus may have some context-dependent properties with the same name as well.

Existing graph models treat roles of an entity as individual nodes and connects role nodes with entity nodes and context organization nodes respectively, while role nodes are not independent entities and have containing relationships with original entity.

Example 4. A person's resume (education and career network) is a typical example that displays connections between he/she and some schools or companies, and different roles he/she plays in different organizations. In Figure 2, it illustrates the education and work experience of Person Leo. He was an undergraduate student in university WHU from year 2000 to year 2004, and as an undergraduate student, he participated the Man's Football Team in year 2001. From year 2004 to year 2007, he obtained master's degree in WHU and under this context, he was supervised by Ben and took course OS. Then he turned to university CLU and obtained master's degree for year 2007 to year 2009 and was supervised by John, and he obtained doctor's degree for year 2009 to year 2012 and switched to supervisor Alice. After getting his doctor's degree, Leo worked in university PEU as assistant professor from year 2012 to year 2014 and taught course DB. Since 2014, he is a associate professor and teaches course DS and AL.

The following statement shows how to represent above situation in SGM. It represents node Perosn Leo, and connections that Leo relates to university WHU, CLU and PEU, roles that Leo respectively plays under those contexts and context-dependent properties and relationships.

```
create (Leo:Person {name:"Leo"}), (WHU:Univ {name:"WHU"}),
   (CLU:Univ {name:"CLU"}), (OS:Course {name:"Operating System"}),
   (Ben:Person {name:"Ben"}), (DS:Course {name:"Distributed System"}),
   (Alice:Person {name:"Alice"}), (PEU:Univ {name:"PKU"}),
   (DB:Course {name:"Database"}), (John:Person {name:"John"}),
   (AL:Course {name:"Algorithm"}),
   (ManFbTeam:Team {name:"Man'sFootballTeam"}),
   (WHU) <- [:studies_in {(Leo:UnderGrad {year:2000-2004}) -
         [:participate {(Leo:Athlete {year:2001})}] -> (ManFbTeam),
      (OS) <- [:take] - (Leo:Master {year:2004-2007})
      - [:supervisor] -> (Ben)} ] - (Leo) -
   [:studies_in {(Leo:Master {year:2007-2009}) - [:supervisor] -> (John),
      (Leo:PhD {year:2009-2012}) - [:supervisor] -> (Alice)}] -> (CLU),
   (Leo) - [:works_in {(Leo:AssiProf {year:2012-2014}) - [:teaches] ->
(DB)},
      (AL) <- [:teaches] - (Leo:AssoProf {year:2014})
      - [:teaches] -> (DS)] -> (PEU);
```

On the one hand, to represent the roles Leo play under different contexts and on the other hand, to represent the relationships and properties based on the role, in the relationship that the context-dependent properties occur, there is not only text value but also node value for a relationship property, and properties with node values are also treated as relationship based relationships or context-dependent relationships. For example, in the relationship Leo studies in WHU, there are two properties in this relationship and each of the properties is an relationship between a role object derived from the role relationship and other objects. A context dependent relationship can also have context-dependent relationships, like the the relationship Leo participate Man's Football Team, which

depends on the relationship that Leo is an under graduate student in WHU. Hence, the relationship behaves in nested way.

A role node, such as the node Leo:UnderGrad, is a special object node in order to represent an object combined with a certain context and context-dependent relationships by using role nodes as their start/end nodes. The role an object plays is also a class, as shown in role node Leo:UnderGrad, UnderGrad is a role class and a role class is the subclass of the class of the original object, for example UnderGrad will be a subclass of class Person. This role class and class relationship will be used when model extracting.

The following is the modeling example of Leo's resume in a different way, that is starting from organizations, WHU, CLU and PEU, Leo is connected by *role* relationships. Corresponding to every role relationship, a role node will be generated in according entity. For example, induced from role relationship underGrad between WHU and Leo, role node WHU.underGrad is generated in entity Leo. It is important to note that the role relationship *athlete* between Man's Football Team and role node WHU.underGrad Leo.

```
create (WHU) - [role:UnderGrad {year:2000-2004,
      (Leo:UnderGrad) - [:participate {year:2001}] -> (ManFbTeam)}] ->
(Leo),
   (WHU) - [role:Master {year:2004-2007}] -> (Leo),
   (CLU) - [role:Master {year:2007-2009,
      (Leo:Master) - [:supervisor] -> (John)}] -> (Leo),
   (CLU) - [role:PhD {year:2009-2012,
      (Leo:PhD) - [:supervisor] -> (Alice)}] -> (Leo),
   (PEU) - [role:AssiProf {year:2012-2014,
      (Leo:AssiProf) - [:teaches] -> (DB)}] -> (Leo),
   (PEU) - [role:AssoProf {year:2014-,
      (AL) <- [:teaches] (Leo:AssoProf) - [:teaches] -> (DS)}] -> (Leo);
```

User Defined Relationship Type. Besides the two pre-defined relationship type which has the certain inference rules, user can classify a relationship to any type according to the application requirement, and by query statements, this type property can be used to infer certain results.

4.2 Hierarchical Relationships

The relationships between objects are sometimes hierarchical from a general one to more specific ones. Usually, the most direct relationship is maintained in graph model, and the semantics indicates the meaning and structure of object connections is lost. SGM keeps the relationship with its hierarchies.

Example 5. Figure 3 shows the committee structure of CIKM 2015. The committee of CIKM 2015 is specified into relationships "General Chairs" and "Program Chairs", and "Program Chairs" is further specified for tracks, such as "Database Track" and "Tutorial Chairs".

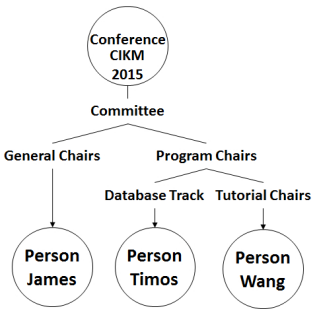


Fig. 3. The Committee of CIKM 2015.

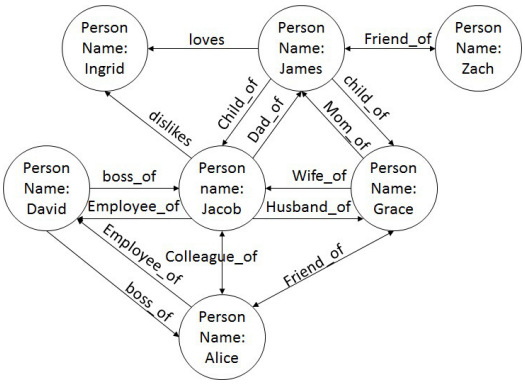


Fig. 4. Jacob's social network (relationships and inverse relationships).

```
create (CIKM2015:Conference {name:"CIKM 2015"}),
   (James:Person {name:"James Bailey", affiliation:"The University
of Melbourne"}),
   (Timos:Perosn {name:"Timos Sellis", affiliation:"RMIT
University"}),
   (Wang:Person {name:"Wei Wang", affiliation:"University of New
South Wales"}),
   (CIKM2015) - [:Committee] - [:General_Chairs] -> (James),
   (CIKM2015) - [:Committee] - [:Program_Chairs] -
[:Database_Track] -> (Timos),
   (CIKM2015) - [:Committee] - [:Program_Chairs] -
[:Tutorial_Chairs] -> (Wang);
```

Based on the relationship of Cypher, between nodes, more notations (-[]) are used to represent the relationship hierarchy. For example, between nodes CIKM2015 and James, the relationship is comprised of two parts, from Committee to General Chairs.

4.3 Relationships and Inverse Relationships

The relationships between entities are either unilateral or mutual. Affection connections can be unilateral, for instance one man loves the other while the love may not be requited. Compared with unilateral ones, mutual relationships are more prevalent. In such a case, once a one-way connection occurs from an entity to another, the reflexive connection also needs to be established.

For mutual relationships, people always define two relationships in two directions to make the mutual semantics complete. The two relationships for a mutual connection have two features. Firstly, the label of a mutual relationship in two directions can be either the same or different. Secondly, this two relationships likewise dependent on each other, and once connection in one direction changes, the other one should change correspondingly. From the perspective of semantic capturing, the interdependency relationship between two mutually inverse relationships are important for data analysis. Existing graph models do not support this feature of mutual connections. Hence, in SGM, users can directly define mutual relationships using relationship pairs, so that the system can know and maintain the consistency of relationship pairs.

Example 6. Figure 4 illustrates a network of a person's family, friends and colleagues. Jacob and Grace are a couple and James is their child. Jacob and Alice are both employees of David. James loves Ingrid (the love is not requited), while Jacob dislikes her. Grace and Alice are friends, so are James and Zach. As the figure shows, parallel and pair directed lines indicate mutual connections, in which one is another's inverse and they have different labels. The two way lines, like friend_of can be treat as a combined pair relationships for they have the same label in two directions. Single directed lines are uniliteral relationships.

```
create (Jacob:Person {name:"Jacob"}), (Grace:Person {name:"Grace"}),
   (Ingrid:Person {name:"Ingrid"}), (James:Person {name:"James"}),
   (Zach:Person {name:"Zach"}), (David:Person {name:"David"}),
   (Alice:Person {name:"Alice"}),
   (Jacob) - [:husband_of(<- wife_of)] -> (Grace),
   (Jacob) - [:dad_of(<- child_of)] -> (James),
   (Jacob) - [:dislikes] -> (Ingrid),
   (Jacob) - [:employee_of(<- boss_of)] -> (David),
   (Jacob) - [:colleague_of] - (Alice),
   (Grace) - [:mom_of(<- child_of)] -> (James),
   (Grace) - [:friend_of] - (Alice),
   (James) - [:friend_of] - (Zach),
   (James) - [:loves] -> (Ingrid),
   (Alice) - [:employee_of(<- boss_of)] -> (David);
```

As shown above, -> indicates outgoing relationships, <- indicates the inverse ones. - represents mutual relationships with the same labels in two direction. Within a outgoing relationship, we use a clause (<-) to indicate the inverse relationship of the current one. On the one hand, it defines the inverse relationship and needs not another clause for the inverse relationship. On the other hand, it represents that that like "husband_of" and "wife_of" are conceptually inverse to each other.

5 Implementation Upon Neo4j

5.1 Data Conversion to Cypher

To represent the extra semantics introduced by SGM, we take advantage of properties and labels of Neo4j. For nodes and relationships, we assign some properties with particular functions to keep the semantics. In this section, we introduce the particular properties and structures we use for SGM in Neo4j system.

Type. *Type* is a most important property we use to represent the type of a class or a relationship. Firstly, there are three kinds of nodes in SGM: class, object and role. The type of an object node is *object* and a class node *class*. Role node is a special one that represents roles an object plays, and by the role node the context-dependent properties can be represented and kept in Neo4j. Secondly, *type* keeps the relationship type for a relationship.

Path. For a hierarchical relationship, the each layer of it may have its own properties and hierarchies are not only relationship labels. Also, two objects connected by a hierarchical relationship can be regarded as that they have connections of each layer of the hierarchies, and a general relationship contains a more specific one. Thus, for hierarchical relationship, we split the every layer out and connect the objects by them, and for the more specific relationship, its super relationships are kept in the property *path*.

Inverse. For relationships which have according interdependent inverse relationship, SGM will automatically generate their inverse ones if given the inverse clauses. To convert relationship to Cypher, two independent relationships are created from two directions, and for every relationship there is a property *inverse* indicating its inverse relationship.

New Nodes. There are situations new intermediate nodes are needed. One is for aggregated properties as Neo4j does not support aggregation of properties and new nodes are generated to capture the aggregation information of properties. The second situation is for a property when its value is a node that is the context-dependent relationship, and in this case role nodes are generated.

5.2 Query Examples

Figure 5 illustrates the results of data from above examples converted and stored in Neo4j. Note that the properties of nodes and relationships are omitted in the graph.

There give some query examples which leverage the semantics of SGM. For every query, the result is presented. As every relationship has a property type, query pattern can match the relationship by type. In this way, we can get meaningful relationships in a schema-less manner without knowing the relationship names. Moreover, we can get results with certain semantics by taking advantage of relationship semantics, like containment and role.

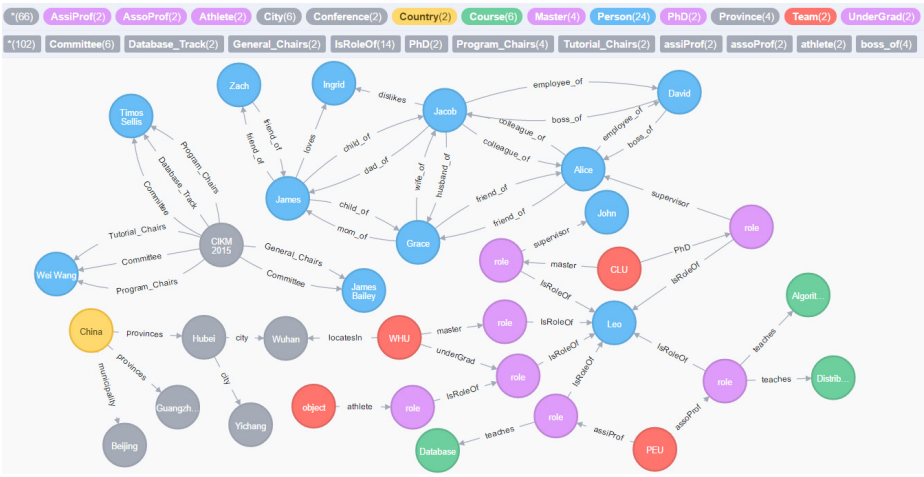


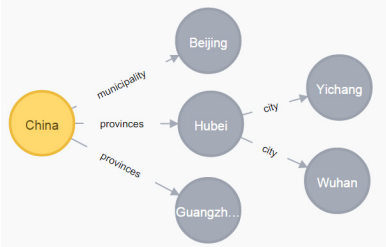
Fig. 5. Results of SGM Data Managed in Neo4j.

1. The following query finds the relationships which have containment relationship with China.

```
Match (n {name:"China"})-[r {type:"containment"}] -() return n, r;
```

The following one finds the containment topology graph of node China. Note that *1..4 means traverse the relationships in 1 to 4 hops and the condition {type:"containment"} indicates that only continue traversing if the relationship type is containment.

```
Match (c {name:"China"})-[r *1..4 {type:"containment"}] -> (p)
return c, p;
```

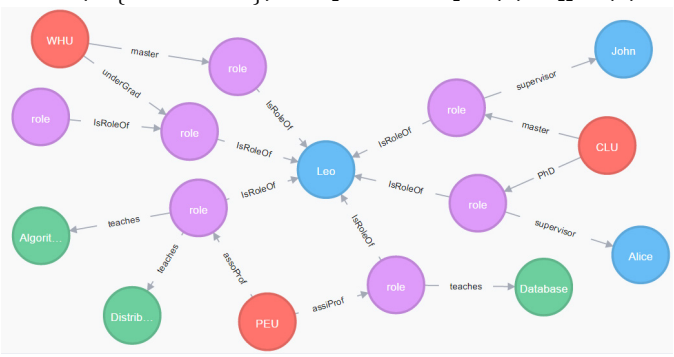


2. If we want to find China's university, the universities locating in cities and provinces China contains should be returned. Taking use of the containment relationship, directly find the universities locates where China contains regardless of the relationship labels.

```
Match (u)-[:locatesIn]->()<-[r1 *1..4
{type:"containment"}]-(c {name:"China"}) return u, r1, c;
```

3. The following query returns the object Leo and all roles he plays.

```
Match (n {name:"Leo"}) <- [:IsRoleOf] -(r) -[]- (o) return n, r, o;
```



4. Find the context-dependent relationship that Leo has as a Master student.

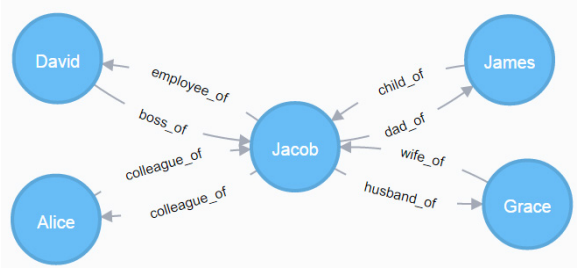
```
Match (n {name:"Leo"}) <- [:IsRoleOf] -({name:"role:Master"})
-[r]- (o) where r.type="general" return r;
```

5. As the split of hierarchical relationship, the super relationship will contain sub relationships. Hence, for the following two queries, the result of the first one ([“James Bailey”,“Timos Sellis”, “Wei Wang”]) contains that of the second one ([“Timos Sellis”, “Wei Wang”]).

```
Match (c {name:"CIKM 2015"}) - [:Committee] -> (p) return p;
Match (c {name:"CIKM 2015"}) - [:Program_Chairs] -> (p) return p;
```

6. Find the relationships that have corresponding inverse relationships of Jacob.

```
Match (n {name:"Jacob"}) - [r] -> (t) - [ir] ->(n) where ir.name =
r.inverse return r,ir;
```



5.3 Extracted Domain Model

Model information is an outline of the nontrivial integrated heterogeneous data and provides conceptual knowledge like semantic correlation of words from a domain and etc. Additionally, the model information can be used to analyze data structure and assist query optimization, like cost estimation and optimal plan selection and etc. Although the property and relationship definitions of a

class are not given at the beginning, we can extract model information from the labeled nodes and relationships.

Following statement is a part of the extracted class definition of above example data in Cypher. To distinguish the metadata from instances, the type of class nodes is specified as "class". The model information are generated.

```
create (Country:Country {name:"Country", type:"class"}),
   (City:City {name:"City", type:"class"}),
   (Province:Province {name:"Province", type:"class"}),
   (Country) - [:municipality type:"containment"}] -> (City),
   (Country) - [:city {type:"containment"}] -> (Province),
   (Province) - [:city {type:"containment"}] -> (City)
   (Univ:Univ {name:"Univ", type:"class" }),
   (Person:Person {name:"Person", type:"class"),
   (Team:Team {type:"Team", type:"class"}),
   (UdGrad:UnderGrad {name:"UnderGrad", type:"class"}),
   (Athlete:Athlete {name:"Athlete", type:"class"}),
   (Master:Master {name:"Master", type:"class"}),
   (UnderGrad) - [:IsRoleOf] -> (Person),
   (Athlete) - [:IsRoleOf] -> (UdGrad),
   (Master) - [:IsRoleOf] -> (Person),
   (Univ) - [:undergraduate {type:"role"}] -> (UnderGrad),
   (Team) - [:athlete {type:"role"}] -> (Athlete),
   (Master) <- [:master {type:"role"}] - (Univ),
   (Conf:Conference {name:"Conference", type:"class"}),
   (Conf) - [:Committee] -> (Person),
   (Conf) - [:General_Chairs {path:["Committee"]}] -> (Person),
   (Conf) - [:Program_Chairs {path:["Committee"]}] -> (Person),
   (Conf) - [:Database_Track {path:["Committee",
      "Program_Chairs"]}] -> (Person),
   (Person)- [:husband_of {inverse:"wife_of"}] -> (Person),
   (Person)- [:dad_of {inverse: "child_of"}] -> (Person),
   (Person)<- [:wife_of {inverse:"husband_of"}] - (Person),
   (Person)<- [:child_of {inverse:"dad_of"}] - (Person);
```

Figure 6 demonstrates the model information we extracted and presented in Neo4j. There are 14 classes in which 6 classes are induced from role relationships. Roughly, we can extracted some knowledge from the semantic object data. Firstly, Country consists of provinces and provinces consists of cities which are induced from containment relationships. Secondly, from role relationships, Person can play roles: undergraduate, master, athlete and etc. Thirdly, relationship "committee" can be specified as relationship "program_chairs", and if two objects only have relationship "program_chairs", we can induct that they are possibly have a relationship "committee". The quality of heterogenous data

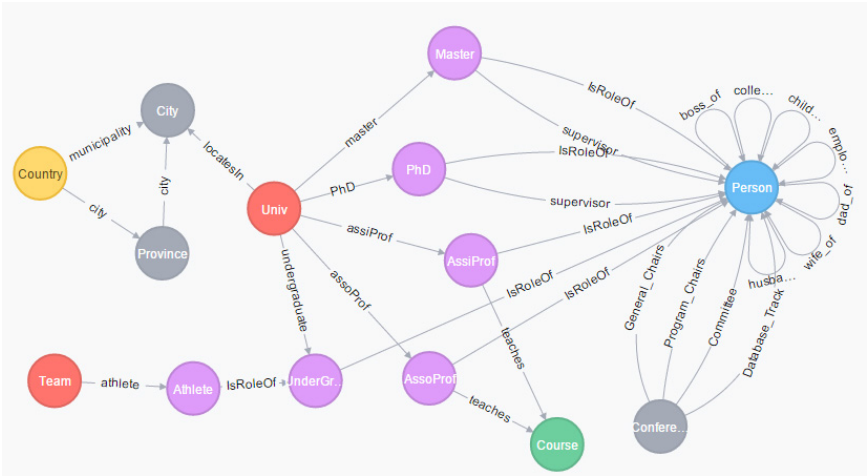


Fig. 6. Neo4j Results of Extracted Model.

varies, extracted model from data with high quality and more information can be used to complement imperfect data and benefit schema-less query.

6 Conclusion

Graph models do excel where data have an element of uncertainty or unpredictability and the relationships are the features of the data. To represent and integrate heterogeneous data and at the same time to capture as much semantics as possible, we have proposed a novel graph model called Semantic Graph Model (SGM), that extends the nodes of graph model with object-oriented features and attachs some semantics to relationships.

Objects and classes are introduced to nodes of SGM; that is, every node is an object which is an instance of one or more classes. Additionally, we have introduced two kinds of relationship: containment relationship and role relationship, a new relationship structure, hierarchical relationship, and a new relationship between relationships; that is, the relationship and its inverse relationship are coexisting. Containment relationship is introduced to represent relationships with composition semantics. Role relationship models the dynamic aspect of an object. Some relationships are directed, some are bidirectional, and some directed relationships are pairwise. Pairwise relationships have a strong dependency on each other and they are fundamentally one relationship between two objects with two different expression from two directions. Hence, aiming at these pairwise relationships, relationship and its inverse can be represented and maintained in SGM.

The features of SGM are as follows. Firstly, it is a schema-less model. Data with various structures and semantics can easily be represented with it. Secondly,

besides the label of nodes and relationships, their type can be specified as well. Semantics of the data are represented and captured as what they demonstrate in sources. Thirdly, the structures and definitions of classes can be extracted from the instance data, and other knowledge can be mined from the extracted model and used on further integrate and fuse data. Finally, the extra semantics captured by SGM are important to graph query optimization.

We have also implemented a conversion layer to leverage the data management provided in Neo4j. We firstly convert the data represented in SGM to data that the graph database Neo4j can process, secondly make use of the semantics SGM captured to do some queries, and thirdly extract domain model from the object data.

In the future, we will investigate how to integrate, fuse and analyze data using SGM and how to apply SGM to applications by leveraging extracted model information, and we will study the query optimization methods taking advantage of structural semantics.

References

1. Bondy, J.A., Murty, U.S.R.: Graph theory with applications. Macmillan, London (1976)
2. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semantic Web Inf. Syst., 1–22 (2009)
3. Madhavan, J., Halevy, A.Y., Cohen, S., Dong, X.L., Jeffery, S.R., Ko, D., Yu, C.: Structured data meets the web: A few observations. IEEE Data Eng. Bull. **29**(4), 19–26 (2006)
4. Talukdar, P.P., Ives, Z.G., Pereira, F.: Automatically incorporating new sources in keyword search-based data integration. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 387–398. ACM (2010)
5. Klyne, G., Carroll, J.J., McBride, B.: RDF 1.1 Concepts and Abstract Syntax (2014). http://www.w3.org/TR/rdf11-concepts/
6. Isaac, A., Summers, E.: SKOS Simple Knowledge Organization System Primer (2009). http://www.w3.org/TR/2009/NOTE-skos-primer-20090818/
7. Hitzler, P., Krotzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S.: OWL 2 Web Ontology Language Primer, 2nd edn. (2012). http://www.w3.org/TR/2012/REC-owl2-primer-20121211/
8. Suchanek, F., Weikum, G.: Knowledge harvesting in the big-data era. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 933–938. ACM (2013)
9. Bizer, C., Boncz, P., Brodie, M.L., Erling, O.: The meaningful use of big data: four perspectives-four challenges. ACM SIGMOD Record **40**(4), 56–60 (2012)
10. Dong, X.L., Srivastava, D.: Big data integration. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE), pp. 1245–1248. IEEE (2013)
11. Hull, R., King, R.: Semantic database modeling: Survey, applications, and research issues. ACM Computing Surveys (CSUR) **19**(3), 201–260 (1987)

12. Flesca, S., Greco, S.: Querying graph databases. In: Zaniolo, C., Grust, T., Scholl, M.H., Lockemann, P.C. (eds.) EDBT 2000. LNCS, vol. 1777, pp. 510–524. Springer, Heidelberg (2000)
13. Rodriguez, M.A., Neubauer, P.: The graph traversal pattern. In: Graph Data Management, pp. 29–46 (2011)
14. Wood, P.T.: Query languages for graph databases. SIGMOD Record, 50–60 (2012)
15. Miller, J.J.: Graph database applications and concepts with neo4j. In: Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA, March 23–24, 2013
16. Robinson, I., Webber, J., Eifrem, E.: Graph Databases, 2nd edn. O'Reilly Media Inc., USA (2015)