Survey of Graph Database Models
(ATFD Topic 6, Paper 4)

Kristan Papirio s1674939

April 14, 2017

1 Introduction

The study of graph database models that was popular in the eighties and nineties, gradually subsided with the shift toward other database models such as spacial, semistructured, geographical and XML. However, such a focus once again began to draw attention as the need arose to manage graph-like information.

The scope of the work presented in this paper is limited to a survey of present research in graph databases. As a result, the focus is not on the analysis or critique of current work, but rather an objective overview of the data structures, query languages, and integrity constraints of explored models. In doing so, a systematic examination of developments is presented.

The main contributions offered by the authors include a conceptual overview of database models, including a summation of the various definitions presented in literature; a summary of the distinctive components of graph database models; and a comparison of the characteristics and draw backs of graph database models in contrast to other relevant database models. An in-depth description of these alternative database models is also given for comparability purposes.

2 Graph Database Models

The majority of the proposed definitions agree that *graph database models* are defined with respect to the following: Schema and data structures modeled by directed graphs or generalizations of graphs that are often labeled, graph oriented operations and type constructors used for data transformation, and graph structures used to represent suitable integrity constraints. Graph database models are motivated by real-world applications in areas that value interconnectivity and topology with the same regard as actual data.

The use of these models has three key advantages. First, data is structured and handled by the user in a more intuitive way that promotes the centralization of all information about an entity in one node and provides a visual representation of its relationship to related information. Second, due to the graph operations that are expressed in query language algebra, graph structure can be directly referred to by queries. Lastly, with respect to the actual implementation, specific operations can be accomplished by efficient algorithms and storage structures.

The necessity for graph databases can be seen through the shortcomings of classical applications. Traditionally, classical models prohibit the modeling of complex objects by hindering the ability to visualize data due to the flat nature of data structures. Often times, the complexity of the data surpasses the ability of the relational database and query languages are not powerful enough to express this complexity. Classical models also experience limitations in their ability to represent knowledge in a detailed and adaptable way. Recognition that graphs are a fundamental aspect of database design instigated a call for the development of graphical interfaces, as well as pictorial and multimedia systems. Additionally, the emergence of hypertext and the Web produced a need for alternative models to handle information exchange.

Social, information, technological, and biological networks are examples of complex networks whose mathematical properties require graph database models for data management. Social networks capture the relation between individuals and groups of people who are connected through personal relationships, business relationships, research communities, etc, and require data processing methods that can provide useful visualizations. Information networks deal with information flow. Examples of these relations are literature citations, peer networks, and the World Wide Web. Technological Networks such as the internet, telephone networks, and Geographic Information Systems (GIS), require a structure that exemplifies the importance of spatial and geographical features. Due to the automation of data collection, biological networks experi-

ence challenges in managing the volume and analysis of their data. Examples of such networks include, but are not limited to, neural networks, genomics, and food webs. In all mentioned networks, different measures are used to compensate for the inability of traditional query languages to express desired queries.

2.1 Types of Database Models

Database models are simply comprised of the conceptual tools and mathematical frameworks that reflect relationships between real-world entities. In practice, they have become abstraction tools that can be used to specify permitted data types, support database methods, adapt to developments, develop query languages, aid data manipulation, focus on the architecture of database management systems (DBMS), and encourage research in data management systems and corresponding characteristics. The following are examples of what the authors deem to be the most influential types of database models.

Physical database models, such as hierarchical and network, were among the first with the ability to manage large collections of data and utilize low-level operations that allow for navigation ease and support abstract structures. However, it is difficult to separate these models from the implementation, as abstraction is not appropriately supported. In addition, physical models can only represent traditional applications due to the unadaptable nature of provided data structures.

Relational database models were developed in response to the lack of abstraction levels in physical databases. Such models create a separation between physical and logical levels, and model data based on what is seen and experienced by the user. The mathematical foundation and corresponding query language, SQL, have made these databases a focal point in the research community. The main differences from graph database models are the fixed schema of relational models that restricts the ability for database expansion and the inability of query languages to analyze relationships between data points and underlying patterns.

Semantic database models incorporate an expressive set of semantics that enable the user to view objects and their relations clearly. High level abstraction concepts, such as classification and aggregation, allow for designers to provide these benefits to users. Semantic databases aid research in graph databases through the examination of the graph-like structures that result from entity relations.

Object-oriented (O-O) database models were developed in response to the failure of semantic databases for knowledge bases and engineering applications, as well as the emergence of unconventional database applications that involve complex objects and relations. These models organize data into classes and, like relational databases, follow a fixed schema, and deviate from semantic models in their tendency to ascribe objects with unchanging, external identifiers. Object-oriented database models operate in a world where objects have certain states that are represented by data and interactions between states only happen through method passing. Such a view focuses on object dynamics. In contrast, in graph database models, the world is a interconnected network of object relations.

Semistructured database models are designed for semistructured, also known as unstructured, data that by nature is irregular and has a large, evolving schema that allows for data exchanges. Similar to graph databases, these models utilize a tree-like representation of data, that due to the allowance of node cycles results in a graph-like structure.

2.2 Specific Graph and Related Database Models

The following is an overview of common graph databases, as well as other relevant database models that utilize graph structures in some capacity, whether it be for navigation or language representation. All models are discussed in terms of their data structures, query language, and integrity constraints.

With the recognition of the need for semantics that were not offered by the relational databases at the

time, came the development of the *Logical Data Model* (LDM). LDM generalizes relational, hierachical, and network models by restructuring data and effectively using logical and algebraic query languages. The directed graph schema consists of four types of nodes: basic types, which contain stored data; composition types, which contain tuples of components taken from child nodes; collection types, which contain sets of elements from child nodes; and union types, where the domain of the collection is the union of the domains of child nodes. The allocation of values to these varying types of schema make up instances of nodes, which can either be sets from the instance of child nodes or a set of elements from the domain and tuples. The query language and integrity constraints of LDM are defined by a first order many-sorted language, as well as an equivalent algebraic language that allows for operations for the creation of nodes and relations; union, difference, and projections operations; and the reduction of instances.

The *Hypernode Model* is based on a nested graph structure. It is able to model complex objects with ease, provide a data structure for an O-O model, and exploit the use of a graph-based interface to increase the usability of database systems for complex objects. In general, hypernodes, which is a directed graph consisting of nodes which themselves can be graphs, represent hierarchical, composite, and cyclic objects, and mapping records. The second version used primitive and complex types depicted by nested graphs to introduce the concepts of schema and type checking. Additionally, the third version utilized hypernode functional dependency, in which attribute A decides the value of B in all hypernodes. The original declarative logic-based language that was used to query and update hypernodes was supplemented by Hyperlog, which is a rule-based language that allows for a user to query and browse with derivations. HNQL was later introduced so that programs could be created by compounded statements. The model also defines a set of entity, referential, and semantic integrity constraints over hypernode databases. However, the model struggles to manage data redundancy, and with limited restriction in the schema level.

Influenced by the Hypernode Model, the *Hypergraph-Based Data Model* (GROOVY), is used to model complex objects through the use of hypergraphs, which extend the concept of edges to hyperedges. The use of hypernodes allows for a single formalization of structural inheritance and subobject sharing to avoid redundancy, and the definition of of complex objects, functional dependencies, object-ID, and multiple structural inheritance. An object data model is defined by value schemas, objects over value schemas, funcitonal dependencies, objects schemas, object over object schemas, and class schemas, which can all be defined by hypergraphs. HML is the hypergraph manipulation language used to query and update hypergraphs through the use of two operators for for querying and eight operators for the insertion and deletion of hypergraphs and and hyperedges. Additionally, semantic integrity constraints for object schemas are determined by value functional dependencies.

Simatic-XT was developed as an object-oriented graph database model to deal with multi-scaled networks, such as train and water transport networks. The network type, as well as node and edge types, are represented by a labeled directed multi-graph. The model successfully integrates graph and object oriented paradigms. Object identifiers are used to reference each object and abstraction levels of subnetworks and paths are determined by the nested level of Master Nodes or Master Edges, which represent the set of edges that arrive in the subgraph, known as `in_edges`, and the set of edges leaving the Master Nodes and Edges, known as `out_edges`. Simatic-XT uses basic operates, which include the Develop and Undevelop operators and used to manage abstraction; elementary operators, which are used to manage the union, selection, concatenation, and difference of graphs and subgraphs; and high level operators, which are used to manage paths, inclusions, and intersections. Simatic-XT, however, fails to define integrity constraints.

The *Graph Database System for Genomics* (GGL) is used for biology purposes and based on the binary relationship that exists between objects. The model stores genome maps as graphs and represents edge types with vertices that allow for higher-order relations. These graphs consist of vertices, symbols, edges, packaged graph vertices, and relation type vertices. Vertices can be either labeled or unlabeled and model simple concepts; symbols are used to define nodes that do not have outgoing edges; edges contain a relation name label and connect two vertices together; packaged graph vertices depict graphs encapsulated into vertices; and relation type vertices represent higher-order relations. The model includes a genome graph language, as well as query operators. Additionally, two database integrity constraints are defined by GGL.

The *Graph Object-Oriented Data Model* (GOOD) develops database end-user graphical interfaces. Labeled direct graphs represent all instances and schema within the model. There are two types of nodes, printable and unprintable, and two types of edges, functional and nonfunctional, which are unique and multi-valued respectively. Additionally, there is no specification made between the atomic, composed, and set objects. Group transformations are used to express a data manipulation language that possesses graphical syntax and semantics. GOOD contains five operations: node deletions and insertion, edges deletion and insertion, and an abstraction operation for node grouping. The model also carefully manages the problem of data redundancy, allows for recursive relations, and studies the issue of computational-completeness.

The *Graph-Oriented Object Manipulation* (GMOD) model is primarily concerned with graph-oriented database user interfaces. GMOD consists of two labeled digraphs. The schema graph does not make a distinction between single and multi-value properties. The two classes of nodes include abstract objects that represent class names, and primitive objects that represent basic types. The properties of these abstract objects are in turn represented by edges. The instance nodes for both abstract and primitive objects, as well as the data, are contained within the instance graph. The properties of instance objects are defined by the same edges used in the schema graph. Uniform object manipulation for querying, updating, and browsing, is achieved by graph pattern matching. An example is the declarative query language, G-Log, which is compatible with GMOD. In addition, the model is able to avoid data redundancy, making the simple representation of objects and relations possible.

Object-Oriented Pattern Matching Language (PaMaL), an extension of GOOD, is a graphical object-oriented language that explicitly represents tuples and sets. The language uses patterns that are represented by graphs to designate the instance parts on which operations have been executed. The schema of PaMaL is defined by four types of nodes- class, basic, tuple, and set, and four types of edges that indicate the attribute of a tuple, type of elements in a set, type of class, and hierarchical relationship. Depending on the schema, an instance graph may or may not include atomic objects, which are labeled with domain values, instance objects, which are labeled with object-IDs, as well as tuples and set nodes, which are specified by outgoing edges. The latter reduces instance graphs by merging nodes that represent the same tuple or set. Additionally, deletion, addition, and reduction operators are provided by PaMaL, as well as loop, procedure, and program constructs. The operations, constructs, definitions of tuples and sets, multiple inheritance property, and graphics used to describe queries make PaMaL a computationally complete language.

Similar to PaMaL, the *Graph-Based Object and Association Language* (GOAL) is also an extension of GOOD, whose purpose is to describe instances and schemas of object databases by including association nodes in addition to objects. Associations define multi-attribute and multi-valued relations. The schema in GOAL are consistent and represented by finite directed labeled graphs and define object nodes, which are used to represent objects; value nodes, which are used to represent printable values such as Booleans; and

association nodes, which are used to represent the relationship between more than two nodes. Single-head edges are used to represent the potential functional properties, whereas double-headed edges are used to represent potential multi-valued properties. Instances are also represented by finite directed labeled graphs and are in charge of allocating values to value nodes and creating instances for association and object nodes. Associations and objects differ in that the identity of an object is not dependent on its properties, where as two associations with the same properties are regarded as identical. The operations of the provided data manipulation language include addition and deletion at the instance level and utilize pattern matching techniques. GOAL also has the ability to model incomplete information.

Graph Data Models (GDM) are graph based extensions of GOOD that add complex values, inheritance, and n-ary symmetric relationships. Both the schema and instances within GDM are represented by labeled graphs. Schema graphs are comprised of object, composite-value, and basic-value nodes. Edge attributes are used to indicate that that entities within that class may have the same attribute. These entities are represented by nodes in an instance graph, and their attributes by the edges. GUL is a graph-based update language that is provided by GDM and utilizes pattern matching. However, in addition to deletion and addition operations, GDM also includes a reduction operation which merges similar composite-value and basic-value nodes to allow data graphs to be reduced to instance graphs. The four constraints defined in GDM include no edge leaves from basic-value nodes; a real value assigned to each basic-value node that is in the domain of the node's basic type; a path for each class-free node, n, that starts in a class labeled node and ends in n; and composite nodes either labeled by exactly one class name or possessing exactly one in coming edge. Additionally, *extension relations* are used to indicate the correspondence between classes and entities through many-to-many relations between nodes in data and schema graphs. The main advantages of graph data models include, but are not limited to, the representation of semistructured data because of the independence of instances and schemas and the introduction of consistency and well-formed graphs.

Gram is a graph database model in which all data is structured in a graph. Directed labeled multi-graphs are used to represent schema in which nodes are labeled by a symbol called a type and edges are labeled in a manner that is representative of a relation between types. Gram uses regular expressions to define paths, also known as walks, which are represented by alternating sequences of nodes and edges. Hyperwalks are created by the combination of different walks. The algebraic language that allows for model querying defines an algebra for hyperwalks that permits unary and binary operations, such as concatenations and joins.

In addition to these models, the authors briefly describe auxiliary models that employ the use of graph-like features, despite a lack of concern for data connectivity. *GraphDB* models graphs in an object-oriented environment by defining class objects consisting of nodes, edges and stored paths. GraphDB has simple, link, and path classes, as well as data, object, and tuple types. Data is queried by derive, rewrite, union, and graphs operators. GOQL is a similar object-oriented graph model. *Database Graph Views* manage and query data independent of the implementation. Underlying graphs over the database are defined, and unary and binary derivation operations are used to select nodes and edges and create new graph views through intersections, unions, and differences. The *Object Exchange Model* (OEM) is a semistructured model that uses nesting properties and the notion of object identity to represent complex features. The syntax of OEM was designed for information exchange purposes and is represented as a rooted, directed, connected graph. OEM data is parsed without the recurrence to external schema, and therefore, self-describing. The *Extended Markup Language* (XML), although originally designed for information exchange between Web applications, has data that is represented by labeled, ordered trees and is valued by its meaning and not

just presentation. Despite the utilization of a restricted graph type, XML allows for graph simulation of semistructured data via a referencing mechanism. Additionally, unlike graph databases, XML is also self-describing. *Resource Description Framework* (RDF) is a mechanism for describing resources regardless of application domain and uses a graph-like structure to model data. Expression, which are labeled graphs, consist of a set of triples that include a subject, which is the resource being described, a predicate, which is the resource property, and an object, which is the property value. Many SQL, OQL, and XPath-like languages, such as SPARQL, have been proposed for querying RDF data.

2.3 Data Structures of Graph Database Models

All graph database models vary in foundational properties and data representation, depending on the presence of certain mathematical definitions of graphs. For example, graphs can be directed or undirected or contain labeled or unlabeled nodes and edges. Such diversity can be seen in the digraph structure of GOOD, GMOD, G-Log, and Gram models that have labeled nodes and edges, and the presence of hypernodes in the Hypernode, Simatic-XT, and GGL models that allow for the representation of complex objects. With the exception of LDM, GROOVY, and the Hypernode model, nodes and edges are explicitly labeled. Additionally, the use of a hypernode database instead of a simple flat graph supports abstraction, can represent each real-world object as an individual entity, can be defined by multiple complex relations, and promotes clear information presentation to the user.

One of the foundational aspects of a graph database is the representation of relations and entities. Entities are often referred to as objects and are used to characterize single and complete units. Graph database models have the ability to represent the connectivity between these units separate from the attributes of a relational database, the abstractions of a semantic database, the complex objects of an object-oriented database, and the composition relations of a semistructured database. In schema graphs, nodes with a label of zero define *entity types*, nodes with basic type labels define *primitive entities*, and edges with relation names as labels define *relations*. In instance graphs, nodes with entity type name or object identifier labels represent *concrete entities*, nodes with domain value labels of primitive entities represent *primitive values*, and edges with labels that correspond to relation-name determined by the schema represent *relations*. Additionally, PaMaL and GDM models explicitly represent tuples and sets, and GOAL and GDM include n-ary relations.

The property in which the connection between two or more entities is established is called a relation. *Simple relations* are represented by labeled edges and use simple semantics to connect two entities, whereas, *complex relations* can be represented in various ways, depending on the data structure of a given model and deal with networks of relations. The following are examples of six relation types that can be found in graph model:. *Attributes* can be defined in various ways, but are typically represented by labeled edges and associated with entities. Tuples are used by PaMaL and LDM for sets of attributes, triples inside of hypernodes and hyperedges are used by the Hypernode model and GROOVY, and functional and nonfunctional edges are used by GOOD and GOAL for monovalued and multivalued attributes. *Entities* define distinct model objects through the relation of two or more objects. This type of relation is supported by GROOVY, GGL, the Hypernode model, and is partially supported by GOAL. *Neighboorhood relations* can be visually represented by any model with a basic graph structure, but are not clearly expressed by models such as PaMaL that allow for the use of tuple-nodes. Models that contain hypernodes and hypergraphs transform these relations into nested relations. *Nested relations* are defined by nested objects and are recursively specified. *Standard abstractions* include aggregation, composition, association of n-ary

relations, and grouping. LDM, PaMaL, and GDM explicitly represent tuples; LDM and PaMaL explicitly represent sets; and GOAL and GDM explicitly represent n-ary relations. *Derivations and inheritance* are abstractions at the instance and schema level. Inheritance allows certain entity classes to share semantic content and is supported in some capacity by GOAL, PaMaL, GDM, the Hypernode model, and GROOVY.

2.4 Data Manipulation of Graph Database Models

The manipulation and querying of data is preformed by a set of operators and rules that define a query language. The authors focus on what they deem to most important graph querying languages.

A logic language and equivalent algebraic language that are similar to relational tuple calculus are used by LDM to perform queries that result in objects over valid instances that satisfy the query formula. A family of graphical languages that can be used for simple graph models consists of G, which is a language of regular expressions that permit the simple formulation of recursive queries; G+, which is an expansion of G that uses a query graph that specifies the class patterns being searched and a summary graph for simple queries; and GraphLog, which unifies the notion of a query graph and uses negation to query hypertext. Some models use graph-based, declarative, nondeterministic, and computationally complete languages, such as G-Log, that evaluate queries expressed as G-Log programs, which are a set of rules that define the nature of how instances and schema change. Many graph transformations are based on pattern matching and are utilized by models like GOOD, GOAL, and PaMaL. GOAL includes fix-points for the recursive nature of additions and deletions, and PaMaL includes loop, procedure, and program constructs. HML is a hypernode manipulation language (HML), used by GROOVY, to update and query hypergraphs. Other similar languages include operators for intersection, union, and difference, and the creation of edges. Languages that support querying of hypernode structures include a logic based language that expresses queries as a set of rules; Hyperlog, which includes insertions and deletions; and HNQL, which uses a set of operators to update and query hypernodes. Glide expresses queries using linear notation and uses a method of subgraph matching. Partially ordered languages that are often used to search the Web, provide insight into the computational complexity of path queries. Languages based on spatial logic use pattern matching and recursion to make variable substitutions in order to determine which graphs satisfy a given formulae. Simatic-XT utilizes a query language of basic, set, and high level operators. Additionally, GOQL and Gram support a SQL-style language with path, or walk expressions.

2.5 Integrity Constraints of Graph Database Models

Schema-instance consistency is a form of entity type and type checking that allows for the representation of database schema. Guidelines for checking this consistency are that only concrete entities and relations that were defined in the schema should be included in the instance, and that the entities contained in the instance can only include relations defined for its type. The models that comply to these constraint guidelines are GOOD, GDM, Gram, PaMaL, GOAL, GMOD, and G-Log. The LDM model, specifies constraints using a logic that determines whether a sentence is true in a given database. GROOVY uses an object schema that defines value functional dependencies, the validity of objects, and the validity of values that are shared by objects. *Schema-instance separation* is used to test the degree of difference between schema and instances in a database. Since data does not necessarily follow a previously defined data type, a data model can be either structured or unstructured, but there is almost always a clear separation between schema and instances. The *redundancy of data* is considered due to the fact that many models, such as GOOD, PaMaL, and GDM, allow instance graphs to contain multiple instances and entities. Abstractions are often introduced to account for this redundancy by creating a unique entity for duplicates.

Object identity and referential integrity constraints either identify tuples in relations by the values of their attributes, or identify objects by unique identifiers, such as a label. This allows for the identification of complex objects, objects that share subobjects, and the simplification of query and update operations. Models like GOOD, GMOS, GOAL, and G-Log utilized the former method, while PaMaL, Gram, GGL, GROOVY, and the Hypernode model use the latter. Two additional constraints are introduced by the Hypernode model. *Entity integrity* mandates that hypernodes are real world entities and identified by their content. *Referential integrity* ensure that the only existing entities are allowed to be referenced.

Functional dependencies are influenced by the relational model, which cannot be expressed by all graph models. GROOVY represents value functional dependencies that are semantic integrity constraints through the use of directed hyperedges. Similarly, the Hypernode model promotes Hypernode functional dependency in which a set of attributes, determines another set, for all hypernodes in the database.

3 Extension

As noted in section 2 above, the applications of graph databases are extensive, as they often allow for a more natural representation of data. The notion that the relationship between data points is more important than the actual data points themselves has insight implications for modern enterprises. Although common use cases of graph databases include fraud detection [2] and recommendation engines [3], for the sake of this essay, the focus will be on social networks.

3.1 Graph Databases and Community Detection

Social networks are a structural representation of how people interact with and relate to each other, and can determine how information flows within a group of people. Within every social network, there exists communities that share characteristics, interests, and other distinct properties. The question that arises is how similar these communities are to each other and what measures can be used to detect such properties? For the sake of this experiment, similarity is defined as the closeness between two communities.

Much work has been done to characterize the similarities between overlapping communities in the social networks represented by graph databases. Social networks are comprised of community components that are easily detected because of the predetermined groups that are embedded within them. In their paper [4], Yang and Leskovec explore the current performances for community detection that are evaluated manually and propose a more effective method. This is done by explicitly defining ground-truth communities, evaluating common structural definitions of these communities by the use of scoring functions of node connectivity, and extending the spectral clustering algorithm in a way that can detect communities based on a single seed node and scales to large networks. With the existence of these essential and now well-defined network communities, there is a foundation for the evaluation and development of similarity measures.

Queries can be made over graph databases for the purposes of community detection and determining closeness. One such algorithm, is the Jaccard Similarity Coefficient [5], which can be used to detect similarities between nodes and their corresponding attributes. However, connectivity within graph databases is an essential component, so therefore, it is not sufficient to simply look at nodes, as it lacks insight into the external edges that exist between communities and contribute to closeness. As a result, the Jaccard algorithm will be modified for these eternal edges, or weak ties, that span between communities, in order to provide a more exhaustive closeness measure.

3.2 Network Construction

For experimental purposes, an undirected social network graph was constructed, consisting of 1000 nodes with randomly generated edges that form ten ground truth communities, representing the memberships that are explicitly stated by users. Although intuitive distinctions can be made between the assignment of network nodes in these groups, they display heavy structural correlations and shared characteristics. Each node represents an individual in the network, and edges were constructed between nodes to determine community membership. To demonstrate how likely two people within a respective community are to be connected to each other, probabilities were randomly assigned for intra-community edge creation. The probability, $p_{a,b}$, of a node a and b being connected varies from community to community, as does the number of nodes claiming membership. This allows for a realistic portrayal of the division of people within groups in real life, which is scarcely uniform. In order to model the natural structure of a social network, some nodes were assigned dual community membership. As a result, edges construction between nodes in overlapping communities was considered twice, with respect to the probability of each of the two communities. The final step was the construction of individual communities to each other. Using the notion of weak ties [6], node construction was considered in which each end is in a distinctly different community. The resulting structure is a circular network, consisting of 13,127 edges and an average node degree of 26.3.

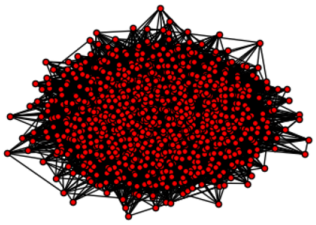

Figure 1: Undirected network graph.

3.3 Jaccard Algorithm and Edge Extension

The Jaccard Similarity Coefficient measures closeness in terms of the quantity of shared nodes [5] by dividing the nodes in the intersection of two communities by the union of all nodes in both. Let A and B be ground truth communities in a network, N, such that $A, B \subset N$. Then,

$$Jaccard = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

While this simplifies the relationship between nodes, it does not exploit the notion of connectivity that exists in graph databases. For example, suppose community A shares fifty nodes with community B and another fifty with community C. Now suppose that $X_b < X_c$, where X is the number of nodes in the community. The Jaccard algorithm above would suggest that community A is closer to community B than to community C. However, suppose that in community A, 50% of the external edges, *i.e* edges anchored at exactly one end in community A, connect to a nodes exclusively in community C. In contrast, only 5% of these edges coming from community A connect to nodes in community B. Such a situation would imply a level of closeness between communities A and C that is not clearly defined by the previous method. In order for such a connection to be sufficiently considered, an edge similarity algorithm was developed. Modeled closely after Jaccard's method, given a community A, the edge algorithm divided the number of external edges between two communities, A and B, by the total number of external edges anchored in A. Thus,

$$EdgeSimilarity = \frac{|E_{a,b}|}{\sum_{\forall i \in N, i \neq a} |E_{a,i}|} \tag{2}$$

3.4 Results

The results in the table below show the closeness between communities in the constructed network based on the Jaccard Similarity Coefficient and the modified Edge Similarity Algorithm.

			1	2	3	4	5	6	7	8	9	10
	1	Jaccard	-	0.247	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.277
		Edge	-	0.139	0.075	0.102	0.161	0.131	0.35	0.384	0.117	0.29
	2	Jaccard	0.27	-	0.182	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		Edge	0.15	-	0.10	0.103	0.146	0.123	0.354	0.381	0.127	0.329
	3	Jaccard	0.0	0.182	-	0.053	0.0	0.0	0.0	0.0	0.0	0.0
		Edge	0.112	0.14	-	0.14	0.152	0.116	0.313	0.316	0.12	0.30
	4	Jaccard	0.0	0.0	0.053	-	0.164	0.0	0.0	0.0	0.0	0.0
		Edge	0.089	0.083	0.082	-	0.335	0.126	0.329	0.353	0.126	0.278
	5	Jaccard	0.0	0.0	0.0	0.164	-	0.091	0.0	0.0	0.0	
		Edge	0.096	0.08	0.06	0.227	-	0.158	0.363	0.377	0.129	0.316
	6	Jaccard	0.0	0.0	0.0	0.0	0.091	-	0.118	0.0	0.0	0.0
		Edge	0.105	0.092	0.062	0.116	0.214	-	0.32	0.347	0.102	0.29
	7	Jaccard	0.0	0.0	0.0	0.0	0.0	0.118	-	0.197	0.0	0.0
		Edge	0.126	0.118	0.075	0.136	0.221	0.144	-	0.499	0.145	0.367
	8	Jaccard	0.0	0.0	0.0	0.0	0.0	0.0	0.1967	-	0.11	0.0
		Edge	0.14	0.129	0.077	0.148	0.233	0.159	0.506	-	0.185	0.419
	9	Jaccard	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.11	-	0.182
		Edge	0.095	0.096	0.065	0.117	0.176	0.103	0.325	0.409	-	0.248
	10	Jaccard	0.277	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.182	-
		Edge	0.115	0.121	0.079	0.126	0.211	0.143	0.403	0.453	0.121	-

Table 1: Community Similarity

As can be clearly seen in the table above, communities are most similar to neighboring communities, for which the combination of these algorithms gives a more holistic view. Due to the circular nature of the network, it is expected that communities only overlap with nodes on each side. In order for two communities to be relatively similar, there must be a relationship among both nodes and edges. For example, although community 10 shares approximately 42% of its external edges with community 8 (the highest of any other), there is an absence of overlapping nodes, causing community 10 to have the most in common with community 1, which shares only 29% of its external edges, but approximately 28% of its nodes.

3.5 Evaluation and Conclusion

In order to determine the accuracy and usefulness of these measures, a comparison was made to a Rumor Spreading Algorithm. To investigate similarity, the algorithm initiates at a random node within a selected community and generates a random walk at each visited node until all nodes in the entire network or specific community have been visited. Though it can encounter a dead-end and never reach the target community in the presence of low average node degree and sparse data, rumor spreading is a reasonable way of indicating the closeness between communities of comparable size and connectivity on a small scale. As a result, it is an adequate measure for a baseline comparison. The results can be seen in the table below.

	1	2	3	4	5	6	7	8	9	10
1	-	0.18	0.154	0.228	0.273	0.197	0.338	0.365	0.167	0.335
2	0.182	-	0.15	0.227	0.273	0.196	0.338	0.365	0.167	0.349
3	0.153	0.148	-	0.189	0.234	0.157	0.299	0.326	0.129	0.315
4	0.228	0.227	0.188	-	0.302	0.229	0.396	0.396	0.199	0.388
5	0.273	0.273	0.234	0.301	-	0.273	0.414	0.441	0.244	0.433
6	0.197	0.196	0.1581	0.229	0.272	-	0.338	0.366	0.169	0.357
7	0.338	0.338	0.299	0.369	0.414	0.338	-	0.507	0.309	0.498
8	0.365	0.365	0.326	0.396	0.441	0.366	0.506	-	0.336	0.526
9	0.166	0.167	0.129	0.199	0.244	0.169	0.309	0.336	-	0.316
10	0.332	0.351	0.314	0.388	0.433	0.358	0.498	0.526	0.317	-

Table 2: Rumor Spreading

The values above represent the length of the path generated by moving from the starting community to every node in another specified community, with respect to the path generated by visiting the entire network from the starting community. The table shows similar trends to the original results: communities 1 and 3 are the least similar, there is a strong similarity between communities 10 and 8, as well as 7 and 8 etc. Although initial experiments show that the combination of the Jaccard Similarity Coefficient and the Edge Extension Algorithm has potential, further research is needed to explore the suitability for larger, real-world networks that utilize graph databases.

References

[1] Angles, R. and Gutierrez, C., 2008. Survey of graph database models. ACM Computing Surveys (CSUR), 40(1), p.1.Vancouver

[2] Eifrem, E., 2016. Graph databases: the key to foolproof fraud detection?. Computer Fraud & Security, 2016(3), pp.5-8.

[3] Miller, J.J., 2013, March. Graph database applications and concepts with Neo4j. In Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA (Vol. 2324, p. 36).

[4] Yang, J. and Leskovec, J., 2015. Defining and evaluating network communities based on ground-truth. Knowledge and Information Systems, 42(1), pp.181-213.

[5] Rawashdeh, A. and Ralescu, A.L., 2015, April. Similarity Measure for Social Networks-A Brief Survey. In MAICS (pp. 153-159).

[6] Granovetter, M.S., 1973. The strength of weak ties. American journal of sociology, pp.1360-1380.