Comparison Queries for Uncertain Graphs

Denis Dimitrov, Lisa Singh, and Janet Mann

Georgetown University
Washington, DC, USA 20057

Abstract. Extending graph models to incorporate uncertainty is important for many applications, including disease transmission networks, where edges may have a disease transmission probability associated with them, and social networks, where nodes may have an existence probability associated with them. Analysts need tools that support analysis and comparision of these uncertain graphs. To this end, we have developed a prototype SQL-like graph query language with emphasis on operators for uncertain graph comparison. In order to facilitate adding new operators and to enable developers to use existing operators as building blocks for more complex ones, we have implemented a query engine with an extensible system architecture. The utility of our query language and operators in analyzing uncertain graph data is illustrated using two real world data sets: a dolphin observation network and a citation network. Our approach serves as an example for developing simple query languages that enables users to write their own ad-hoc uncertain graph comparison queries without extensive programming knowledge.

Keywords: graph query language, comparison queries, uncertain graphs

1 Introduction

Traditional graph models can be extended to incorporate uncertainty about vertex / edge existence and attribute values. Applications data that could benefit from uncertain graph models, include disease transmission networks with disease transmission probabilities, observed social networks with node existence probabilities, and physical computer networks with associated reliability probabilities.

A variety of tools, databases, algorithms, and research consider analysis of probabilistic data [16,26,27]. Analysis of graph data is a subject of another large, but mostly unrelated research area [8, 9, 11–13]. We are interested in combining both in a general way to give users the capability to examine and compare uncertain graphs.

While uncertain graph analysis and comparison is relevant to a number of different domains, we focus on two settings in this work, uncertainty occurring during scientific observation and uncertainty resulting from data analysis.

Observational scientific data: Observational scientists study animal societies in their natural settings, often, with the purpose of understanding the social relationships and behaviors within the society. Such social network data can be captured as a graph, where nodes represent observed animals and edges between

nodes represent sightings of both animals together. A researcher observing a particular animal may be uncertain about its identification, features, or behavior. This uncertainty can be expressed as existence probabilities between 0 and 1, associated with nodes and/or edges, and represented as representing discrete probability distributions over the set of possible categorical attribute values. Analyzing and comparing uncertain graphs can be useful for answering questions about changes in animal sociality over time, similarities of local animal communities to the general animal population, differences among animal subgroups across locations, and observation bias across researchers.

Analysis output data: A second setting we consider involves machine learning algorithms generating uncertain graphs that can be used as the basis for prediction, generalization, and statistical analysis. One specific example is a *node labeling* algorithm. These algorithms can be used to predict the topics of each publication in a citation network, predict which customers will recommend products to their friends using a customer network, or predict the political affiliations of people in a social network [23]. They take as input a partially observed graph to predict the label (attribute value) of nodes in the graph. The output of such algorithms is an uncertain graph containing a probability distribution across the possible set of labels for each node. Comparing and contrasting these uncertain graphs to each other or to a ground-truth graph allows researchers to analyze the performance of different machine learning algorithms, experiment with a single algorithm under different assumptions, and examine the graph dataset by highlighting parts of data where the algorithms disagree in their predictions or perform poorly.

Contributions: Our focus is on uncertain graph analysis with an emphasis on comparison of them. The contributions of this work are as follows:

1. A basic SQL-like language, which incorporates uncertain graph analysis and comparison operators, while taking advantage of most of the SQL capabilities. The semantics of this proposed language is a combination between relational database and uncertain graph semantics, part of which is novel and part of which is necessary for the language to be applicable. Section 4 introduces the language.
2. A set of composable, comparative operators for uncertain graphs, where the previous literature focuses on single graph operators, on specific graph algorithms in the presence of uncertainty, or on operators for multiple certain graphs. We introduce operators for estimating similarity between graphs, nodes, edges, and their attributes, including finding a common subgraph that exists across two graphs containing edges with high certainty and identifying a set of nodes that have the same predicted node label across two uncertain graphs. Section 5 describes our proposed operators in more detail.
3. A novel system framework that (1) uses a combination of a layered and a service oriented architecture, and (2) is extensible, modular, and expandable, allowing for easy integration of new operations. The novelty of our design is the focus on extensibility and modularity. Traditionally, databases construct query trees whose set of possible operations is predefined. This design allows for query optimization by applying a set of rewrite rules. Our

approach provides a flexible mapping of operators to their implementation. The query engine can, therefore, support easy integration of new operators without affecting the existing ones and without requiring significant changes to the framework itself. Section 6 presents our high level system architecture emphasizing details related to operator extensibility and composition.
4. An implementation of our query framework and an empirical analysis using two real world data sets (presented in Section 7).

2 Related Literature

Storage, analysis, and manipulation of graph data is a vast area of interest for both the research community and the industry. In particular, there are multiple graph databases in existence: [2,4–7], to name a few, offering efficient data storage and access, as well as scalability and transaction management. The query options sometimes include proprietary APIs (Neo4j Traverser) or proprietary SQL-like query languages [6]; in other cases ([1, 2, 5–7]) there is support for Gremlin [3] or SPARQL [20]. SPARQL is a query language for the RDF format with similarities to our approach in terms of semantics and SQL-like syntax, including joins and the capability to retrieve and combine data from several graphs. In comparison to our language, SPARQL offers more flexible pattern matching, but is more restricted in that its standard data types are XML-based, its set of operators is more limited, and it does not offer extended SQL-like constructs such as MERGE BY and SPLIT BY. Gremlin is a relatively simple but powerful language for graph traversal and manipulation supporting built-in functions such as union, difference, intersection that are applicable to graph comparison. Neither of these languages focus on uncertain graphs. Similarly, many of the languages suggested in existing research [8,9,11–13] often do not consider uncertainty during graph analysis and comparison across multiple graphs.

Querying similar graphs in graph databases has been studied in recent years [30]; however, existing works mainly focus on structural information and connectivity. Uncertainty is often incorporated in the context of specific algorithms [14, 15, 18, 19, 24, 32]. These problems are important in answering some of the possible uncertain graph queries, yet our goal is to create a more comprehensive set of albeit simpler uncertain comparison operators. Other researchers study uncertainty arising from approximate queries rather than uncertain data [29].

Probabilistic databases, on the other hand, typically support queries based on the concept of Possible World Semantics [16,26,27]. Recently researchers have extended the Possible World Semantics to uncertain graphs [31], [28]. While applicable for many problems, this concept is different from our focus on graph comparison regardless of the nature of the underlying probabilities. We do, however, build upon operators for comparison of probabilistic attributes [26], as they are applicable for uncertain graph attributes.

Finally, there are a number of visual graph tools, including [23], [10]. Excellent for visual comparison of uncertain graphs, they could complement rather than substitute the capability to execute user-defined queries.

3 Probabilistic Formulation

Throughout the subsequent sections we use the following background definitions, underlying assumptions, and notation. The central object of interest are uncertain graphs that represent a generalization of exact graphs, incorporating uncertainty about vertex / edge existence and attribute values. The existence probability of an edge is assumed to be conditional upon the existence of its endpoints. Uncertain attributes contain probabilities associated with each possible value from their domain, expressing the likelihood that the attribute takes on this particular value. To broaden the application of our model, we make no assumptions about the nature of probability values assigned to the graphs that we need to compare. In other words, the analyst can decide if the probabilities are marginal or posterior.

Uncertain Graph. An uncertain graph $G = (V, E, A^V, A^E, PA^V, PA^E)$ has a non-empty finite set of vertices, $V = \{v_1, \dots v_m\}$, and a finite set of undirected edges, $E = \{e_1, \dots e_n\}$, where each edge e_y is a pair of vertices, $e_y \in V \times V$, and $V \times V = \{(v_i, v_j) | v_i \in V, v_j \in V\}$; $A^V = \{A_1, \dots A_p\}$ is a set of (certain) attributes for vertices; $A^E = \{A_1, \dots A_q\}$ is a set of (certain) attributes for edges; $PA^V = \{PA_1, PA_2, \dots PA_r\}$ is a set of uncertain attributes for vertices; and $PA^E = \{PA_1, PA_2, \dots PA_t\}$ is a set of uncertain attributes for edges. The attributes are consistent across vertices and across edges respectively, i.e. all vertices have the same schema and so do all edges. We refer to both edges and vertices as *graph elements*.

Certain Attributes. The set of all certain attributes is defined as $A = \{A_1, A_2, \dots A_s\} = A^V \cup A^E$. Given an attribute $A_j \in A^V$, its domain D_j, and a vertex $v_i \in V$, we associate a value $p_k \in \{D_j\}$ with the pair (v_i, A_j) and denote it using the notation $a(v_i, A_j) = p_k$. As shorthand, we use a_{ij}. Similarly, $a(e_i, A_j) = p_k$. To express structural uncertainty, we designate one of the attributes in A^V and A^E to store our confidence about existence of the corresponding graph element: $\exists A_j : a_{ij} \in [0, 1]$, $\forall i \in [1, m]$ and $\exists A_j : a_{ij} \in [0, 1]$, $\forall i \in [1, n]$. Henceforth, we refer to this attribute as 'conf' or 'confidence'.

Uncertain Attributes. By analogy, the set of all uncertain attributes is $PA = \{PA_1, PA_2, \dots PA_o\} = PA^V \cup PA^E$. Uncertain attributes allow the data model to express semantic uncertainty in the graph. The value of an uncertain attribute PA_j is a set of pairs of each possible attribute value and a probability associated with each possible value. For example, an uncertain attribute *sex* with value domain $\{male, female\}$ reflects the researcher's uncertainty about the sex of the observed animal. For a specific vertex, the set of its value pairs could be $\{(male, 0.8), (female, 0.2)\}$.

More precisely, value domain VD_j is the constrained (discrete) domain of possible values associated with attribute PA_j. The value domain is ordered and we use the notation a_j^t to designate the t-th member of VD_j, where $t \in [1, |VD_j|]$.

Given an uncertain attribute $PA_j \in PA^V$ and a vertex $v_i \in V$, we associate a value $p_k \in \{PD_j\}$ with the pair (v_i, PA_j) and denote it using the notation $pa(v_i, PA_j) = p_k$. As shorthand, we use pa_{ij}. Similarly, given an attribute $PA_j \in PA^E$ and an edge $e_i \in E$, we associate a value $p_k \in \{PD_j\}$ with the pair

(e_i, PA_j) and denote it using the notation $pa(e_i, PA_j) = p_k$. As shorthand, we again use pa_{ij}.

By analogy to using a_j^t to refer to members of value domain VD_j, the shorthand p_{ij}^t refers to the corresponding probability $f(a_j^t)$, associated with value a_j^t for vertex v_i. We define the set of uncertain attributes for a particular vertex v_i as $PA(v_i) = \{PA_j : PA_j \in PA^V \text{ and } pa(v_i, PA_j) \neq null\}$. Similarly, the set of uncertain attributes for a particular edge e_i is defined as $PA(e_i) = \{PA_j : PA_j \in PA^E \text{ and } pa(e_i, PA_j) \neq null\}$.

Assumptions. In comparing two graphs g1 and g2, we use the simplifying assumption that the following partial mapping exists between their elements: (1) the vertex mapping consists of a bijective mapping function for those vertices that are mapped, plus a set of unmapped vertices in each of the graphs $g1$ and $g2$; and (2) edge mapping is equivalent to vertex mapping, with the added constraint that edges $g1.e$ and $g2.f$ can be mapped to each other only if both of their endpoint vertices are also mapped. We refer to two graphs with this property as aligned graphs. Different alignment schemas may be possible. Unless stated otherwise, alignment is assumed to be based on element id: elements from graph $g1$ are mapped to elements with the same unique id in graph $g2$; they are unmapped if there is no corresponding element with the same id.

4 Query Language

Our approach was to take advantage of the SQL semantics by handling graphs, graph elements, and attributes using relations. We chose to base our query language on SQL because it is a mature, proven, and well-known language. While we do not claim that it is the best language for the purpose, we believe it is sufficient for expressing a wide range of uncertain graph comparison queries using our set of operators. Using these operators directly with traditional SQL was certainly another option; however, there are several disadvantages. On a syntactic and semantic level, a dedicated query language allows the flexibility for any modifications that best suit the specifics of uncertain graph analysis. On an implementation level, extending an existing SQL query engine effectively would mean using a relational database as storage for uncertain graph data. Instead, our goal was to develop an approach that borrows important SQL semantics without being restricted to a particular storage.

Our query language supports the major SQL operations, such as *SELECT*, *FROM*, *JOIN*, *WHERE*, *GROUP BY*, *HAVING*, and *ORDER BY*, introducing modifications and extensions to accommodate the specifics of graph comparison. For example, the *FROM* operation can extract individual nodes, edges, both nodes and edges, attributes from the specified graph (creating a tuple for each of them), or return the graph as a whole in a table as a single tuple. An example that returns a table containing edges from graph $g1$ with high existence probability along with the confidence of existence sorted by confidence is as follows:

```
SELECT e, conf(e)
```

```
FROM g1 TYPE edge AS e
WHERE conf(e) > 0.5
ORDER BY conf(e) DESC
```

We introduce two new operations, *MERGE BY* and *SPLIT BY*, to support collections that are returned by some common operators, such as $adjacentVertices()$. The *SPLIT BY c* operation is used to "flatten out" a relation, when column c may contain a set of values rather than a single value. In the returned relation, each tuple t in the source relation is replaced by a number of tuples, one for every value in the set of values from t in column c. *MERGE BY* is similar to *GROUP BY*, but retains the remaining columns, which are not part of the clause, "visible" to subsequent non-aggregate operators by collapsing the values from the original tuples into a collection.

While this approach has many strengths, the main advantages are the use of a familiar query language in SQL and the additional operation extensions for collections that improve the flexibility of querying uncertain graphs, without being restricted to a particular storage.

5 Proposed Operators

While we take advantage of the SQL-like operations to retrieve, filter, sort, group, and join data, individual operators are used within each of these clauses to specify the required behavior. As shown in the query example, the same operator can be re-used with several operations, subject to rules between aggregate vs. non-aggregate operators and operations. Because limited space precludes us from describing all operators within the text, we provide a list of operators, categorized based on their target, i.e. those that apply to attributes (Table 2), graph elements (Table 1), and ego-networks and graphs (Table 3). Each of these tables contains the name of the operator, a brief description, and in some cases, an example result based on the sample graph in Figure 1.

More specifically, because the attribute value represents a discrete probability distribution, the proposed *attribute operators'* functionality ranges from simply extracting the most / least probable value to analyzing the shape of the distribution. For example, $peakToNextDist()$ can be used to identify uncertain attributes with a dominant (peak) probability that significantly exceeds the probabilities for the remaining values. *Graph element operators* may be incorporated into a query such as the introductory example in section 4 to identify weak connections within a single graph or to compare the confidence of the corresponding elements across two aligned graphs, isolating the elements not only based on their low or high confidence, but also on whether the two graphs agree or differ significantly. Common usage examples of *graph* and *ego-net operators* are provided in Table 3.

We focus the remainder of this section on our similarity operators since they are most relevant to the uncertain graph comparison tasks outlined in the motivating examples.

Operator	Description
$conf()$	confidence of element's existence - $conf(v1) = 0.8$
$bin()$	true or false bin, corresponding to high or low $conf()$ relative to a threshold. Ex. $bin(v1, 0.5) =$ true
$compBin()$	"high", "opposite", or "low", depending on the relationship between the output of the bin() operator applied to each of the two operands. Ex. $compBin(v1, v2) = high$
$magnitudeDiff()$	difference between confidence of existence of 2 elements. Ex. $magnitudeDiff(v1, v2) = 0.2$
$diffSignificance()$	whether the absolute value of magnitude difference is above a threshold. Ex. $diffSignificance(v1, v2, 0.1) = true$
$valueCertaintyScore()$	average $maxValueCertainty()$ of all uncertain attributes of the element. Ex. $valueCertaintyScore(v1) = 0.6$
$sim()$	similarity score between 2 elements of the same type (vertices or edges), typically in the range [0, 1]

Table 1. Graph Element Operators: Most of these operators serve to query and analyze the confidence of existence of a single graph element or relative to another vertex or edge. Other operators in this group aggregate the results from attribute-level operators for the given graph element.

Uncertain Attribute Similarity. The $sim()$ operator is one of our novel operators for comparing uncertain attributes pa_{ij} and pa_{lj}. In the proposed set of measures, similarity is classified as either structural or semantic. The former identifies the similarity between the general shapes of the two distributions, ignoring the attribute values and their arrangement relative to each other. For example, attribute $\{(a, 0.8), (b, 0.1), (c, 0.1)\}$ should be considered structurally equivalent to attribute $\{(a, 0.1), (b, 0.1), (c, 0.8)\}$, as both have a dominant value (peak) of 0.8. We support two structural similarity measures, entropy ratio and absolute distance ratio, where the entropy ratio compares the distribution spread for the specified uncertain attribute and the absolute distance ratio compares the magnitude of the distance between the different uncertain attribute values. For example, the absolute distance is calculated as: $AD(pa_{ij}) = \sum_{t=2}^{|VD_j|} |p_{ij}^t - p_{ij}^{t-1}|$ and by analogy, for pa_{lj}. To correctly reflect structural similarity through absolute distance, probability sets in both attributes must first be sorted.

The semantic similarity, on the other hand, compares probabilities between the corresponding attribute values. An instance of an uncertain attribute can

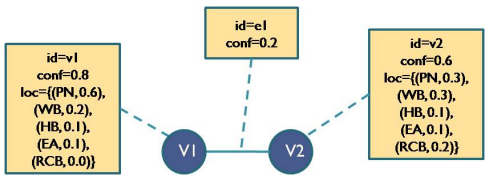

Fig. 1. Sample network for operator examples.

Operator	Description
$mpv()$, $lpv()$	most / least probable attribute value. Ex. $mpv(v1.loc) =$ "PN"
$valueCertainty()$	probability that the attribute has a specific value from the domain. Ex. $valueCertainty(v1.loc, \text{"}WB\text{"}) = 0.2$
$maxValueCertainty()$, $minValueCertainty()$, $avgValueCertainty()$, $medianValueCertainty()$	max, min, mean, or median probability among all probabilities associated with an attribute. Ex. $maxValueCertainty(v1.loc) = 0.6$
$peakToAvgDist()$	difference between the max and the average certainty. Ex. $peakToAvgDist(v1.loc) = 0.4$
$peakToNextDist()$	difference between the max and the second-highest attribute value probability. Ex. $peakToNextDist(v1.loc) = 0.4$
$valueCertaintyDev()$	standard deviation of probabilities for an uncertain attribute
$valueCertaintyRange()$	difference between highest and lowest probability of an attribute. Ex. $valueCertaintyRange(v1.loc) = 0.6$
$sim()$	similarity score between two uncertain attributes of the same type, typically in the range [0, 1]. It is generally measured between the two sets of their respective attribute values and probabilities. The specific similarity measures are described in this section.

Table 2. Uncertain Attribute Operators: These operators answer queries about values and probabilities associated with one or more uncertain attributes.

be represented as a histogram. We refer to each possible attribute value as a 'bin' in the histogram, conceptually containing the associated probability. This representation allows us to use a number of measures that have been proposed for histogram similarity. They generally fall into two categories [21]. The *bin-by-bin* similarity compares the contents of only corresponding bins, or in our case, probabilities for the same attribute values in two attribute instances. *Cross-bin* measures, on the other hand, compare non-corresponding bins. This is possible only if the *ground distance* between pairs of non-corresponding attribute values is known. In this work, we focus on the following bin-by-bin similarity measures [21]: (1) Default: $sim(pa_{ij}, pa_{lj}) = 1 - \frac{\sum_{t=1}^{|VD_j|} |p_{ij}^t - p_{lj}^t|}{2}$; (2) Minkowski-Form Distance; (3) Histogram intersection; (4) K-L divergence.

Ego network similarity The $egoSim()$ operator uses a variety of similarity measures and algorithms depending on user-specified constraints and on ego network containment within the same or different graphs. For measuring similarity between two ego networks (or ego-nets), the two center nodes are mapped to each other, each of the non-center nodes from the first subgraph is mapped to 0 or 1 non-center nodes from the second subgraph, and vice versa. Depending on existence and on alignment between the two ego-nets, similarity can be aligned and unaligned. In the aligned case, the mapping is determined by the alignment scheme. If no alignment scheme is chosen (not aligned case), the ego-nets' elements are mapped in a way that maximizes similarity.

Operator	Description
$egoNet()$	given a vertex v_i, returns the set of vertices and edges that are part of v_i's ego-network, including v_i itself
$egoSim(v_1, v_2)$	similarity score between two ego-networks defined by their center vertices v_1 and v_2, respectively, typically in the range [0, 1]
$intersect()$, $union()$, $difference()$, $bidirectionalDifference()$	creates a new graph that represents, respectively, an intersection, union, difference, and bidirectional difference of two graphs
$toGraph()$	recreates a graph from a set of vertices and edges
$toElements()$	breaks down a given graph into a set of vertices and edges

Table 3. Graph and ego-net operators: Operators allowing for structural graph comparison based on graph alignment include: $intersect(g1, g2)$, $union(g1, g2)$, $difference(g1, g2)$, and $bidirectionalDifference(g1, g2)$. For example, by intersecting the ego networks (subgraphs consisting of the starting node v_i (center), all of its adjacent nodes, and all edges between them) of two specific dolphins in the same graph, the analyst can discover their common friends. The graph reconstruction operator, $toGraph()$, can be used in a query to derive a subgraph based on specified conditions. For example, to obtain a subgraph of high-confidence elements, it can be combined with the $bin()$ operator and MERGE BY clause. While this operator is not as sophisticated as pattern matching [13], it does provide the possibility of subgraph filtering based on a flexible set of conditions.

Ego-net similarity can be structural, semantic or both. Structural similarity only takes into account the existence or confidence of existence of vertices and edges in each mapped pair between the two ego-nets, while ignoring attributes and their values. Semantic similarity, on the other hand, ignores confidence and derives the similarity score by only using similarity measures between the individual nodes and edges in the mapped pairs. Structural-semantic similarity is a combination of both.

We now intuitively describe different types of ego-net similarity. They are the cornerstone of our uncertain comparative operators, allowing researchers to better compare graph substructures, not just the entire graph or single graph elements. Due to space limitations, we cannot describe them more formally. We have an extended version of this paper that contains those details.

We propose the following three different structural similarity operators for uncertain graphs. *Topological similarity* compares the structure of the two ego-nets based on the existence of their elements, but not on confidence values associated with existence. *Probabilistic-topological similarity* takes into account the confidence values associated with edges and non-center vertices. *Comparison count* is simply a count of aligned non-center nodes between the two ego-networks. It is useful when the researcher is interested in an absolute similarity measure, related to the size of the ego-networks, rather than in a ratio between 0 and 1 that is returned by the topological and probabilistic-topological similarity.

Semantic similarity takes into account uncertain attribute values and probabilities, rather than confidence of existence of mapped vertices and edges. In the aligned case, similarity is measured by aggregating similarities between pairs of attributes with the same name and definition, belonging to each pair of aligned

vertices. In the unaligned case, alignment is chosen in an attempt to maximize the total similarity of all attribute pairs between aligned vertices - usually by greedy heuristics.

Depending on the number of attributes under consideration, the measure can be either single- or multiple-attribute. In both of those cases, similarity between a pair of uncertain attributes can be estimated using different measures. We propose two of them: mpv and distribution similarity. The former calculates the similarity between a pair of attributes to be 1 if there is either a full or a partial match between their sets of mpv values, and 0 otherwise. The latter represents the user's choice of one of the uncertain attribute similarity measures, described in a previous section. In the aligned case, attribute similarity is calculated as average pairwise similarity between mapped vertices.

In the unaligned case, we restrict the similarity measure to a single attribute for considerations of computational complexity. Even in the case of a single attribute, the brute force approach for finding the alignment that would maximize similarity is highly inefficient in some cases. In those cases, we propose using a greedy heuristics, similar to the merge-sort algorithm, that reduces running time but does not guarantee optimality. As in the aligned case, the user has a choice of two similarity measures, MPV and distribution based similarity.

Other operators: In addition to operators related to uncertain graph comparison, the proposed query language supports general operators, most of which are commonly present in many other languages, including SQL, e.g. aggregate operators, logical operators, set operators, etc.

Route operators: While path operators are central to graph query languages, their use is not as central as similarity for uncertain graph comparison. Some operators that are useful in this context include: comparing high confidence path existence between two nodes or ego-nets, comparing high confidence shortest paths, and comparing connected components when taking into account the confidence of existence of graph elements.

6 System Architecture

6.1 System overview

Our highest priority design goals in developing the query engine architecture and prototype implementation include:

Extensibility. Because we intend to continue building upon our initial query language, allowing for extensibility at all levels was our highest priority. At the lowest level, it must allow easy integration of any additional operators and operations. When concepts that do not fit in the existing implementation are introduced, for example, aggregate operators, it is desirable to minimize the required changes to the query processing framework. We refer to this as mid-level extensibility. At a high level, the design must provide room for new system capabilities, such as plugging in different data storage implementations.

Operator composition. Operators sometimes re-use the functionality of other existing operators. For example, vertex and ego-network similarity operators

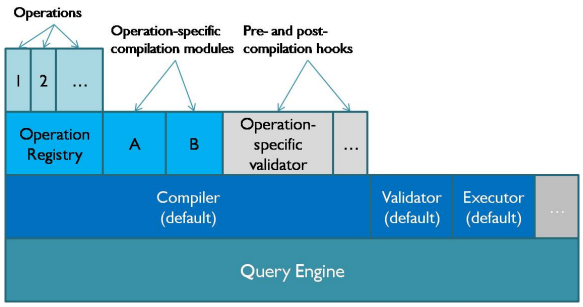

Fig. 2. Layered architecture.

build upon different attribute similarity operators. Because we anticipate that being a common situation, the system should provide re-use of existing operators to the programmer, who creates new operators. The user can also compose operators implicitly within the limits of the query language by creating expressions or within the limits of the pre-programmed sub-operator selections, such as choosing an underlying attribute similarity measure.

Adaptability. Capabilities to introduce future optimizations specific to our data model and query language without restricting the implementation to a particular platform or data storage.

To meet these goals, we use a combination of layered and service-oriented architecture, illustrated in Figure 2. The main component of this architecture is the query engine, a lightweight and generic platform for deployment of modules responsible for the individual steps in the query processing workflow, such as parsing, compilation, optimization, validation, and execution. The set of included modules, represented as services, is not pre-defined, making it different from traditional database query engines.

The engine offers two important capabilities: service configuration and service lookup. The former allows parameter tuning without code recompilation, including deploying the same implementation under different configurations. The latter allows flexible and dynamic linking of services, e.g. transparent replacement of the underlying data storage implementation. The individual modules are designed with the goal of decoupling them from each other and, in turn, they can be customized by plugging in implementations of their sub-components. For example, an operation registry is the sub-component that provides the default mapping between operators and their compiled representations. For integrating simple operators, it is sufficient to add a reference to the registry.

The query execution process is as follows. It begins when the Parser module transforms the textual representation of a query into an abstract syntax tree (AST), which we refer to as *logical query*. The Compiler module translates the AST using a post-order traversal of operations in the logical tree into an internal representation suitable for optimization and execution. Next, the Optimizer module generates and evaluates several alternative execution plans, choosing the best one. The Executor is the key module where the operations and operators that make up the query are executed. The Validator module can be invoked

at different stages to ensure compliance with the pre-defined rules. The Facade and Connector modules provide the interface for interaction between external systems and the query processing workflow. Data Store serves to retrieve the data requested in the query. Due to space limitations, we cannot go through the details of each step. Instead, we focus on how one develops and composes new operators given the extensible system design.

6.2 Developing and Composing Operators

Traditionally, databases construct query trees whose set of possible operations is predefined. This design allows for query optimization by applying a set of rewrite rules. Our approach differs because we provides a flexible mapping of operators to their implementation. This allows the query engine to support easy integration of new operators without affecting the existing ones and without requiring significant changes to the framework itself.

Developing an operator involves implementing a simple interface with two methods. The first method allows the Executor to set the operator's input parameters. Then, the second method is called, in which the operator performs its calculations over these supplied parameters and returns the result. In the simple case, no other code is required. Adding the operator to the configuration of the OperationRegistry is sufficient to incorporate it into the query language, as the registry is used for both compilation and execution. Composing an operator using operators that are already in the language is also straightforward, as the framework supports their lookup and execution from the dependent operator.

7 Empirical Evaluation

To show the utility and composition ability of our operators, we have integrated our query engine with Invenio [25], a visual analytic tool for graph mining. We use the two motivating scenarios presented in section 1 to highlight a subset of our operators.

7.1 Dolphin observation network

The Shark Bay Research Project studies dolphins in Shark Bay, Australia for over 30 years [17]. Our data set includes demographic data about approximately 800 dolphins, represented as graph nodes with certain attributes (id, conf, dolphin name, birth_date) and uncertain attributes (sex_code, location, mortality_status_code). Survey data about social interactions between these dolphins are captured as approximately 29,000 edges with attributes (id, conf).

Our team met with researchers on this project and developed a list of typical queries that observational scientists would like the capability to issue when analyzing these dolphin social network and its inherent uncertainty:

- Selecting the number of associates and sex composition of associates for male and female dolphins, respectively, using the most probable value of the *sex_code* attribute.

– Visualizing the union, intersection, difference, and bi-directional difference between the ego-networks of a particular dolphin during two different years, where the confidence of relationship existence is above a specified threshold.
– Finding the common associates (friends) of two specific dolphins with a relationship confidence above a certain threshold.
– Finding all dolphins having associates whose most probable location is different from their own.
– Calculating a measure of structural and semantic similarity between ego-networks of two particular dolphins.
– Selecting the subgraph that consists only of dolphins linked by observations with low confidence of existence (lower than a specified threshold). The results of this query tell researchers if observers are having difficulty identifying certain dolphins.

The query in Table 4 is an example that shows counts by sex of dolphins seen together (task 1). The inner query selects pairs of dolphins seen together and uses *SPLIT BY* to split into a set of rows the collection that is returned by the $adjacentVertices()$ operator. The outer select produces counts for each possible sex combination, grouping the nodes based on the most probable sex_code. Researchers can use the resulting table to see that dolphins who are most probably males are seen together more often than any of the other combinations.

```
SELECT sex, sexAdj, count(adj) AS cntFriends
FROM (
   SELECT *
   FROM
   (
      SELECT n, adjacentVertices(n) AS adj
      FROM g1 TYPE node AS n
   )
   SPLIT BY adj
   )
GROUP BY first(mpv(n.sex_code)) AS sex,
   first(mpv(adj.sex_code)) AS sexAdj
```

MALE	MALE	9930
MALE	FEMALE	6184
FEMALE	MALE	6184
FEMALE	FEMALE	6092

Table 4. Sample query and its result.

The second task focuses on determining the union, intersection, difference, and bi-directional difference between the ego-networks of a particular dolphin during two different years. It introduces a time component. The results for a particular dolphin are displayed in graph format in Figure 3. It is easy to see that the dolphin has almost as many new associates as repeat associates, i.e. occurring during both years. Researchers can then visually explore who these associates are, what sex they are, etc., to gain more insight about dolphin sociality.

To validate the significance of our similarity operators, we evaluate one of the more complex measures. We estimate the ego-network semantic similarity between dolphin *JOY* and other dolphins in the same graph, in absense of alignment, using the most probable location attribute. By picking dolphins with dif-

ferent characteristics, we can demonstrate the behavior and validity of the chosen similarity measure. For example, we discovered that the average ego-net similarity by location is twice as high for dolphins located in the same area as JOY, e.g. RCB: 0.28 vs 0.14. This is the expected result, since dolphins are likely to have associates mostly in their primary location.

To compare uncertain ego-nets, we randomly chose several dolphins from different locations with high and low similarity relative to *JOY's*. For every dolphin under consideration, we ran a query to retrieve their most probable location, their ego-network location similarity to that of *JOY*, and a breakdown by location of the dolphin's ego-network. The results are summarized in Table 5. They are consistent across the two cases of same and different location. Both *LITTLE* and *WHELK* differ from *JOY* in their most probable location; however, the ego-network's location composition between *LITTLE* and *JOY* results in a much higher similarity score. *PUCK* and *JOYSFRIEND* reside in the same location as *JOY*, share many associates with her, and have a very similar distribution of associates by location. These commonalities lead to a particularly high similarity score. *MYRTLE*, on the other hand, who only shares 88 out of 147 associates with *JOY* despite the same location, is average in similarity. For *WANDA*, the most probable location is a tie between WB and RCB, which is also reflected in having associates from mostly those locations. This difference with *JOY's* ego-network again corresponds to the lower similarity.

Overall, examining different cases confirms that the similarity measure provides a relevant single numeric value that correlates with the semantic composition of a pair of ego-networks based on the chosen attribute. Researchers can use this simple result to identify and rank potentially similar ego-networks.

The query also shows operator re-use and composition from the user's perspective. By supplying context parameters, the user configures the general ego-net similarity operator. Specifying the mpv-based similarity measure and attribute name causes the similarity operator to re-use the *mpv*() operator to retrieve the most probable attribute value.

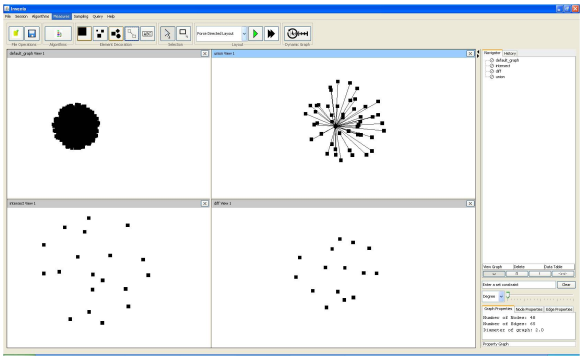

Fig. 3. Clockwise from upper left: complete dolphin network, union, intersection, difference of ego-networks of dolphin 'JOY' between years 2010 & 2009

	JOY	LITTLE	WHELK	PUCK	JOYSFRIEND	MYRTLE	WANDA
RCB	211	125	1	171	172	92	56
EA	38	5	43	32	47	6	7
WB	28	48		7	8	45	33
HB		4				1	4
PN						5	
sim with JOY		0.56	0.14	0.75	0.78	0.44	0.32
primary loc	RCB	WB	EA	RCB	RCB	RCB	WB, RCB

Table 5. Ego-network similarity results.

7.2 Citation network

In the second scenario, we examine the output of two different node labeling algorithms. For this analysis, we use the CiteSeer paper citation data set from [22]. It consists of 3312 scientific publications classified into one of six topics. In the citation network each publication is a node and each citation is an edge. We use partially observed citation data to predict the probability distribution of the topic attribute of each paper by applying two different classification algorithms. The queries of interest deal with understanding the similarities and differences between most probable node labels across the two classification algorithms and include:

- Selecting the papers, whose topic certainty is significantly higher in one uncertain graph when compared to the other.
- Selecting the papers, for which the predicted discrete probability distribution differs the most between the two graphs, using different attribute similarity measures, e.g. KL divergence, Minkowski-form distance, and histogram intersection.
- Counting the number of papers that are misclassified by both models.
- Selecting the papers, which are misclassified with high confidence by both classifiers.

Due to space limitations, we cannot show the actual queries in the paper. Some of the results we found using this data set are as follows: model 1 misclassified fewer documents (83) than model 2 (103); of the documents misclassified by both classifiers (65), both models misclassify them with the same label; and 8 of the 10 largest ego neworks were in the area of information retrieval.

Because our experiments focus on expressibility, we do not include a performance evaluation. Yet it should be noted that all of the presented queries were completed within a couple of seconds on a personal computer with a moderate dual core 2.4 GhZ processor and 4 GB of memory. Optimizing performance is an important direction for future research.

These usage examples demonstrate how our language and implementation enable scientists to formulate a wide range of ad-hoc queries that can be used to analyze and compare uncertain graphs without the need for custom programming.

8 Conclusions

A need exists to compare graphs with uncertainty using a flexible set of operators that consider the graph structure, the graph semantics, and the inherent uncer-

tainty in the application domain. In this paper, we develop a query engine with a SQL-type language, set of comparative operators for uncertain graphs, and an extensible system framework that combines layered and service-oriented architecture. Our language combines elements of relational, uncertain, and graph databases, with operators introduced especially for the purpose of uncertain graph analysis and comparison. We also present complimentary use cases and empirically illustrate their utility on two real world data sets.

There are many future directions, including optimizations on certain similarity measures, efficient measures for unaligned graphs, and measures that combine both similarity and routing.

9 Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable feedback, Prof. Bala Kalyanasundaram and Prof. Richard Squier for helpful comments during the development of the system, and the Shark Bay Dolphin Research Project (SBDRP). This work was supported in part by NSF Grants #0941487 and #0937070 and ONR Grant #10230702.

References

1. Arangodb graph database. http://www.arangodb.org/.
2. Dex graph database. http://www.sparsity-technologies.com/dex.
3. Gremlin language for graph traversal and manipulation. https://github.com/tinkerpop/gremlin/wiki.
4. Neo4j graph database. http://neo4j.org/.
5. Oracle spatial and graph option. http://www.oracle.com/technetwork/database-options/spatialandgraph/overview/index.html.
6. Orientdb document-graph dbms. http://www.orientechnologies.com/.
7. Titan graph database. http://thinkaurelius.github.com/titan/.
8. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1:68–88, 1997.
9. R. Angles and C. Gutierrez. Survey of graph database models. *ACM Computer Surveys*, 40:1:1–1:39, 2008.
10. N. Cesario, A. Pang, and L. Singh. Visualizing node attribute uncertainty in graphs. In *SPIE Proceedings on Visualization and Data Analysis*, 2011.
11. S. Fortin. The graph isomorphism problem. Technical report, 1996.
12. R. H. Güting. GraphDB: Modeling and querying graphs in databases. In *VLDB*, 1994.
13. H. He and A. K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In *ACM SIGMOD*, 2008.
14. R. Jin, L. Liu, and C. C. Aggarwal. Discovering highly reliable subgraphs in uncertain graphs. In *ACM SIGKDD*, 2011.
15. R. Jin, L. Liu, B. Ding, and H. Wang. Distance-constraint reachability computation in uncertain graphs. *Proc. VLDB Endow.*, 4(9):551–562, June 2011.

16. C. Koch. *Managing and Mining Uncertain Data*, chapter MayBMS: A system for managing large uncertain and probabilistic databases. Springer, 2009.
17. J. Mann and S. B. R. Team. Shark bay dolphin project. http://www.monkeymiadolphins.org, 2011.
18. O. Papapetrou, E. Ioannou, and D. Skoutas. Efficient discovery of frequent subgraph patterns in uncertain graph databases. EDBT/ICDT, 2011.
19. M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. k-nearest neighbors in uncertain graphs. *Proc. VLDB Endow.*, 3:997–1008, 2010.
20. E. PrudHommeaux and A. Seaborne. Sparql query language for rdf. W3C recommendation 15, 2008.
21. Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vision*, 40:99–121, November 2000.
22. P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.
23. H. Sharara, A. Sopan, G. Namata, L. Getoor, and L. Singh. G-PARE: A visual analytic tool for comparative analysis of uncertain graphs. In *IEEE VAST*, 2011.
24. D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *PODS*, 2002.
25. L. Singh, M. Beard, L. Getoor, and M. B. Blake. Visual mining of multi-modal social networks at different abstraction levels. In *Information Visualization*, 2007.
26. S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah. Orion 2.0: native support for uncertain data. In *ACM SIGMOD*. ACM, 2008.
27. J. Widom. *Managing and Mining Uncertain Data*, chapter Trio: A system for data, uncertainty, and lineage. Springer-Verlag, 2009.
28. Y. Yuan, L. Chen, and G. Wang. Efficiently answering probability threshold-based shortest path queries over uncertain graphs. DASFAA, pages 155–170, 2010.
29. H. Zhou, A. A. Shaverdian, H. V. Jagadish, and G. Michailidis. Querying graphs with uncertain predicates. In *ACM Workshop on Mining and Learning with Graphs*, 2010.
30. Y. Zhu, L. Qin, J. X. Yu, and H. Cheng. Finding top-k similar graphs in graph databases. In *EDBT*, 2012.
31. Z. Zou, H. Gao, and J. Li. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 633–642, New York, NY, USA, 2010. ACM.
32. Z. Zou, J. Li, H. Gao, and S. Zhang. Finding top-k maximal cliques in an uncertain graph. *ICDE*, 2010.