Mining the Space of Graph Properties

Glen Jeh
glenj@db.stanford.edu
Stanford University

Jennifer Widom
widom@db.stanford.edu
Stanford University

Abstract

Existing data mining algorithms on graphs look for nodes satisfying specific properties, such as specific notions of structural similarity or specific measures of link-based importance. While such analyses for predetermined properties can be effective in well-understood domains, sometimes identifying an appropriate property for analysis can be a challenge, and focusing on a single property may neglect other important aspects of the data. In this paper, we develop a foundation for mining the properties themselves. We present a theoretical framework defining the space of graph properties, a variety of mining queries enabled by the framework, techniques to handle the enormous size of the query space, and an experimental system called *F-Miner* that demonstrates the utility and feasibility of property mining.

1 Introduction

Graph analyses have been used for a variety of applications to analyze interrelationships among entities. Some of these analyses concern standard graph-theoretic properties, such as the radius of the graph or embedded cliques. Other analyses yield high-level, subjective information about the data. For example, the web graph has been analyzed using the *PageRank* [20] and *HITS* [16] algorithms to identify web pages likely to be deemed "important" by the user. The citation structure of scientific papers has been analyzed to find papers related to a given paper [13, 15, 21].

These techniques have in common that they analyze graph structures for predetermined properties. Although such analyses can be very effective, coming up with a good property for analysis is often a challenge, especially when little is known about the data to begin with. Moreover, by fixing specific properties for analysis, other important aspects of the data may be ignored. Therefore the *space of properties* itself should be explored.

As a concrete example, consider the simple case of looking for intuitively "similar" nodes in the graph of Figure 1.

This work was supported by the National Science Foundation under grant IIS-9817799 and by a Junglee Stanford Graduate Fellowship.

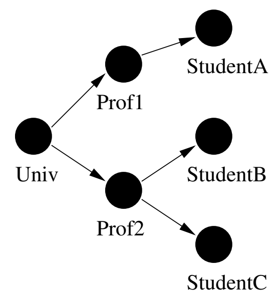

Figure 1: Graph example.

One possibility is to conclude that `Prof1` and `Prof2` are similar because they are both pointed-to by `Univ`, as in the commonly used *co-citation* metric [21]. Analogously, we may conclude that `StudentB` and `StudentC` are similar because they are both pointed-to by `Prof2`. On the other hand, we may argue that `StudentA`, `StudentB`, and `StudentC` are all similar because they are pointed-to by a node that is pointed-to by `Univ`, as in the recursive *SimRank* metric introduced in [13]. Each or all of these inferences may be valid, depending on the domain and application. However, current methods require the user to fix one measure of similarity (e.g., co-citation or SimRank) and query for nodes found to be similar under this measure. Ideally, we would like to query simply for "similar" nodes and get as a result the sets {`Prof1`, `Prof2`}, {`StudentB`, `StudentC`}, and {`StudentA`, `StudentB`, `StudentC`} along with explanations for why they are similar. This functionality is not supported by any current system we know of. It is supported by *F-Miner*, an implementation of the framework to be presented.

In this paper, we develop a framework for mining "interesting" or "important" graph properties. Essentially, we treat the space of properties as a domain and perform data mining on this domain. Our goal is to develop an appropriate analysis for mining the space of properties, just as analyses have been developed for mining the graph data itself. An obvious challenge is in handling the enormous size of the space of properties, in which even the simplest data mining operations seem hopelessly infeasible. We develop techniques that allow computational resources to be focused on only the most important properties, allowing us

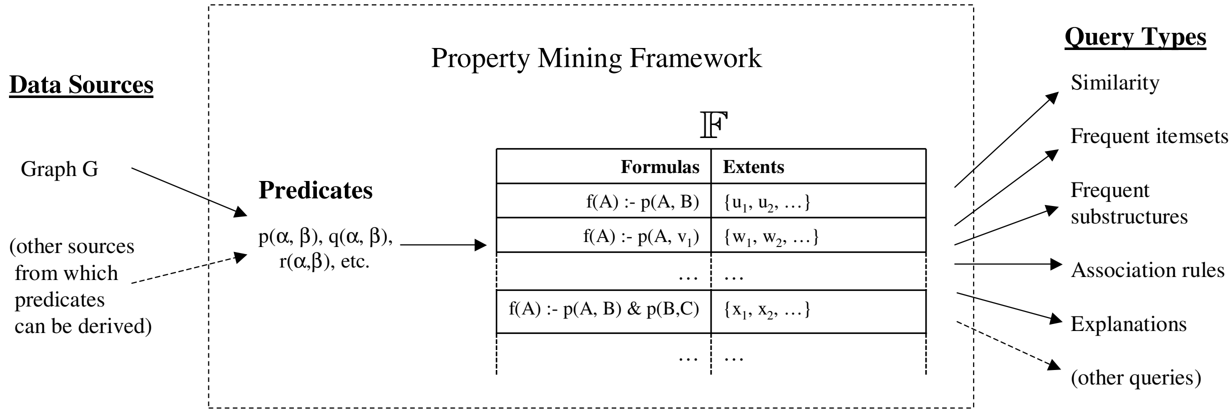

Figure 2: A logical diagram of the property mining framework, as seen by the user.

to implement a practical mining system based on the framework.

The main contributions of this paper are:

- A theoretical framework that defines the space of graph properties to be queried.
- Several specific data mining query types enabled by this framework, offering new capabilities not supported by existing systems.
- Techniques for dramatically reducing the computational resources required to answer queries within the framework.
- A simple and intuitive metric for calculating the "importance" of properties.
- The *F-Miner* experimental system, an implementation of the framework presented, demonstrating its utility and feasibility.

2 Framework for Property Mining

Before we can pose queries on the space of graph properties, we first have to define precisely what this space is.

- We begin with a labeled directed graph representing the objects and relationships in the domain we want to mine. Each edge represents a basic relationship between two objects.
- We encode properties as *formulas* in the syntax and semantics of *Datalog* [23]. Each property (formula) is composed of the basic relationships (*predicates*) in the domain.
- We consider the set $\mathbb{F}$ of all formulas and their *extents*, the objects which satisfy the formula. Queries on the space of properties can be stated as queries on $\mathbb{F}$.

A logical diagram of the framework, as seen by the user, is shown in Figure 2. The property mining framework itself is enclosed in the dashed box. The framework enables querying on properties of the input data (in our case a graph), from which a set of predicates representing basic relationships are derived. It is possible to derive predicates from sources other than graphs, such as the less-than relationship in a numeric data set, but for concreteness we limit ourselves in this paper to discussing predicates corresponding to graph edges. The predicates form the basis of formulas, and the user's queries posed are on the set of all formulas and their extents $\mathbb{F}$. Of course, $\mathbb{F}$ is only an abstraction provided to the user; in most cases $\mathbb{F}$ would never be materialized by an implementation. We define $\mathbb{F}$ formally in the next section.

2.1 The Query Space

Let $G = (V, E)$ be a labeled directed graph, where $E \subseteq V \times V \times \mathtt{L}$, for an arbitrary set $\mathtt{L}$ of strings serving as edge labels. For simplicity, we do not consider edge weights, although they can be added to the framework with slight modification. The fundamental relationships encoded in the edges are building blocks for the properties we will consider. In many domains there is an obvious canonical representation of the data as a graph, although in some domains the representation may require some consideration. As an example, consider the data shown in Figure 3, which is a small fragment of a survey of members of the Stanford Database Group. The obvious representation of the data as a labeled graph is also shown in the figure. Many data types can readily be modeled as a graph, including data in relational and XML format.

The next step is to represent properties of the domain as *formulas*. We use formulas in the language of *Datalog* [23], although arbitrary logic formulas, say of first order logic, are also possible. Clearly, the more powerful the logic, the greater the semantic and computational complexity of the system. Datalog has expressive power far beyond

```
      Sriram
Position: PhD Student
Advisor: Hector
Home: India
Food: Chinese, Indian

      Hector
Position: Faculty
Home: Mexico
Food: Indian, Thai
Research: Digital Libraries
```

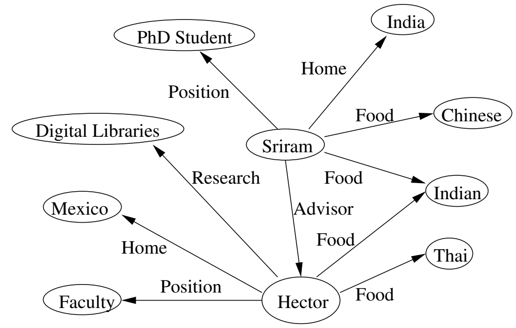

Figure 3: Sample data and its graph representation.

what we can hope to support in practice. As we shall see, even the very restricted subset of Datalog that we use allows functionality well beyond what existing systems can support. We will define the syntax and semantics of our formula language as we go along.

The *constants* (or *objects*) of the logic are the nodes in G (i.e., objects of the domain), while predicates correspond to edges (i.e., relationships of the domain). We consider Datalog *formulas* f of the form:

$$f(A) \coloneq \mathtt{p}_1(\alpha_1, \beta_1) \,\&\, \cdots \,\&\, \mathtt{p}_k(\alpha_k, \beta_k)$$

for $k \geq 1$, where $\mathtt{p}_i \in \mathtt{L}$, A is the *head variable*, and each α_i, β_i is either a variable or a constant (object). We use capital letters to denote variables, lowercase letters to denote constants, and Greek letters to denote either. Each *predicate* $\mathtt{p}(\alpha, \beta)$ corresponds to the existence of an edge with label $\mathtt{p}$ between its two arguments: predicate $\mathtt{p}(u, v)$, for $u, v \in V$, is *true* if and only if the edge $\langle u, v, \mathtt{p} \rangle$ exists in G. In the example of Figure 3, the predicate $\mathtt{Food}(\mathtt{Sriram}, \mathtt{Indian})$ is true. A formula f is *satisfied* by the *satisfying assignment* a, a function that maps variables to constants, if all the predicates in f are true when all variables X in f have been replaced by $a(X)$. In a formula, at least one of the arguments of each predicate must be a variable, for otherwise the predicate would have a constant truth value. Two formulas that are identical except for variables names and predicate ordering are considered identical.

The most important aspect of a formula f is its *extent*, denoted $\mathtt{E}(f)$. It is the set of all objects v for which there exists a satisfying assignment a such that $a(A) = v$ (recall that A is always the head variable). Intuitively, each formula specifies a "property" of the domain, and the extent of the formula is the set of objects which satisfy the property. For example, the formula

$$f(A) \coloneq \mathtt{Advisor}(B, A) \,\&\, \mathtt{Food}(B, \mathtt{Chinese})$$

encodes the property "being the advisor of someone who likes Chinese food", and its extent is $\{\mathtt{Hector}\}$.

We define the set $\mathbb{F} = \{(f, \mathtt{E}(f)) \mid f \text{ is a formula}\}$ of all formulas and their extents as the space in the context of which we pose our queries. A fragment of $\mathbb{F}$ is shown in Figure 2 as a relation. Of course, $\mathbb{F}$ is infinite, and is not meant to be computed. It serves only as the logical relation over which queries are posed.

2.2 Sample Query Types

There are many interesting data mining operations one can perform over $\mathbb{F}$, and many common notions in data mining, such as the *frequent itemsets* computation [3], have analogues in the space of formulas. Here are some examples:

- **Object similarity**. Two or more objects can be considered similar if they satisfy "many" formulas in common. In contrast to previous approaches, now similarity can be computed based on multiple aspects (as in the example of Section 1), with the system automatically identifying the more "important" aspects (Section 4).

  Furthermore, similarity can be *explained*. Rather than simply returning a score, the formula(s) which contributed to the score can be returned as an explanation of the results.

  Finally, "find more" queries can be supported. Given a set of objects U presumed to be similar in some ways, we can query for objects similar to objects in U in the same ways that objects in U are similar to one another. The objects returned are those which occur frequently in the formulas satisfied by members of U.

  These features are supported in the F-Miner system (Section 5).

- **Frequent itemsets**. We can run a *frequent itemsets* computation on the set of all extents, identifying those sets $U \subseteq V$ for which U is a subset of "many" extents. Each subset U represents a group of objects having "a lot" in common. For example, in the survey data to be discussed in Section 5, the set of all professors may be a frequent itemset because they all have advisees and have written many papers.

- **Frequent substructures**. Many graph structures correspond to formulas, and vice versa. We can find frequent substructures in the graph by finding formulas which have large extents. For example, the formula

$$f(A) :\!-\, \mathtt{e}(A,B)\ \&\ \mathtt{e}(B,C)\ \&\ \mathtt{e}(C,D)$$

corresponds to a path of length 3 (with edge labels `e`). If the extent of f is large, the graph contains many paths of length 3.

- **Association rules**. We can look for *association rules* of the form $f \implies g$, where f and g are formulas whose extents have "a lot" of overlap. The rule says that objects which satisfy f also satisfy g, or at least with a high probability.
- **Explanations**. Given a set of objects U, we can "explain" what the objects in U have in common with one another by considering the formulas they satisfy in common. This functionality is demonstrated in F-Miner (Section 5).

These are but some examples of the kinds of queries that can be posed on $\mathbb{F}$. We use the examples as motivation for some of the techniques we develop, but it is important to remember that these queries are end-applications of the framework, which itself consists only of the logical relation $\mathbb{F}$. Other queries are possible in the framework and some may be more application-specific. In contrast to traditional data mining, the queries here generally focus on the properties themselves, rather than the objects which satisfy the properties.

As a reminder, it is not possible (nor necessary) to materialize $\mathbb{F}$ in order to query it. Rather, once a specific application has been determined (e.g., finding similar objects), we can solve the end-to-end problem without explicitly constructing $\mathbb{F}$.

2.3 Challenges

There are two major technical challenges to answering queries within our framework. The first is in dealing with the enormous size of the query space. We develop techniques to handle this problem in Section 3. The second challenge is in determining the "importance" of formulas. This is a major component of property mining: just as we look for important (interesting) objects in traditional data mining, here we look for important properties. Moreover, identifying important formulas is intertwined with reducing the space considered, since we would like as much as possible to restrict ourselves to considering only the most important formulas. Computing importance of formulas is discussed in Section 4.

3 Building Blocks Approach

The query space $\mathbb{F}$ is infinite, and it is impossible to consider all formulas in $\mathbb{F}$. Instead, we want to focus as much of our computational resources as possible on the most important formulas. This poses a dilemma, since one of the goals of our mining is to identify these important formulas!

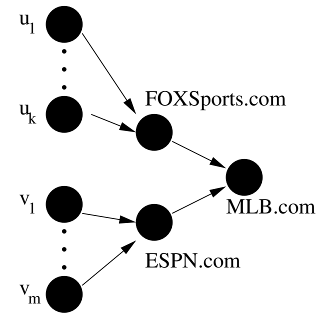

Figure 4: Graph example.

At a high level, our solution is to construct formulas from basic building blocks called *pseudopredicates*. We analyze these building blocks for importance instead of analyzing the actual formulas. This allows us to determine the importance of the formulas that can be constructed from the building blocks, so that only the most important formulas are ever created. Hence much of the mining actually takes place in the space of pseudopredicates. Our approach can be broken into 3 steps:

1. A set of *pseudopredicates* is computed to serve as basic building blocks for formulas.
2. Importance scores are computed for the pseudopredicates, identifying those from which important formulas can be constructed.
3. Important formulas are constructed from the pseudopredicates.

In this section we present steps (1) and (3). Step (2) is presented in Section 4.

3.1 Example

We motivate our approach using an example. Consider the top half of the structure shown in Figure 4, a hypothetical fragment of the web graph. The web pages $u_1, \ldots, u_k$ all point to `FOXSports.com`, so they satisfy the formula

$$f(A) :\!-\, \mathtt{e}(A, \mathtt{FOXSports.com}) \tag{1}$$

where `e` is taken to be the label of every edge in the (unlabeled) graph. `FOXSports.com` in turn points to baseball's `MLB.com`, so `FOXSports.com` satisfies the formula

$$g(A) :\!-\, \mathtt{e}(A, \mathtt{MLB.com}) \tag{2}$$

Finally, $u_1, \ldots, u_k$ all satisfy

$$h(A) :\!-\, \mathtt{e}(A,B)\ \&\ \mathtt{e}(B, \mathtt{MLB.com}) \tag{3}$$

It seems redundant to record this fact, however, since it follows immediately from the facts that u_i, for all $1 \le i \le k$, satisfies formula (1) and that FOXSports.com satisfies formula (2). More generally, for any formula g satisfied by FOXSports.com, each u_i points to a node that satisfies g. That is, u_i satisfies the formula

$$h(A) :\!- \, \mathtt{e}(A, B) \,\&\, g(B)$$

where $g(B)$ is g with head variable A replaced by a variable B not appearing already in g.[1]

Now we consider the entire Figure 4, where $v_1, \dots, v_m$ all point to the sports site ESPN.com. Since ESPN.com points to MLB.com, $u_1, \dots, u_k$ and $v_1, \dots, v_m$ are all related by their common satisfaction of formula (3), a consequence of the fact that the u_i's and v_j's all point to either FOXSports.com or ESPN.com. We record this by saying that u_i's and v_j's satisfy the *pseudoformula*

$$f(A) :\!- \, \mathtt{e}(A, \{\mathtt{FOXSports.com}, \ \mathtt{ESPN.com}\})$$

3.2 Pseudopredicates and Pseudoformulas

A *pseudopredicate* is a predicate that may have as an argument a nonempty set of objects, as well as variables. It is a generalization of a regular predicate, which can be thought of as the special case when the only set-arguments of a pseudopredicate are singleton sets. We have already seen one example of a pseudopredicate in Section 3.1, $\mathtt{e}(A, \{\mathtt{FOXSports.com}, \mathtt{ESPN.com}\})$, which represents the property of pointing to either FOXSports.com or ESPN.com. We define a *pseudoformula* as a formula consisting of pseudopredicates, and define the extent of a pseudoformula f as the set of objects v for which there exists a satisfying assignment a for f such that $a(A) = v$, where a assigns each set-argument to one of its members. For example, the extent of the pseudoformula

$$f(A) :\!- \, (A, \{\mathtt{FOXSports.com}, \ \mathtt{ESPN.com}\})$$

is $\{u_1, \dots, u_k, v_1, \dots, v_m\}$.

Note that the set of formulas can be thought of as a subset of the set of pseudoformulas, since we can replace each constant argument v by $\{v\}$ to get a pseudoformula having the same semantics. Conversely, some pseudoformulas can be converted to formulas: if all set-arguments of a pseudoformula f are either singleton sets or the set of all objects V, then f has the same semantics as the formula f' which is a copy of f except with singleton objects replaced by their sole members, and each set-argument V replaced by a *dangling* variable not appearing anywhere else in f'.

3.3 Chaining Pseudoformulas

We take the set of *basic building blocks* to be $(\mathbb{P}, \mathtt{E}(\mathbb{P}))$, the set of all head pseudopredicates $\mathbb{P}$ and their extents $\mathtt{E}(\mathbb{P})$. A *head pseudopredicate* is a pseudopredicate whose two arguments are the head variable A and a set-argument $S \subseteq V$. These pseudopredicates can be treated as 1-predicate pseudoformulas. In the coming sections, we will talk about extents of head pseudopredicates as though they were 1-predicate pseudoformulas, and omit the "head" qualification when the meaning is clear.

From this base set of head pseudopredicates we can compose a large class of more complex pseudoformulas and thus formulas, which as noted before are a subset of pseudoformulas. The two composition steps are *conjoining* and *chaining*.

Conjoining two formulas f and g creates a new formula h whose predicates are a conjunction of the predicates in f and g, with appropriate renaming of non-head variables to avoid conflict. For example:

$$\begin{aligned}
f(A) \quad &:\!- \quad \mathtt{Advisor}(B, A) \,\&\, \mathtt{Food}(B, \mathtt{Chinese}) \\
g(A) \quad &:\!- \quad \mathtt{Advisor}(B, A) \,\&\, \mathtt{Home}(B, \mathtt{India}) \\
h(A) \quad &:\!- \quad \mathtt{Advisor}(B, A) \,\&\, \mathtt{Food}(B, \mathtt{Chinese}) \\
&\qquad \,\&\, \mathtt{Advisor}(C, A) \,\&\, \mathtt{Home}(C, \mathtt{India})
\end{aligned}$$

Chaining is a formalization of the example in Section 3.1 of deriving formula h from f and g. Suppose we have a 1-predicate pseudoformula $f(A) :\!- \, \mathtt{p}(A, S)$. Then for any pseudoformula g whose extent is a superset of S, any object satisfying f also satisfies the pseudoformula $h(A) :\!- \, \mathtt{p}(A, B) \,\&\, g(B)$. We say that h is the result of *chaining* f and g. In the general case, if S appears as a set-argument in f, and $S \subseteq \mathtt{E}(g_i)$ for some formulas $g_1, \dots, g_k$, then we can derive a new formula h by chaining f with $g_1, \dots, g_k$ on S, as follows:

- Let h be f with S replaced by a new variable X not appearing in f.
- For $i = 1 \dots k$, append to h all predicates of g_i, with non-head variables in g_i renamed so as not to conflict with those already appearing in h, and with head variable A in g_i renamed to X.

Note that the resulting h has an extent $\mathtt{E}(h)$ that is a superset of $\mathtt{E}(f)$. Moreover, if $S = \mathtt{E}(g_i)$ for all $1 \leq i \leq k$, then $\mathtt{E}(h) = \mathtt{E}(f)$.

A key concept here is that an object-set $S \subseteq V$, when it occurs as a set-argument in a pseudoformula, represents the set of pseudoformulas satisfied by all members of S. Pseudoformulas (and formulas) are thus partitioned into classes according to their extents. In computation, we deal with the set of object-sets (seen as both extents and set-arguments), with each object-set representing a class of formulas. There are 2^n such sets, which although large is at least finite.

Through chaining and conjoining, the base set of pseudopredicates $\mathbb{P}$ can be used to construct more complex pseudoformulas and formulas. For a formula f, let $\mathtt{G}(f)$ be the undirected, unlabeled graph *corresponding* to f:

[1] Technically, we do not allow formulas within formulas, so g's predicates must be substituted explicitly into h with appropriate variable renaming.

- The nodes of G are the variables and constants appearing in f, except that all instances of a constant in f are treated as different nodes.
- For every predicate $\mathtt{p}(\alpha, \beta)$ in f there is a corresponding edge between α and β in G, so that $\mathtt{p}_1(A, X)$ and $\mathtt{p}_2(A, X)$ would yield two edges between A and X.

Then we can state the following theorem about the formulas that can be constructed.

Theorem. *If $\mathtt{G}(f)$ is a tree, then f can be constructed by chaining and conjoining pseudoformulas, starting from $\mathbb{P}$.*

The theorem says that we can construct all formulas corresponding to tree structures. In general, formulas that are not tree-structured cannot be constructed; however, it is important to note that this does not mean the input graph must be tree-structured. A stronger version of this theorem is stated and proven in Section 3.4.

3.4 Computing a Working Subset of $\mathbb{P}$

The set $\mathbb{P}$ has size $O(2^n|L|)$, which, although much smaller than that of the set of all formulas (which is infinite), is still enormous. In practice, we have to restrict ourselves to using only a subset of $\mathbb{P}$, at the cost of restricting the set of formulas that can be constructed. Ideally, we would use only the most important pseudopredicates as building blocks, from which the most important formulas can be constructed. But we cannot tell which pseudopredicates are important in advance, so we start with an initial set of pseudopredicates $\mathbb{P}_1$ as seed, then iteratively expand (or refine) the set of pseudopredicates included.

We maintain a series of head pseudopredicates and their extents $(P_i, \mathtt{E}(P_i))$, for $i \geq 1$. Each P_i is the set of pseudopredicates created on iteration i, and $\mathtt{E}(P_i)$ is the set of their extents. We compute $(P_{i+1}, \mathtt{E}(P_{i+1}))$ from $(P_i, \mathtt{E}(P_i))$ on each iteration $i+1$. Their successive unions:

$$\begin{aligned} \mathbb{P}_i &= \bigcup_{1 \leq k \leq i} P_k \\ \mathtt{E}(\mathbb{P}_i) &= \bigcup_{1 \leq k \leq i} \mathtt{E}(P_k) \end{aligned}$$

are the working (trimmed) sets of basic building blocks.

We take P_1 to be the set of all head pseudopredicates whose set-argument is a singleton set or the set of all objects. On each iteration, we consider the extents of the pseudopredicates already created, as well as the extents' intersections, and create new pseudopredicates having these sets as set-arguments. More precisely, given that we have $(P_i, \mathtt{E}(P_i))$, we perform the following steps on iteration $i+1$ to get $(P_{i+1}, \mathtt{E}(P_{i+1}))$:

- Compute I_i, the intersection-closure of $\mathtt{E}(P_i)$ (i.e., the smallest set such that $\mathtt{E}(P_i) \subseteq I_i$ and $S_1 \cap S_2 \in I_i$ for all $S_1, S_2 \in I_i$).
- Compute P_{i+1} as:

$$\begin{aligned} P_{i+1} = \{&\mathtt{p}(A, S) \,|\, \mathtt{p} \in \mathtt{L}, S \in I_i\} \\ &\cup \{\mathtt{p}(S, A) \,|\, \mathtt{p} \in \mathtt{L}, S \in I_i\} \end{aligned}$$

Each successive iteration considers formulas corresponding to trees one level deeper. This idea is formalized by the following theorem.

Theorem. *A formula f whose corresponding graph $\mathtt{G}(f)$ is a tree of depth at most k from A can be constructed by chaining and conjoining pseudoformulas starting from $\mathbb{P}_k$.*

Note that the sole formula with a $\mathtt{G}(f)$ depth of 0 is the trivial formula $f(A) :\!-\ \mathtt{p}(A, A)$, which we do not consider.

Proof. The proof is by induction on the depth of $\mathtt{G}(f)$. As the base case, the set of formulas with depth 1 is exactly the base set $P_1 = \mathbb{P}_1$. Now assume that the theorem is true for some $k \geq 1$, and suppose that $\mathtt{G}(f)$ is a tree of depth $k+1$ starting from A. Each child of A is the root of a subtree T_i having depth at most k. Consider those subtrees T_i that have depth at least 1 (i.e., not constants and dangling variables). By the inductive hypothesis, the formula g_i corresponding to each subtree T_i can be constructed from $\mathbb{P}_k$, after we rename the root variable of each subtree (and other instances in the subtree) to A (note that A cannot appear in the subtree already since $\mathtt{G}(f)$ is acyclic).

We want to show that there is a pseudopredicate $\mathtt{p}(A, S') \in \mathbb{P}_k$ such that $\mathtt{E}(\mathtt{p}(A, S')) \subseteq \mathtt{E}(g_i)$. First let us assume that g_i has only one head predicate, $\mathtt{p}(A, \alpha)$ (analogous arguments can be made for $\mathtt{p}(\alpha, A)$). There are two cases:

1. α is a constant. In this case, all other predicates of g_i, if any, are irrelevant, and $\mathtt{E}(\mathtt{p}(A, S')) = \mathtt{E}(g_i)$ where $S' = \{\alpha\}$, and $\mathtt{p}(A, S') \in \mathbb{P}_1 \subseteq \mathbb{P}_k$.
2. α is a variable. Then $\mathtt{p}(A, \alpha)$ was added to g_i through chaining, of a predicate $\mathtt{p}(A, S') \in \mathbb{P}_k$ and g_i', for some formulas $g_1', \ldots, g_k'$. Since g_i is the product of chaining $\mathtt{p}(A, S')$ with other formulas, $\mathtt{E}(\mathtt{p}(A, S')) \subseteq \mathtt{E}(g_i)$.

In either case $\mathtt{E}(\mathtt{p}(A, S')) \subseteq \mathtt{E}(g_i)$, and $\mathtt{E}(\mathtt{p}(A, S')) \in \mathtt{E}(\mathbb{P}_k)$. Now if g_i has more than one head predicate, we can apply the same argument as above on each of its head predicates to derive a corresponding $\mathtt{p}(A, S_j')$ for each of its head predicates. Since I_k is the intersection-closure of $\mathtt{E}(\mathbb{P}_k)$, the set $S_i = \bigcap_j \mathtt{p}(A, S_j')$ is in I_k, and $S_i \subseteq \mathtt{E}(g_i)$. Since $S_i \in I_k$, all head predicates with set-argument S_i are in P_{k+1} and thus $\mathbb{P}_{k+1}$.

We are finally ready to construct f from $\mathbb{P}_{i+1}$. Let f' be the formula which has, for each head predicate $\mathtt{p}(A, \alpha)$ in f (and analogously for predicates $\mathtt{p}(\alpha, A)$), a head predicate of the form:

- $\mathtt{p}(A, \alpha)$ if α is a constant or dangling variable

- $\mathtt{p}(A, S_i)$ if α is a variable which is the root of some subtree T_i corresponding to formula g_i

Then f' can be constructed by conjoining predicates in $\mathbb{P}_{i+1}$, and f is the chain of f' and $g_1, \ldots, g_k$ on set-arguments $S_1, \ldots, S_k$. □

It follows immediately from the theorem that in the limit where $k = \infty$, all formulas corresponding to tree structures can be constructed from $\mathbb{P}_\infty$ (and hence $\mathbb{P}$, a superset of $\mathbb{P}_\infty$), from which follows the weaker version of the theorem, as stated in Section 3.3.

The larger the k, the more formulas can be constructed, at the cost of using more computational resources. In the F-Miner system, we found that $k = 3$ accounts for a wide range of interesting formulas while having very manageable resource requirements (Section 5).

There are many ways the set $\mathbb{P}_k$ can be further pruned or tailored for the class of formulas suitable for a specific application. First, instead of taking I to be the intersection-closure of $\mathtt{E}(P_i)$ in the iterative step $i+1$, we can simply take I to be $\mathtt{E}(P_i)$, or take I to be the (first-level) intersection of elements of $\mathtt{E}(P_i)$. Thus conjunctions of pseudoformulas are only formed when their extents are equal. Second, those pseudopredicates in $\mathbb{P}_k$ and those sets in $\mathtt{E}(\mathbb{P}_k)$ deemed unimportant (Section 4) can be pruned away after each iteration. By keeping only a fixed number of the most important pseudopredicates and their extents, we can limit the growth of $(\mathbb{P}_k, \mathtt{E}(\mathbb{P}_k))$, while still allowing important formulas corresponding to deep tree structures to be considered.

4 Computing Importance

In Section 3 we established a set of head pseudopredicates and their extents $(\mathbb{P}, \mathtt{E}(\mathbb{P}))$ as building blocks for formulas. Formulas can be constructed from $\mathbb{P}$ through conjoining and chaining. Because $\mathbb{P}$ is usually extremely large, we showed how to iteratively compute a manageable subset $\mathbb{P}_k$ of $\mathbb{P}$.

The next step is to construct important formulas from $\mathbb{P}_k$. We first analyze $(\mathbb{P}_k, \mathtt{E}(\mathbb{P}_k))$ as to the importance of the formulas that can be constructed. The problem of computing importance on formulas becomes that of computing importance on sets (representing classes of formulas satisfied by these sets) and pseudopredicates. Just as in traditional data mining we look for interesting objects satisfying some predefined property, we now mine the space of properties for interesting properties satisfying some predefined notion.[2] Accordingly, the development of a good measure of importance for properties is fairly ad-hoc, although we try as much as possible to develop upon known principles. The ranking techniques presented in this section are largely based on empirical experimentation. We present these techniques only as a concrete, viable example. In practice, the computation of importance should be specialized to the application.

4.1 Ranking Head Pseudopredicates and Sets

We start with some fundamental notions of importance for head pseudopredicates (simply "predicates" in the rest of this section), and then let the analysis compute importance based on these notions. We borrow a technique from the field of web search. The *PageRank* [20] and *HITS* [16] algorithms have been used to analyze web pages for importance to aid in web search. The idea behind PageRank is that a web page is important if it is pointed-to by important web pages. Similarly, the HITS algorithm identifies good *hub* pages and good *authority* pages recursively: good hubs are those which point to good authorities, and good authorities are those pointed-to by good hubs. Good authorities are regarded as important pages. Common to these two algorithms is their recursive, mutually-reinforcing definition of importance, and the iterative computation method (corresponding to an eigenvector computation).

In the same spirit, we develop an iterative algorithm for ranking the importance of sets and pseudopredicates. And analogous to the definition of hubs and authorities in HITS, we say that:

- A pseudopredicate is important if its set-argument is important.
- A set is important if it satisfies important pseudopredicates.

Thus, the basic notions from which we derive importance are satisfaction of pseudopredicates (for sets), and importance of the set-argument (for pseudopredicates).

To compute importance scores, this intuition must be formalized mathematically. We take importance scores to be in the interval $[0, 1]$, with importance scores for all pseudopredicates summing to 1, and importance scores for all extents of pseudopredicates summing to 1. For $S \in \mathtt{E}(\mathbb{P}_k)$, we define $\mathbb{P}_k(S)$ to be the set of predicates satisfied by S:

$$\mathbb{P}_k(S) = \{p \in \mathbb{P}_k \mid S \subseteq \mathtt{E}(p)\}$$

As a basis, we start with the core equations

$$I(p) = I(\mathtt{arg}(p))$$

for predicates, which says that the importance $I(p)$ of a predicate p is the importance of its set-argument $\mathtt{arg}(p)$, and

$$I(S) = \frac{c}{|\mathtt{E}(\mathbb{P}_k)|} + (1-c) \sum_{p \in \mathbb{P}_k(S)} I(p)$$

[2]Now, we might mine the space of notions-on-properties to identify the important notions, but obviously this just pushes the same problem up a level. Instead, we settle for mining the space of properties using some predefined notions. Note this does not mean that we are back to where we started, since we can now mine for (first-level) properties instead of just atomic objects. Extension to mining higher-level properties (i.e., properties of properties) is a possible direction for future work.

which says that the importance of a set S has two components: (1) a small *inherent importance* $\frac{c}{|\mathtt{E}(\mathbb{P}_k)|}$ (in our experiments we used $c = 0.2$), and (2) the sum of the importances of the predicates p satisfied by S. This "recursive" equation is analogous to that used for PageRank [20].

These two core equations provide a good starting point in capturing the recursive intuition presented, but more specific details of the analysis should be incorporated.

First, instead of summing over the set of all predicates p satisfied by S, we should sum only over those that are not *subsumed* by another predicate satisfied by S. We say that a pseudopredicate $\mathtt{p}(A, S)$ *subsumes* another pseudopredicate $\mathtt{p}(A, S')$ if $S \subseteq S'$, in which case $\mathtt{E}(\mathtt{p}(A, S)) \subseteq \mathtt{E}(\mathtt{p}(A, S'))$. Intuitively, $\mathtt{p}(A, S)$ specifies a property more specific than that specified by $\mathtt{p}(A, S')$. For example, in Figure 4, if we already know that v_1 satisfies $\mathtt{e}(A, \mathtt{ESPN.com})$, it is pointless to also record that v_1 satisfies $\mathtt{e}(A, \{\mathtt{FOXSports.com}, \mathtt{ESPN.com}\})$.

Another aspect that can be improved is when $S \subseteq \mathtt{E}(p)$, but S is only a small fraction of the objects in $\mathtt{E}(p)$. Then $I(p)$'s contribution to $I(S)$ should be weighed lower than to $I(S')$, where $S' = \mathtt{E}(p)$. For example, in Figure 4, we have

$$\mathtt{E}(\mathtt{e}(A, \mathtt{ESPN.com})) = \{v_1, \ldots, v_m\}$$

so $\mathtt{e}(A, \mathtt{ESPN.com})$'s contribution to the importance of a set $S = \{v_1, v_2\}$ should be smaller than to that of $S' = \{v_1, v_2, v_3, v_4\}$. Thus we consider the term

$$\delta(S, p) = \frac{w_1(|S|, |\mathtt{E}(p)|)}{\sum_{\substack{S' \in \mathtt{E}(\mathbb{P}_k) \\ S' \subseteq \mathtt{E}(p)}} w_1(|S'|, |\mathtt{E}(p)|)}$$

which assigns weights according to the relative sizes of S and $\mathtt{E}(p)$, as compared with other sets S' satisfying p. For both data sets in our experiments we used $w_1(x, y) = (\frac{x}{y})^{\frac{1}{3}}$, which we found to work well empirically.

We may also attribute more importance to those sets that satisfy many pseudopredicates independently of the importance of the pseudopredicates. Let

$$\sigma(S) = w_2(|\mathbb{P}_k(S)|)$$

be the number of predicates satisfied by S, weighted by a function $w_2(x)$. In our experiments, we found $w_2(x) = x^{\frac{1}{10}}$ to work well empirically.

The equations we used in our experiments for scoring predicates and sets are:

$$I(p) = I(\mathtt{arg}(p)) \tag{4}$$

$$I(S) = \frac{c}{|E|} + (1 - c)\sigma(S) \sum_{p \in \mathbb{P}_k(S)} \delta(S, p) I(p) \tag{5}$$

As with the HITS equations, equations (4) and (5) can be solved by iterating to a fixed-point. On each iteration, the scores are normalized so that $\sum_p I(p) = 1$ and $\sum_S I(S) = 1$. For equation (5), the set $\mathbb{P}_k(S)$ must be precomputed for each S, which in general is an expensive operation. One way to alleviate the problem is to set $w_1(|S|, |\mathtt{E}(p)|)$ to 0 when it is below a certain threshold t, in which case we need not check whether $S \subseteq \mathtt{E}(p)$ at all. In our experiments, we used $t = 0.01$, which sped up the computation with no noticeable effect on quality of results.

Note that an appropriate choice of equations is in general dependent on the data set and query type. However, we have found the above equations to work well on the two data sets and two query types we tried. Also note that inherent importances can be assigned nonuniformly to bias the results when there are sets we know apriori to be important. This is analogous to biasing web pages nonuniformly in PageRank to enable a personalized web search [9, 14].

4.2 Selective Construction of Pseudopredicates

The importance rankings for the base set of pseudopredicates and their extents $(\mathbb{P}_k, \mathtt{E}(\mathbb{P}_k))$ tell us the importance of the formulas that can be constructed. Using the chaining procedure as described in Section 3.3, it is straightforward to construct formulas from $\mathbb{P}_k$. However, many queries, such as those in Section 2.2, are computed based on the importance scores of the extents, while the actual formulas serve only as an explanation to the user. Thus an exhaustive construction of all constructable formulas is usually not necessary (nor feasible). Instead, we want to construct only the most appropriate formulas, taking into account not only the computed importance of the formulas but such human aspects as the formulas' brevity, comprehensibility, and variety. Here we present the chaining operation from Section 3.3 as a procedure that allows us to take these factors into account.

We define the function $\mathtt{chain}(f)$, which takes a pseudoformula f as argument and returns the result f' of chaining f with some pseudopredicates. The result is a pseudoformula whose graph $\mathtt{G}(f')$ is one level deeper than f:

- Start off with f' set equal to f.
- For each pseudopredicate (not just head pseudopredicates) p in f' having a non-singleton set-argument:
  1. Let S be the set-argument of p, and let $P(S) \subseteq \mathbb{P}_k(S)$ be a set of pseudopredicates satisfied by S.
  2. Replace S in p by a new variable X not appearing anywhere in f'.
  3. For each $p' \in P(S)$, append p' to f' with head variable A of p' replaced by X.
- Return f'.

The function $P(S)$ can be adjusted, based on the computed importance of the pseudopredicates (and human factors, etc.), to suit the specific query types and end application. As a general rule, it should consist of the m most important pseudopredicates satisfied by S. Most of the variability is in choosing m properly so as to produce informative formulas while minimizing complexity for the sake of user intelligibility. Specific rules for choosing m in the F-Miner system are discussed in Section 5.

Each call to `chain` results in a more complex formula. In theory, we could `chain` some formulas indefinitely, since the same set may be chained over and over again in a cycle. In practice, users will want formulas to be simple, so a maximum-depth or cycle-detection stopping criterion will be used anyway.

As an example, consider the pseudoformula

$$f(A) \text{ :- } \mathtt{Home}(\{\mathtt{Glen},\ \mathtt{Beverly}\}, A)$$

which represents the property of "being the home of `Glen` or `Beverly`". If `Glen` and `Beverly` both like either `Chinese` or `Indian` food, then the formula may be expanded (through one call to `chain`) to

$$f'(A) \text{ :- } \mathtt{Home}(B, A) \,\&\, \mathtt{Food}(B, \{\mathtt{Chinese},\ \mathtt{Indian}\})$$

which represents the property of "being the home of someone who likes `Chinese` or `Indian` food". Finally, this formula may be expanded to

$$\begin{aligned} f''(A) \text{ :- } & \mathtt{Home}(B, A) \,\&\, \mathtt{Food}(B, C) \\ & \&\, \mathtt{Food}(\mathtt{Jennifer}, C) \end{aligned}$$

if `Jennifer` likes `Chinese` and `Indian` food.

5 The F-Miner Experimental System

Based on the framework and algorithms presented in the previous sections, we have implemented an experimental system, *F-Miner*, that supports some of the data mining queries discussed in Section 2.2. Specifically, F-Miner supports the following two query types on arbitrary input graphs:

- **Similarity.** Given a set of input objects, return a ranked list of objects similar to members of that set in the same ways the objects in the set are similar to one another. For example, given two professors in the research group data described below, F-Miner returns a third professor.

- **Explanation.** Given a set of input objects, return formulas "explaining" what the objects have in common. This function can be used to explain the answers returned in the similarity queries.

For efficiency, the user may assign types to each object to minimize redundant comparisons by the system. For example, a university would never be considered as an argument to an `Advisor` predicate between two people. Types help to speed up the implementation without having any effect on semantics.

The exact parameters used in F-Miner are given in Section 5.3.

```
Formulas satisfied:
  4.72: f(A) :- Position(A, MS Student)
  2.08: f(A) :- Advisor(A, Jennifer)
Most similar objects:
  [input]: 79.5
  Jing: 20.2
  Glen: 13.8
  Chris: 13.5
  Beverly: 13.4
  Brian: 10.1
```

Figure 5: Results for query "`Steve`".

5.1 First Data Set: Database Group Survey

We ran F-Miner on two data sets. The first is based on a survey of Stanford University's Database Group, along with publication data from the Database Group's publication server [1]. The data is modeled as a graph where nodes represent all entities that participate in relationships, such as people, food types, and publications. The edges represent relationships, including those that denote food preferences, advisors, undergraduate institution, home country, research interests, and authorship for publications. The graph consists of 1725 nodes and 3552 edges.

When the system is first run, the set of basic building blocks $(\mathbb{P}_k, \mathtt{E}(\mathbb{P}_k))$ is precomputed as described in Section 3.4. A prompt is then presented to the user where a list of objects can be entered as a query for both similarity and explanation. The precomputation takes less than a minute, and each query returns in milliseconds.

We begin with a simple single-object query for "`Steve`". The results of the query are shown in Figure 5. Scores in the query results for this data set have been scaled by 10^6 for legibility. The top portion of the output shows the most important formulas (as determined by the system) satisfied by the input. We find that `Steve` is a Masters student, and that his advisor is `Jennifer`. We chose to list these as separate formulas, although they can be printed as a single conjunctive formula instead. Among the properties satisfied by `Steve`, including the foods he likes and where he went as an undergrad, these two are found to be the most important by the algorithm. The importance scores are listed next to the formulas. The bottom portion of the output lists the 5 objects most similar to `Steve`, along with their similarity scores. The similarity score $s(x)$ for an object x is the weighted sum of the importance scores of the extents satisfied by both x and `Steve`:

$$s(x) = \sum_{\substack{E \in \mathtt{E}(\mathbb{P}_k) \\ \mathtt{Steve}, x \in E}} w_1(2, |E|) I(E)$$

where w_1 is the same weighting function used in Section

```
Formulas satisfied:
  11.6: f(A) :- Undergrad(A, Stanford)
  7.72: f(A) :- Home(A, California)
Most similar objects:
  [input]: 107
  John: 72.4
  Brian: 66.2
  Jing: 65.9
  Glen: 60.5
  Wang: 59.9
```

Figure 6: Results for "`Steve`, `Beverly`".

```
Formulas satisfied:
  3.89: f(A) :- Undergrad(A, B)
     & Undergrad(C, B) & Advisor(C, Jeff)
  1.67: f(A) :- Undergrad(A,B)
     & Undergrad(C, B) & Advisor(C, Hector)
Most similar objects:
  [input]: 34.0
  Chris: 32.0
  Taher: 29.1
  John: 26.0
  Wang: 26.0
  Calvin: 25.4
```

Figure 7: Results for "`Glen`, `Qi`".

4.1. The self-similarity $s(\texttt{Steve})$ is given as a reference (listed as `[input]`) for comparison. In effect, the program finds those people who have "a lot" in common with `Steve`, taking into account the numerous properties they may share. The top match is `Jing`, whom we know to be the only other Masters student in the group. The following are two of `Jennifer`'s other students. The next match, `Beverly`, is neither a Masters student nor `Jennifer`'s student.

To find out why `Beverly` is listed, we can type in "`Steve, Beverly`" as a new query. The results are shown in Figure 6. We find that `Beverly` and `Steve` both went to Stanford as undergrads and are from California originally. Note that these attributes were not regarded by the program to be `Steve`'s most important attributes, but they are the most important of those attributes he shares with `Beverly`. Appropriately, the top matches returned are other students who went to Stanford as undergrads, followed by other students from California.

The next example illustrates more complex formulas for the query "`Glen`, `Qi`". The results are in Figure 7. Comparing the absolute magnitudes of the formula scores with those for the query "`Steve`, `Beverly`", we see that there is relatively little in common between `Glen` and `Qi`. The first formula says that the two people both went to schools that `Jeff`'s students tend to go as undergrads. The second formula is analogous. The two students do not share advisors or other preferences, and these formulas are the best connection between them.

Of course, we can also query on objects other than people. The query results for "`UC Berkeley`, `Stanford`" are shown in Figure 8. The formula identifies these as schools that tend to be attended by people from California. This is indeed the most intuitive result that can be inferred from the data.[3]

```
Formulas satisifed:
  148: f(A) :- Undergrad(B, A)
    & Home(B, California)
Most similar objects:
  [input]: 543
  Cal Poly: 223
  IIT Madras: 217
  MIT: 153
  IIT Bombay: 137
  University of Colorado: 137
```

Figure 8: Results for "`UC Berkeley`, `Stanford`".

```
Formulas satisfied:
  0.182: f(A) :- Gender(A, Male)
  0.169: f(A) :- Knows(A, B) & Knows(B, user-178)
      & Knows(user-178, B)
  0.152: f(A) :- Knows(B, A) & Knows(B, user-898)
Most similar objects:
  user-2244: 21.8
  user-500: 21.8
  user-297: 21.7
  user-1081: 21.7
  user-2353: 21.7
```

Figure 9: Results for "`user-8`, `user-9`, `user-10`".

5.2 Second Data Set: Club Nexus

To test F-Miner on a larger data set, we used data from *Club Nexus* [2], which contains various personal information about 2469 Stanford students. Attributes used include the student's academic standing, major, and a list of Club Nexus members he knows. The data is modeled in F-Miner analogously to the Database Group survey data. The resulting graph has 2852 nodes and 74197 edges. The precomputation step, which needs to be done only once, takes about 3 hours, and each query at the prompt takes about 2 seconds. (Note that our system has not yet been optimized or tuned for scalability.) Sample results for the random queries "`user-8, user-9, user-10`" and "`user-7, user-98, user-178`" are shown in Figure 9 and Figure 10, respectively. Scores in the query results for this data set have been scaled by 10^{10} for legibility.

The results in Figure 9 say that the input students are related because they are all males, they all know someone who knows and is known by `user-178`, and they all know someone who knows `user-898`. The results in Figure 10 say that the students are related because they are all undergraduates and are known by a person who majors in international relations. These kinds of connections are found by

[3] Note that the `Undergrad`, `Home`, and `Advisor` relationships tend to be favored over, say, `Food` because each person has a unique choice for these attributes, whereas he usually has multiple food preferences. This is an effect of the δ function (Section 4.1), which causes a preference for a particular food to be deemphasized when the person has other food preferences.

```
Formulas satisifed:
  0.102: f(A) :- Academic(A, Undergrad)
  0.101: f(A) :- Knows(B, A)
     & Major(B, International Relations)
Most similar objects:
  [input]: 6.45
  user-1503: 6.41
  user-188: 6.41
  user-1854: 6.41
  user-304: 6.40
  user-1735: 6.40
```

Figure 10: Results for "`user-7, user-98, user-178`".

the system for most random groups of people.

5.3 Implementation Details

Our experiments were run on a 2.4GHz Pentium with 1GB of RAM using Java SDK 1.4.1. The code is written entirely in Java, unoptimized and without native methods. The core of the F-Miner system is implemented based on the techniques presented in the previous sections. The same parameter settings were used for both data sets. We used $k = 3$ when deriving the basic building blocks $(\mathbb{P}_k, \mathtt{E}(\mathbb{P}_k))$, and ranked pseudopredicates using 10 steps of the fixed-point iteration process. In computing $\mathtt{E}(\mathbb{P}_{k+1})$, we used $I = \mathtt{E}(\mathbb{P}_k)$, omitting the intersection step for speed, and found this to have little effect on the results (conjunctions were already accounted for).

A proper setting of m for $P(S)$, as discussed in Section 4.2, is largely a user-interface issue. We have developed a heuristic to determine m. Let p_i $(i = 1, 2, \dots)$ be the i-th ranked predicate in order of decreasing importance. We take m to be the minimum of 10, the smallest i such that $\sum_{1\le j<i} I(p_j) \ge 10 * p_i$ (i.e., when extra pseudopredicates are trivial compared to those already included), and the smallest i such that $\sum_{1\le j\le i} I(p_j) \ge 0.9 * \sum_{1\le j} I(p_j)$ (i.e., when at least 90% of pseudopredicates have been accounted for). We have found this heuristic to work well in most cases, providing the results illustrated in the previous figures.

6 Related Work

Our framework most resembles that of *inductive databases* [11], which are based on the *inductive logic programming* (*ILP*) framework [19]. In inductive databases, rules (e.g., association rules) about database objects are treated as first-class objects of the database, so that queries (e.g., in SQL) may be posed on rules as well as objects. For example, in the *MolFea* [10] system for molecular databases, one can query for "all structures (represented by formulas) occurring as substructures in more than 30 molecular structures". While our framework also supports such queries (Section 2.2), it further develops the treatment of formulas as first-class objects by considering the interrelationships between formulas and objects and among formulas themselves. This development is manifested in two key features of F-Miner: the relationships between objects and formulas are used to support similarity queries, and the relationships among formulas are analyzed in the recursive computation of importance.

Along similar lines, the traditional association rules of market basket analysis are generalized in the *WARMR* system [6, 7] to association rules on *Prolog* formulas (similar to our Datalog formulas) evaluated on a relational database. The goal is to find association rules of the form $f \implies g$ where f and g are formulas. As discussed in Section 2.2, this extension of association rule mining can be formulated as a query type in our framework.

The traditional data mining problem of finding frequent itemsets in market basket data [3] has also been extended to graph structures [12, 17, 18, 24, 25]. The focus in graphs is on finding frequent substructures, the graph equivalent of frequent itemsets. Again, such queries are but one instance of the query types supported in our framework, as discussed in Section 2.2.

Other instances of property mining have been studied in specific contexts. One is the problem of identifying "patterns and relations" in the unstructured text of web pages, e.g., [5, 22]. *Patterns* are essentially regular-expressions and correspond to the formulas of this paper; *relations* correspond to extents. The sets of patterns and relations are expanded iteratively starting from a small initial set of known relations. The process can be seen roughly as an extension of the frequent itemsets problem in our framework: frequent itemsets are used to discover additional frequent itemsets.

Other graph mining algorithms to compute similarity of nodes based on graph structure include *co-citation* [21] and its generalization *SimRank* [13]. Again, similarity is but one application for our framework, and advantages of the similarity computation enabled by our framework over specific measures of similarity were noted in Section 2.2.

A particular feature of F-Miner is the ability to relate nodes in a graph through relationships beyond just a single edge, as in the query of Figure 7. This feature was also exhibited in the *proximity search* of [8], which finds nodes in a graph that are nearby in terms of graph distance. However, there is no mechanism in [8] for explaining query results, one of the strengths of our approach. A system was presented in [4] that, given keywords matching tuples across different tables in a relational database, returns a tree denoting the schema relating the matching tuples, where the edges of the tree are foreign-key relationships. The tree serves to explain how the tuples are related. However, these tree structures lack the expressive power of our formulas, and there is no ranking of explanations.

As discussed in Section 4.1, the recursive notion of importance of pseudopredicates is analogous to the notion of importance computed by the *PageRank* [20] and *HITS* [16] algorithms for web pages.

The syntax and semantics of the formulas used in our framework are borrowed from the logic-programming language *Datalog* [23].

7 Conclusion

The main contributions of this paper are summarized as follows:

- We presented a framework under which data mining queries can be posed on graph properties. We showed that many common notions in data mining have analogues in the space of formulas that can be formulated as query types in our framework.

- We developed techniques to deal with the enormous size of the query space. Our *basic building blocks* approach partitions properties into classes, bypassing the prohibitive process of analyzing each property individually.

- We defined a general measure of importance for properties by treating properties as first-class objects and applying known techniques. The measure was a vital component of the experimental system.

- We implemented the *F-Miner* experimental system supporting queries under the property mining framework that are not supported by existing systems.

Our experiments to date have been with relatively small data sets. Much work is yet to be done in algorithms, approximations, tuning, and optimizations if we wish to scale to the largest data sets, such as the web. Nonetheless, many modest-sized data sets with acceptable precomputation and query response times pose interesting applications for our framework already, such as the examples in our experiments. With the proliferation of XML and other easy means of expressing and interlinking data, we expect in the near future to see numerous graph-structured datasets amenable to property mining.

The intent of this paper is mainly to provide a foundation for the mining of (graph) properties. The emphasis has been on demonstrating the utility and feasibility of this new kind of data mining. We have only scratched the surface in terms of theory, algorithms, implementation, and applications; many aspects are open for further research.

References

[1] `http://dbpubs.stanford.edu`.

[2] `http://clubnexus.stanford.edu`.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB)*, Santiago, Chile, September 1994.

[4] Gaurav Bhalotia, Arvind Hulgeri, Charuta, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, San Jose, California, USA, February 2002.

[5] Sergey Brin. Extracting patterns and relations from the World Wide Web. In *Proceedings of the WebDB Workshop at the 6th International Conference on Extending Database Technology (EDBT)*, Valencia, Spain, March 1998.

[6] Luc Dehaspe and Hannu TT Toivonen. *Discovery of Relational Association Rules*, pages 189–212. Springer-Verlag Heidelberg, Germany, 2001. `http://citeseer.nj.nec.com/486120.html`.

[7] Bart Goethals and Jan Van den Bussche. Relational association rules: getting WARMeR. In *Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining*, volume 2447 of *Lecture Notes in Computer Science*, pages 125–139, Germany, 2002. Springer-Verlag Heidelberg.

[8] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity search in databases. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB)*, New York City, New York, USA, August 1998.

[9] Taher H. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the Eleventh International World Wide Web Conference (WWW)*, Honolulu, Hawaii, USA, May 2002.

[10] Christoph Helma, Stefan Kramer, and Luc De Raedt. The molecular feature miner MolFea. In *Proceedings of the Beilstein-Institut Workshop*, Bozen, Italy, May 2002.

[11] Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.

[12] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the Principles of Data Mining and Knowledge Discovery, 4th European Conference (PKDD)*, Lyon, France, September 2000.

[13] Glen Jeh and Jennifer Widom. SimRank: A measure of structural-context similarity. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD)*, Edmonton, Alberta, Canada, July 2002.

[14] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the Twelfth International World Wide Web Conference (WWW)*, Budapest, Hungary, May 2003.

[15] M. M. Kessler. Bibliographic coupling between scientific papers. *American Documentation*, 14:10–25, 1963.

[16] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, San Francisco, California, USA, January 1998.

[17] Michihiro Kuramochi and George Karypis. An efficient algorithm for discovering frequent subgraphs. Technical re-

port, Department of Computer Science, University of Minnesota, 2002. http://www.cs.umn.edu/~kuram/papers/fsg-long.pdf.

[18] Surnjani Djoko Lawrence B. Holder, Diane J. Cook. Substructure discovery in the SUBDUE system. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases (KDD)*, Seattle, Washington, USA, July 1994.

[19] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[20] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford University Database Group, 1998. http://citeseer.nj.nec.com/368196.html.

[21] Henry Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24:265–269, 1973.

[22] Neel Sundaresan and Jeonghee Yi. Mining the web for relations. In *Proceedings of the Ninth International World Wide Web Conference (WWW)*, Amsterdam, The Netherlands, May 2000. http://www9.org/w9cdrom/363/363.html.

[23] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. 1-2*. W H Freeman & Co., New York City, New York, USA, 1989.

[24] Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the International Conference on Data Mining (ICDM)*, Maebashi City, Japan, December 2002.

[25] Mohammed J. Zaki. Efficiently mining trees in a forest. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD)*, Edmonton, Alberta, Canada, July 2002.