OASSIS: Query Driven Crowd Mining

Yael Amsterdamer[1], Susan B. Davidson[2], Tova Milo[1], Slava Novgorodov[1], and Amit Somech[1]

[1]Tel Aviv University, Tel Aviv, Israel
[2]University of Pennsylvania, Philadelphia, PA, USA

ABSTRACT

Crowd data sourcing is increasingly used to gather information from the crowd and to obtain recommendations. In this paper, we explore a novel approach that broadens crowd data sourcing by enabling users to pose *general questions*, to *mine the crowd* for potentially relevant data, and to receive *concise, relevant answers* that represent *frequent, significant data patterns*. Our approach is based on (1) a simple generic model that captures both ontological knowledge as well as the individual history or habits of crowd members from which frequent patterns are mined; (2) a query language in which users can declaratively specify their information needs and the data patterns of interest; (3) an efficient query evaluation algorithm, which enables mining semantically concise answers while minimizing the number of questions posed to the crowd; and (4) an implementation of these ideas that mines the crowd through an interactive user interface. Experimental results with both real-life crowd and synthetic data demonstrate the feasibility and effectiveness of the approach.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining

1. INTRODUCTION

Consider the following scenario: Ann is planning a vacation in New York City with her family. In particular, she is interested in finding combinations of popular child-friendly activities and a nearby restaurant to eat at afterwards, and for each such combination, useful related advice (e.g., walk or rent a bike). She immediately thinks of two options: searching the web, or posting a question on some forum to receive input. However, both of these options have drawbacks.

Web search (e.g., using a search engine, or a dedicated website like TripAdvisor) may return valuable information, but if Ann queries for child-friendly activities or for good restaurants she would still need to sift through the results to identify the appropriate combinations: Not all good restaurants, even if child-friendly, are appropriate after a sweaty outdoor activity; a restaurant may be geographically close to some attraction but not easy to access; and so on. Moreover, much of the information is text-based so finding related advice (e.g., walk or bike) may be time consuming. Forums, on the other hand, are more likely to yield detailed answers relevant to Ann's specific question. However, she would again receive a number of (wide-ranging) text-based results, which she would then have to manually examine and aggregate, to extract the desired information out of them.

In this paper, we explore an alternative approach which broadens crowd-based data-sourcing by enabling users to pose *general queries* to the crowd and receive *concise, relevant answers* that represent *frequent, significant patterns*.

The user's interaction with our system is based on two types of data sources: an *ontology* that captures general knowledge, and a *crowd* of data contributors with personal knowledge. Returning to our example, the ontology would include facts such as "Maoz Vegetarian is a restaurant in NYC". These facts are illustrated in the sample ontology in Figure 1 by labeled edges connected to the "Maoz Veg." element. Using the ontology, Ann can formulate questions such as "I seek an activity at a child-friendly attraction inside NYC and a nearby restaurant".

The ontology data, which consists of known and relatively stable information, can often be found in publicly available knowledge bases or mined from the web. However, the ontology *does not* contain information about people's habits, the frequency with which people do certain activities (the frequency of *facts*), or combinations thereof (the co-occurrence of facts within *fact-sets*). For instance, it does not contain information about how often people eat at Maoz Vegetarian when visiting Central Park, or whether they bike rather than walk. For this dynamic and perhaps unknown individual data, the system mines the crowd by asking them questions.

A crowd member's personal history is modeled as a bag of *fact-sets* (see Table 3), each representing an occasion in their past, which they may not completely recall. A personal history can be thought of as a black-box, some details of which can be exposed through the person's answers to questions [5]. For example, if asked the *concrete* question of whether they bike in Central Park, a person may be able to say "Yes, about once a week" even if they cannot remember the exact dates [5]. People may also be able to answer more general yet focused questions, e.g., what types of activities they typically do in Central Park – an open *specialization* ques-

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.
Copyright 2014 ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2610514.

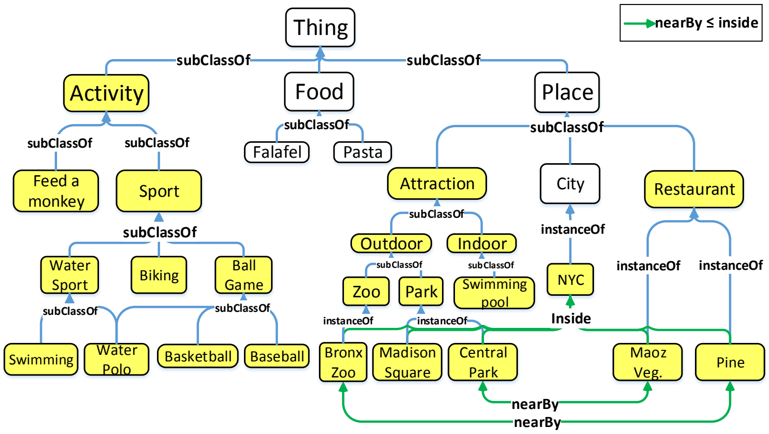


Figure 1: Sample ontology

tion [5]. To find activities that are typically done together (namely, facts which frequently co-occur within fact-sets), questions are bundled together, e.g., "How often do you go to Central Park and also eat at Maoz Vegetarian?" To find additional relevant information the system could add "What else do you do when you go to Central Park and eat at Maoz Vegetarian?", in response to which the person might suggest renting bikes at the Boathouse. Note that this additional information is not explicitly requested by Ann, but is found because it frequently co-occurs with the other two facts.

Semantic connections between terms are known to be useful for speeding up the computation in classic data mining [28] and crowd mining [2]. Thus, in addition to helping phrase the question, the ontology can be used to reduce the number of questions posed to the crowd. For example, if playing ball games in the Bronx Zoo is found to be infrequent then it is not worth asking the crowd if they frequently play basketball in the Bronx Zoo. Furthermore, the ontology can help compute a concise, redundancy-free set of answers. For example, if basketball is frequently played in Central Park, it follows that more general facts (e.g., about ball games in general) are also frequent. Thus only the most specific answers should be included in the query output.

The final answers to Ann's question may include "Go biking in Central Park and eat at Maoz Vegetarian (tip: rent the bikes at the Boathouse).", and "Feed a monkey at the Bronx Zoo and eat at Pine Restaurant". Since we collect new data from the crowd, obtaining the results may not be instantaneous, as when using a forum. However, in contrast to a forum, the answers that we provide aggregate the answers of many users, are concise, relevant and structured, and thus provide an added value to the user.

Contributions. OASSIS extends previous query-driven crowdsourcing platforms, e.g. [19, 21, 25, 31, 34], by enabling users to mine the crowd for significant data patterns. In contrast to the crowd mining frameworks in [2, 3], our query-based approach focuses the mining process on the particular *user information needs.*

The solution that we propose consists of the following components:

1. A simple, generic model capturing both publicly available knowledge (the vocabulary and ontology), as well as the personal history of crowd members from which frequent fact-sets will be mined.
2. A declarative query language OASSIS-QL (Ontology ASSISted crowd mining Query Language) in which the user information needs can be formulated in order to mine of relevant, frequent fact-sets from the crowd.
3. An efficient query evaluation algorithm for computing semantically concise answers to OASSIS-QL queries. The algorithm builds its output incrementally, effectively pruning the search space at each step to minimize the number of questions posed to the crowd.
4. A prototype system OASSIS, which implements the above ideas and evaluates user-specified OASSIS-QL queries with the help of the crowd, through a dedicated crowdsourcing interface.
5. A demonstration of the effectiveness and efficiency of OASSIS through an extensive experimental evaluation, which includes both real crowd and synthetic data.

OASSIS-QL forms the *necessary formal foundations* for evaluating queries such as Ann's. Moreover, it could also be used for mining fact-sets from *standard databases* and thus represents an independent contribution outside of the crowd setting. Our experiments with OASSIS show the feasibility of the approach. Important future extensions include using natural language to formulate queries; mining patterns other that fact-sets, e.g., association rules; interactively extending and cleaning the ontology data with the help of the crowd; controlling the selection of crowd contributors; and retrieving only the top-k query answers.

Outline of paper. We present a formal model of the vocabulary, ontology and crowd in Section 2. The query language in which user questions are formalized, OASSIS-QL, is described in Section 3. Sections 4 and 5 discuss the evaluation of OASSIS-QL queries. The implementation of the OASSIS prototype system, as well as experimental results, are described in Section 6. Related work is in Section 7, and we conclude in Section 8.

2. PRELIMINARIES

We start by presenting a simple, generic model that captures (i) *general knowledge*, including a vocabulary of terms and an ontology of universal facts, and (ii) *individual knowledge*, namely the personal knowledge of each member of the crowd, to be collectively mined. These components will be used in formulating crowd mining queries (Section 3).

DEFINITION 2.1 (VOCABULARY). *A vocabulary $\mathcal{V}$ is a tuple $(\mathcal{E}, \leq_{\mathcal{E}}, \mathcal{R}, \leq_{\mathcal{R}})$, where $\mathcal{E}$ and $\mathcal{R}$ are sets of* element *and* relation *names, respectively, and $\leq_{\mathcal{E}}$ and $\leq_{\mathcal{R}}$ are partial orders over these sets, respectively.*

Elements in $\mathcal{E}$ can be nouns such as Place or NYC, and actions such as Biking. Relations in $\mathcal{R}$ can be terms such as inside, nearBy and parentOf. $\leq_{\mathcal{E}}$ signifies a semantically reversed subsumption relationship between elements, e.g., biking is a sport and hence Sport $\leq_{\mathcal{E}}$ Biking. $\leq_{\mathcal{R}}$ signifies a similar order over relations. From now on, we assume a fixed vocabulary $\mathcal{V}$.

Next, we define the notion of facts, in the spirit of languages such as RDF and OWL [17, 22].

DEFINITION 2.2 (FACTS AND FACT-SETS). *A fact f over $\mathcal{V} = (\mathcal{E}, \leq_{\mathcal{E}}, \mathcal{R}, \leq_{\mathcal{R}})$ is a triple $\langle c_1, r, c_2 \rangle \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$. A fact-set A is a subset of $\mathcal{E} \times \mathcal{R} \times \mathcal{E}$.*

ID	Fact-set		
T_1	Basketball	doAt	Central_Park.
	Falafel	eatAt	Maoz_Veg
T_2	Feed_a_Monkey	doAt	Bronx_Zoo.
	Pasta	eatAt	Pine
T_3	Biking	doAt	Central_Park.
	Rent_Bikes	doAt	Boathouse.
	Falafel	eatAt	Maoz_Veg
T_4	Baseball	doAt	Central_Park.
	Biking	doAt	Central_Park.
	Rent_Bikes	doAt	Boathouse.
	Falafel	eatAt	Maoz_Veg
T_5	Feed_a_Monkey	doAt	Bronx_Zoo.
	Pasta	eatAt	Pine
T_6	Feed_a_Monkey	doAt	Bronx_Zoo

(a) Personal DB D_{u_1}

ID	Fact-set		
T_7	Baseball	doAt	Central_Park.
	Biking	doAt	Central_Park.
	Rent_Bikes	doAt	Boathouse.
	Falafel	eatAt	Maoz_Veg
T_8	Feed_a_Monkey	doAt	Bronx_Zoo.
	Pasta	eatAt	Pine

(b) Personal DB D_{u_2}

Table 3: Example transaction DBs

We denote facts using the notation $\langle c_1, r, c_2 \rangle$ or alternatively the RDF notation $c_1 \quad r \quad c_2$. Now, an *ontology* is modeled as a fact-set with a particular type of data, intuitively capturing "universal truth", that is, facts that hold for all people at all times. An example of such a fact is Central_Park inside NYC.

Example 2.3. *Figure 1 depicts an ontology in graph form. Elements in $\mathcal{E}$ (e.g., Central_Park) are depicted as nodes; and facts (e.g., $\langle$Sport, subClassOf, Activity$\rangle$) are depicted as directed, labeled edges. In this ontology, the relations subClassOf and instanceOf coincide with the reverse of the partial order $\leq_{\mathcal{E}}$, i.e., Activity $\leq_{\mathcal{E}}$ Sport.*

In addition to the "universal truth" represented by the ontology, we are interested in modeling facts (over the vocabulary) that are a part of a particular person's history, e.g., "I played basketball in Central Park (last weekend)", which may be represented as Basketball doAt Central_Park. Given a particular person and occasion (e.g., Ann, last weekend), we call the set of all the facts that hold for them a *transaction*. We also assume that every transaction has a unique ID. Given a set of crowd members $\mathcal{U}$, we associate with each $u \in \mathcal{U}$ a *personal database* D_u, which contains all the transactions of u related to different occasions. $|D_u|$ denotes the number of transactions. Note that D_u is completely *virtual*, and is only used to model a crowd member's history; we assume it is not recorded anywhere, and cannot be directly accessed like a standard database.

Example 2.4. *Table 3 shows the (virtual) personal databases D_{u_1} and D_{u_2} of crowd members u_1 and u_2, respectively. The table columns include the transaction ID and fact-sets, where multiple facts are concatenated by a dot. For instance, T_1 describes an occasion during which u_1 played basketball at Central Park and ate at Maoz Vegetarian. Note that some transaction elements appear in the vocabulary but not in the ontology (e.g., Boathouse).*

Due to the semantic subsumption defined by the vocabulary's order relations, facts may also *implicitly* occur in a database (or in fact-sets, in general). For example, consider a transaction that contains Basketball doAt Central_Park. If Sport $\leq_{\mathcal{E}}$ Basketball, then Sport doAt Central_Park also implicitly occurs in the transaction. We formalize this by *extending* $\leq_{\mathcal{E}}$ and $\leq_{\mathcal{R}}$ to facts and fact-sets.

Definition 2.5 (Facts and fact-sets partial order).

Facts. *Let $f = \langle e_1, r_1, e'_1 \rangle$ and $f' = \langle e_2, r_2, e'_2 \rangle$ be facts. We say that $f \leq f'$ iff $e_1 \leq_{\mathcal{E}} e_2$, $r_1 \leq_{\mathcal{R}} r_2$ and $e'_1 \leq_{\mathcal{E}} e'_2$.*

Fact-sets. *Let A and B be fact-sets. $A \leq B$ iff for every fact $f \in A$ there exists a fact $f' \in B$ such that $f \leq f'$.*

We say that a transaction T *implies* a fact f (resp., fact-set A) if $\{f\} \leq T$ (resp., $A \leq T$), where T is viewed as a fact-set.

Example 2.6. *Returning to our running example, the following also hold: letting $f_1 = \langle$Sport, doAt, Central_Park$\rangle$ and $f_2 = \langle$Biking, doAt, Central_Park$\rangle$, then $f_1 \leq f_2$ since Sport $\leq_{\mathcal{E}}$ Biking; suppose that nearBy $\leq_{\mathcal{R}}$ inside is in our vocabulary, then letting $f_3 = \langle$Central_Park, inside, NYC$\rangle$ and $f_4 = \langle$Central_Park, nearBy, NYC$\rangle$, we have $f_3 \leq f_4$. Based on the previous observations, $\{f_1\} \leq \{f_1, f_3\} \leq \{f_2, f_3\}$; and for T_1 from Table 3, $\{f_1\} \leq T_1$.*

The significance of a fact-set A for a person u is measured by its *support* in transactions in D_u, defined as $\text{supp}_u(A) := |\{T \in D_u \mid A \leq T\}| \,/\, |D_u|$

Example 2.7. *Consider again the personal DBs in Table 3, and the fact-set $f_1 = \{\langle$Pasta, eatAt, Pine$\rangle, \langle$Activity, doAt, Bronx_Zoo$\rangle\}$. For D_{u_1} we have, e.g., $\text{supp}_{u_1}(f_1) = {}^{1}/_{3}$, since the two facts of f_1 are implied by T_2 and T_5.*

Questions to the crowd. Recall that D_u is *virtual*, and so we cannot access it to compute the support for a given fact-set. Instead, we infer this support by asking u a question about its frequency, as in [3]. We define the following two types of questions to the crowd.

Concrete questions: retrieve just the support of a given fact-set from the crowd member.

Specialization questions: given a fact-set ask the crowd member to specify a more specific, significant fact-set along with its support, to speed up information gathering (see Section 4.1).

A concrete question about, e.g., $\{\langle$Biking, doAt, Central_Park$\rangle$, $\langle$Rent_Bikes, doAt, Boathouse$\rangle\}$ may be presented to the user as "How often do you go biking in Central Park and rent bikes at the Boathouse?" The user answer could be "Once a month", which can then be interpreted as, e.g., the support value ${}^{12}/_{365}$ (signifying 12 days a year). An example for a specialization question might be "*what type* of sport do you do in Central Park? How often do you do that?" The answer could be, e.g., "Basketball" for the first part, and "every other Sunday" for the second, translating the frequency to support as in a concrete question. Previous crowd mining work observed that interleaving open-ended and concrete questions is beneficial: specialization questions allow prominent, significant fact-sets to be found quickly, by simply asking people to specify them; whereas, concrete questions are

```
 1 SELECT FACT-SETS
 2 WHERE
 3     { $w subClassOf* Attraction.
 4       $x instanceOf $w.
 5       $x inside       NYC.
 6      $x hasLabel    "child-friendly".
 7       $y subClassOf* Activity .
 8      $z instanceOf  Restaurant.
 9      $z nearBy      $x}
10 SATISFYING
11      { $y+ doAt    $x .
12        [] eatAt   $z.
13        MORE}
14     WITH SUPPORT = 0.4
```

Figure 2: Sample OASSIS-QL Query

helpful to dig deeper into people's memories and discover data they may have not recalled spontaneously [3, 5].

Given the definition for "personal" support, we can define the *overall significant* fact-sets as the ones for which the *average* support over all crowd members exceeds some predefined threshold. Choosing the support threshold is a general problem in data mining [1], but in our setting it has an intuitive interpretation: support is the average frequency of a habit, and the support threshold represents the minimum frequency, e.g., to discover habits that the average user does at least 3 times a year once can use the threshold $^{3}/_{365}$. Moreover, given the significant fact-sets w.r.t. a certain threshold, we can more easily compute the significant fact-sets for a different threshold, by caching and re-using the crowd answers for the new mining task. See details in Section 6.3.

Since we cannot pose all the questions to all the crowd members and crowd answers are not precise, in practice it is necessary to resort to estimations of this average. For simplicity we use below a simple estimation method where each question is posed to a fixed-size sample of the crowd members and the answers are averaged. More generally one could use any black-box (e.g., of [3, 25]) to determine the number of users to be asked and how to aggregate their answers, see Section 4.2.

3. THE OASSIS-QL QUERY LANGUAGE

We now present our query language, OASSIS-QL, which extends the RDF query language SPARQL [26] with features that account for mining frequent fact-sets. The syntax and semantics of OASSIS-QL are illustrated via the example in Figure 2, which represents the scenario presented in the Introduction: "Find popular combinations of an activity in a child-friendly attraction in NYC and a restaurant nearby (plus other relevant advice)". The query answers would include a formal representation of, e.g., "Go biking in Central Park and eat at Maoz Vegetarian (tip: rent the bikes at the Boathouse).", "Play ball games in Central Park and eat at Maoz Vegetarian" and "Feed a monkey at the Bronx Zoo and eat at Pine Restaurant", given as fact-sets in RDF notation.

The advantages of using OASSIS-QL for crowd mining are: 1) the language is rich, and can be used by experienced users to express a wide range of queries (see the discussion about the expressivity of OASSIS-QL at the end of the section); and 2) since it is based on SPARQL, SPARQL programmers can easily learn it, and user-friendly query formulation tools for inexperienced users can be developed by adapting existing tools for SPARQL. These include, e.g., tools that translate natural language to SPARQL . As a first step, we offer a full language guide with many examples [24], and a user-friendly query editor (see Section 6.2).

We next explain how different parts of the query are used to formulate Ann's intuitive question, and the form of the requested answers.

Overview of syntax. The SELECT statement (line 1) specifies the format of the answers, where FACT-SETS requests output in the form of fact-sets. (Alternatively, VARIABLES can be used to request relevant variable assignments.) The WHERE statement (lines 2-9) defines a SPARQL-like selection query on the ontology $\mathcal{O}$, which computes a set of possible assignments for the query variables. Using these assignments, the SATISFYING statement (lines 10-14) defines the data patterns to be mined from the crowd. Patterns with a high enough support are returned in the requested format.

SPARQL-based features. The WHERE statement defines a *meta*–fact-set, where some of the elements or relations are replaced by variables denoted by $ (e.g., $x). [] stands for *anything*, when we do not care about a variable's value, as long as one exists. An *assignment* φ maps the query variables to relations and elements. φ is said to be *valid* if by applying it to all the variables in the WHERE meta–fact-set, we obtain a fact-set $A \leq \mathcal{O}$, i.e., a set of facts which is semantically implied by the ontology.

Another useful feature of SPARQL shown in the example is $w subClassOf* Attraction, which defines a *path* of 0 or more facts with subClassOf relation connecting $w and the Attraction element. In this manner, we can select any (perhaps indirect) subclass of Attraction.

Basic crowd mining features. Valid assignments to the WHERE statement variables are applied to the variables in the meta–fact-set in the SATISFYING statement, to obtain fact-sets whose support should be mined from the crowd. Intuitively, these are the parts of the user question that are not general knowledge but depend on human judgment (e.g., "popular"). The syntax of the SATISFYING statement is composed of a SPARQL part for defining the meta–fact-set, and a WITH SUPPORT part which defines the required support threshold (see Section 2 for a discussion on how to set this threshold). We say that an assignment φ is *significant* if the support of the fact-set it defines exceeds the threshold.

We denote by $A_{\texttt{WHERE}}$ and $A_{\texttt{SAT}}$ the meta–fact-set of the WHERE and SATISFYING statements, respectively. Given an assignment φ, we abuse notation and denote by $\varphi(A)$ the fact-set resulting from applying φ to all the variables in the meta–fact-set A.

EXAMPLE 3.1. *Assuming that the crowd contains only the two users u_1 and u_2, we explain the semantics of the example query (ignore the colored text, to be explained shortly) on the sample ontology and databases (see Figures 1 and 2 and Table 3). There are several valid assignments w.r.t. the WHERE statement of this query, including the assignment $\varphi_{16} : x \mapsto$ Central_Park, $w \mapsto$ Park, $y \mapsto$ Biking, $z \mapsto$ Maoz_Veg., and assignment φ_{20}, which differs from φ_{16} by mapping y to Baseball (the assignment indices will be useful in the se-*

quel). The average support of $\varphi_{16}(A_{\mathtt{SAT}})$ is avg$(^1/_3, ^1/_2) = {}^5/_{12}$*, which exceeds the threshold in line 14, and that of $\varphi_{20}(A_{\mathtt{SAT}})$ is* avg$(^1/_6, ^1/_2) = {}^1/_3$*. Hence, φ_{16} is significant and φ_{20} is not.*

Advanced features

- **MSP.** Consider an assignment φ_{15} that differs from φ_{16} by mapping y to Sport. This assignment is more general (and thus less informative) than φ_{16}: applying φ_{15} on the `SATISFYING` or `SELECT` statements yields fact-sets that are more general than those yielded by applying φ_{16} (according to the partial order on fact-sets). By default, `OASSIS-QL` queries return only the *maximal* (i.e., most specific) *significant patterns* (MSPs), which form a concise output representation. The rest of the patterns can then be inferred by generalization. To obtain all of the significant patterns one can append the keyword `ALL` to the `SELECT` line. See the formal definition of MSPs in Section 4.1.
- **Multiplicities.** A user may wish to know if *multiple* similar facts co-occur in the same occasion. This multiplicity can be specified in the `SATISFYING` clause of an `OASSIS-QL` query by attaching to each variable or fact how many instantiations of it we are interested in (see the "+" after `$y`). Standard notations like +, $*$, ? can be used for "at least one", "any number", "optional". The default multiplicity is "exactly one". The semantics of a query with multiplicities is that sets of values are assigned to variables instead of single values. Multiplicity 0 it is equivalent to deleting all the meta-facts containing the variables.
- **More.** The `MORE` keyword in line 13 is used to identify any fact-set which commonly co-occurs with the other facts, i.e., the "...plus other relevant advice, if there is any" part of the user query. This is a syntactic sugar for `{$u $p $v}`$*$, i.e., any number of unrestricted facts. See [24] for more details and additional advanced `OASSIS-QL` features.

EXAMPLE 3.2. *Our example query allows any multiplicity greater than 0 for `$y`, and any multiplicity of `MORE` facts. Assignments φ_{16}, φ_{20} and φ_{15} (discussed earlier) had multiplicity 1 for `$y` and 0 `MORE` facts. As another example, φ_{16} could be extended to include the `MORE` fact Rent_Bikes doAt Boathouse (multiplicity 1), signifying the combination of biking in Central Park, eating in Maoz Veg., and also renting bikes at the boathouse. Another extension of φ_{16} could map `$y` to {Biking, Ball_Game} (multiplicity 2), signifying the combination of biking in Central Park, playing baseball in Central Park, and eating at Maoz Veg. Both extended assignments are valid w.r.t. the query, but only the former is significant; it yields a fact-set that is implied by transactions T_3, T_4 and T_7 in the example DBs in Table 3, and thus has an average support of $^5/_{12}$.*

Expressivity. As `OASSIS-QL` is based on SPARQL, its capabilities of performing selection over the ontology, are directly derived from those of SPARQL.[1] As we show in the next section, using multiplicities allows to capture standard frequent itemset mining with `OASSIS-QL`. To simplify the presentation, we only show here how to mine fact-sets from the crowd for a given support threshold; some additional features, such as mining association rules, are described in the language guide [24], and possible future extensions are discussed in Sections 1 and 8.

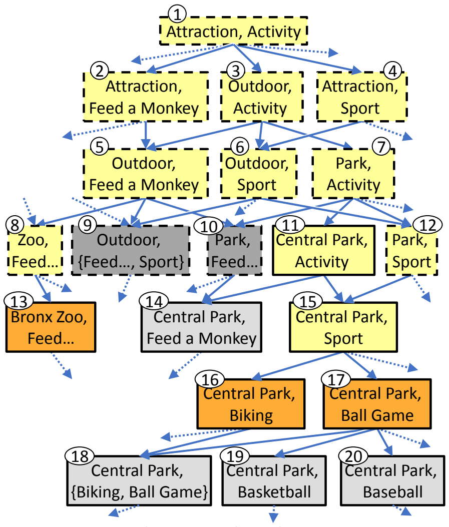


Figure 3: Example partial order over assignments

Query evaluation. Note that there is nothing in the syntax or semantics of `OASSIS-QL` that requires the use of the crowd in the evaluation process, had the user databases been available. Since in practice the user databases are only virtual and cannot be materialized [5] one needs to mine the crowd to obtain the relevant information, and our evaluation techniques are developed accordingly. In the next two sections, we explain how `OASSIS-QL` queries are evaluated by computing valid assignments and identifying which ones are significant. We first assume in Section 4 that we have all the valid assignments, and discuss how to use the crowd to determine the significant assignments. Computing the set of valid assignments in an efficient manner is then discussed in Section 5.

4. MINING THE CROWD

We start by describing query evaluation (focusing on the `SATISFYING` clause) for a single crowd member, then extend the techniques to multiple users.

4.1 Evaluation with a Single User

We describe the *vertical algorithm* that interactively chooses the next question to the crowd. We start by defining a (semantic) partial order over assignments, and then describe how this order is exploited by the algorithm.

Partial order over assignments. We formally define an assignment (with multiplicities) φ as a mapping from the variable space X to $\mathbb{P}(\mathcal{E}) \cup \mathbb{P}(\mathcal{R})$, i.e., to sets of vocabulary elements or relations. The partial order over these assignments is defined as follows.

DEFINITION 4.1 (ASSIGNMENT ORDER RELATION). *Let φ and φ' be assignments. φ' is a* successor *of φ, denoted by $\varphi \leq \varphi'$, if for every variable $x \in X$ and every value $v \in \varphi(x)$, there exists $v' \in \varphi'(x)$ s.t. $v \leq v'$. We use $\lessdot$ to denote* immediate successors, *i.e., $\varphi \lessdot \varphi'$ if $\varphi \leq \varphi'$ and there exists no φ'' (different from φ, φ') s.t. $\varphi \leq \varphi'' \leq \varphi'$.*

When $|\varphi(x)| = 1$, we may identify $\varphi(x)$ with its single element.

EXAMPLE 4.2. *We continue explaining the evaluation of the sample query in Figure 2, but to simplify the presenta-*

[1] For instance, SPARQL does not allow path quantification.

tion, we consider only the parts highlighted by a grey background (i.e., without the nearby restaurant). Figure 3 illustrates parts of the order relation over assignments. Each node represents an assignment φ, and contains two values, $\varphi(x), \varphi(y)$. (The value of w is omitted). There is an edge between φ and φ' if $\varphi \lessdot \varphi'$. Some of the assignments, marked by a dashed outline, are not valid w.r.t. the `WHERE` clause (they are relevant since our algorithm will explore them). Dotted edges denote omitted ancestors or descendants. Node colors are explained in Example 4.5.

Recall assignment φ_{20} from Example 3.1, restricted to the variables x, y (the assignment index denotes the number of the relevant node in Figure 3). Also consider φ_{17} (node 17). It holds that $\varphi_{17} \leq \varphi_{20}$, since the value of x is more specific in φ_{20}. In fact, $\varphi_{17} \lessdot \varphi_{20}$, since Ball_Game $\lessdot$ Baseball.

We can now *formally* define MSP assignments.

Definition 4.3 (MSPs). *Given an `OASSIS-QL` query Q, a valid and significant assignment φ to Q's variables is an MSP if φ is* maximal *w.r.t. $\leq$, i.e., it has no* valid and significant *successor φ'.*

We can infer that an assignment is (in)significant according to its successors and predecessors, using the following observation.

Observation 4.4. *If $\varphi \leq \varphi'$ then if φ' is significant, so must be φ.*

Since all the significant assignments but the MSPs have significant successors, their significance may be inferred. Hence, the MSPs *form a concise representation of the full query result, and it suffices to compute only them.* Such an inference scheme forms the core of many crowd/data mining techniques [10, 20, 2].

Example 4.5. *The color codes in Figure 3 are as follows: MSPs are painted orange; other significant assignments are yellow; insignificant assignments are grey; and the minimal among those (i.e., the most general ones) are painted dark grey.[2] Note that indeed, for every significant assignment, the preceding assignments (its ancestors in the graph) are significant; and this holds symmetrically for insignificant assignments and their descendants.*

The vertical algorithm. The output of the *vertical algorithm* (Algorithm 1) is the set of valid MSPs.[3]

Given a pre-computed set of valid assignments $\mathcal{A}_{\text{valid}}$, the algorithm first *expands* this set in line 1 by adding every assignment that is more general than some assignment in $\mathcal{A}_{\text{valid}}$. This expansion improves the algorithm performance (see below, and Section 6.4, when we compare our algorithm to the naïve approach) as well as the user experience (Section 4.2). Then, it initializes M to be an empty set. As long as there exists an *unclassified* assignment, which is not known to be (in)significant, it chooses the unclassified assignment which is minimal by the partial order (i.e., most general), φ. It then checks whether φ is significant (by asking the crowd, using the function ask$(\cdot)$). If so, it looks for an unclassified assignment φ' s.t. $\varphi \lessdot \varphi'$. For each such φ', if it is significant, the algorithm updates φ to be φ', and in the next iteration of the internal loop, it searches for an even more specific assignment; and so on. The most specific (maximal) significant assignment found in this manner is appended to M. Finally, the set of valid MSPs is returned as the output. Every call to ask$(\cdot)$ can classify multiple assignments (see Observation 4.4).

```
Algorithm
    Input: $\mathcal{A}_{\text{valid}}$: the set of valid assignments
    Output: msp
1   $\mathcal{A} \leftarrow \{\varphi \mid \exists \varphi' \in \mathcal{A}_{\text{valid}}, \varphi \leq \varphi'\}$;
2   $M \leftarrow \emptyset$;
3   while exists unclassified assignment in $\mathcal{A}$ do
4       $\varphi \leftarrow$ the minimal such assignment;
5       if ask($\varphi$) then
6           while exists unclassified $\varphi'$ s.t. $\varphi \lessdot \varphi'$ do
7               if ask($\varphi'$) then $\varphi \leftarrow \varphi'$;
8           add $\varphi$ to $M$;
9   return $M \cap \mathcal{A}_{\text{valid}}$;

Function ask($\varphi$)
1   Ask the user $u$ for $s \leftarrow \text{supp}_u(\varphi(A_{\texttt{SAT}}))$;
2   if $s \geq \Theta$ (the support threshold) then
3       $\forall \varphi' \leq \varphi$ mark $\varphi'$ as significant;
4   else
5       $\forall \varphi' \geq \varphi$ mark $\varphi'$ as insignificant;
6   return ($s \geq \Theta$);
```

Algorithm 1: Vertical algorithm

Example 4.6. *Let us trace one iteration of the outer loop of Algorithm 1 for the user u_{avg}, whose answers are the average support of u_1 and u_2 from the running example. Assume that the input assignments include all the assignments in Figure 3. At the beginning of the algorithm, all the assignments are unclassified, and thus, node 1 represents the minimal (most general) unclassified one. The algorithm can then ask u_{avg} about a sequence of successors of node 1, for example in the following order: 1, 3, 7, 10, 11, 15, 17, and all of 17's successors (including 18-20). This order is obtained by replacing φ with discovered significant successors (e.g., node 3) in the inner loop, and ignoring insignificant successors. For 17 no significant successor is found, and hence it is correctly identified as an MSP. In a second iteration of the outer loop, the selected minimal unclassified assignment could be, e.g., node 2.*

Algorithm analysis. The correctness of the vertical algorithm follows from the monotonicity of significance and the correctness of the inference scheme (proof omitted). We show next that the algorithm is efficient in terms of the number of questions posed to the crowd.

When multiplicities are introduced, the query language is expressive enough to capture standard data mining: e.g., to capture mining for frequent itemsets, use an empty `WHERE` clause and `$x+ [] []` as the `SATISFYING` clause. In this case, the total number of assignments can be exponential in the vocabulary size. Luckily, we can show that even when this is the case, the number of crowd questions can be much smaller in practice.

[2] Nodes 18-20 in Figure 3 are not minimal since they have insignificant ancestors, omitted from the figure.

[3] If the `ALL` keyword is specified, the other significant assignments can be inferred.

We define the *crowd complexity* of an algorithm that evaluates OASSIS-QL queries as the number of unique questions posed to the crowd by the algorithm. The following proposition states the crowd complexity of the vertical algorithm. This can be proved based on the top-down assignment traversal order, as well as the expansion of $\mathcal{A}_{\text{valid}}$.

PROPOSITION 4.7. *The crowd complexity of evaluating an OASSIS-QL query is* $\mathrm{O}((|\mathcal{E}| + |\mathcal{R}|)\,|\mathsf{msp}| + |\mathsf{msp}^{-}|)$.

We now show a lower bound. Let $\mathsf{msp}_{\text{valid}}$ be the set of MSPs among the *valid* assignments.

PROPOSITION 4.8. *The complexity of computing the answer to an OASSIS-QL query using only concrete crowd questions, is* $\Omega(|\mathsf{msp}_{\text{valid}}| + |\mathsf{msp}^{-}_{\text{valid}}|)$.

In practice, $|\mathsf{msp}_{\text{valid}}|$ and even $|\mathsf{msp}|$ are typically of reasonable size, and our optimizations further improve the algorithm performance in practice. See Section 6.

Speeding up with specialization questions. Consider the meta–fact-set `$y doAt $x` and assume that we have already established that $\varphi_{15} : x \mapsto \mathsf{Central\_Park}, y \mapsto \mathsf{Sport}$ is significant. This assignment may have many successors by the partial order, as many as the sport types that we have in our vocabulary. However, some of them may be easily pruned by a human user, who knows, e.g., that people do not ski in Central Park.

By asking specialization questions about φ, e.g., "What type of sport do you do in Central Park?", we are guaranteed to find a following assignment that is at least frequent in the current user's DB, if there exists any. Thus, we are more likely to discover additional significant assignments quickly. This is especially beneficial in incremental evaluation or when the number of query results is limited.

To choose which type of questions to ask, we have used, in previous work, a parameter for the ratio of open-ended crowd questions vs. more concrete ones [3]. In our experiments, instead of forcing the crowd members to answer a specific question type, we allowed them to choose the question type. This was done to study their preferences and to improve their user experience. To study the effect of different ratios of specialization and closed questions, we have varied this ratio in synthetic experiments. See Section 6.

4.2 Evaluation with Multiple Users

We next consider multiple crowd-members working in parallel. Given a set of answers from different crowd members to some question, we assume a *black-box aggregator* that decides (i) whether enough answers have been gathered and (ii) whether the assignment in question is significant or not. Generally, such a black-box could be designed to ensure the *quality* of answers, both for individual answers (e.g., outlier detection [21]) and aggregated answers (e.g., error probability, or an average weighted by trust [3]).

The multiple users algorithm. The assignments per crowd member are traversed in the same top-down order as in the case of a single member, but inferences are done based on the globally collected knowledge. More specifically, Algorithm 1 is changed as follows.

1. The outer loop is executed *per user*, and can be terminated at any point if the user does not wish to answer more questions.
2. Different user answers obtained through the ask($\cdot$) function are recorded per assignment.
3. The **if** condition within the ask($\cdot$) function is changed to "φ is overall significant", which is decided by the black-box aggregator. The aggregator can answer *yes*, *no*, and *undecided*, in which case not enough answers have been collected for φ, and no inference takes place.
4. The return value of ask($\cdot$) is true iff "($s \geq \Theta$) AND φ is not overall insignificant", to prevent the user from being asked about successors of φ if φ is not significant either for the current user or overall.
5. In line 8 of the main algorithm, φ is added to M only if it became an *overall MSP* by the user's answer.

The first fact-set about which a user is asked, is a minimal *unclassified* node, which could be relatively specific among *all* the assignments. To avoid this, the algorithm can be refined to start the traversal from the overall most general assignment (even if it is already classified), and then navigate to a minimal unclassified assignment. This may lead to some redundant questions, but in practice has the advantage of speeding up the computation because when a general assignment is discovered to be insignificant, its (typically many) successors can be pruned for this user. In addition, it allows for a pleasant user experience.

Crowd member selection. We have already noted that the black-box aggregator can be used to monitor answer quality. In addition, previous works propose different methods for evaluating crowd workers' quality, e.g., to filter spammers [18, 27]. These methods can be used here as a preliminary step to filter the crowd members to which we pose questions. In our specific context, two additional methods can be employed to select the crowd members: first, we can check the *consistency* between the answers of the same user, taking advantage of the fact that the support for more specific assignments cannot be larger. In this manner, we can easily filter out spammers, while perhaps still allowing for small inconsistency in a cooperative member's answers. Second, the OASSIS-QL query itself may be extended to specify restrictions on selected crowd members; see the discussion in Section 8.

5. COMPUTING THE ASSIGNMENTS

We complete the picture by explaining how assignments are computed. Since the number of assignments can be large, we adopt a lazy approach for generating them, feeding them to the vertical algorithm when needed. This optimization is important as many assignments may be pruned along the way and their computation may thus be avoided.

Recall that the WHERE clause of OASSIS-QL is specified using SPARQL-like syntax. Without multiplicities, this clause can be efficiently evaluated using a SPARQL query engine. We next explain how to use assignments computed by SPARQL to address multiplicities.

Assignments with multiplicities. Consider two assignments, $\varphi : x \mapsto \mathsf{Central\_Park}, y \mapsto \mathsf{Biking}$ and $\varphi' : x \mapsto \mathsf{Central\_Park}, y \mapsto \mathsf{Baseball}$. These assignments match on every variable (in this case, x) but one (y). If φ and φ' are valid w.r.t. the WHERE clause, then $\varphi'' : x \mapsto \mathsf{Central\_Park}, y \mapsto \{\mathsf{Biking}, \mathsf{Baseball}\}$ must be a valid assignment for multiplicity 2. In the other direction, if φ'' is valid, then by definition φ and φ' are valid, being subsets of φ''.

We say that φ'' is a *combination* of φ and φ' if there exists a variable $x \in X$ s.t. for every $y \neq x$, $\varphi''(y) = \varphi(y) = \varphi'(y)$, $\varphi''(x) = \varphi(x) \cup \varphi'(x)$ and $\varphi(x), \varphi'(x) \subset \varphi''(x)$. I.e., φ and φ' differ only by their assignment to x, and φ'' is equivalent to their "union".

PROPOSITION 5.1. *Every valid assignment with multiplicity $i > 1$ for some variable x is a combination of two valid assignments with multiplicity $j < i$ for x.*

By induction, this means that we can lazily compute assignments of any multiplicity greater than 1 as a combination of multiple assignments with multiplicity 1.

It is left to handle the computation of multiplicity 0: since this requires removing some of the conditions in the WHERE clause, the result may include values that do not appear in assignments with multiplicity 1. Moreover, each combination of multiplicity 0 for a different set of variables may include a different set of values. Here, we have no choice but to compute the union of all the possible combinations of variables with multiplicity 0.

Expanding the assignment set. We have discussed how to compute all the valid assignments. Last, we exemplify the expansion of the assignment set (line 1 of the algorithm). Note that these assignments can also be generated in a lazy manner, as needed by the algorithm.

EXAMPLE 5.2. *Consider lazily computing the assignments of Example 4.6: assume that we keep records of the valid assignments computed so far (at first, these are only the SPARQL results with multiplicity 1, i.e., nodes 11 and 13-17). Node 1 is computed first and generates a question to the crowd. Next (iterating in the inner loop), node 3 is computed by specializing* Attraction *to* Outdoor*. Then we must verify that this node is a predecessor of a valid assignment and unclassified. Next, node 7 is computed, and so on. Node 18, which has multiplicity 2, is computed as successor of 17 by lazily combining assignments 16 and 17.*

6. IMPLEMENTATION

We have implemented the techniques described in the previous sections in OASSIS, a prototype engine for crowd-assisted evaluation of OASSIS-QL. We start this section by describing the system architecture, and the user interface. Then, we describe two sets of experiments that we have conducted: first, *experiments with a real crowd*; and *synthetic experiments*, whose goal was to examine the effect of varying properties of the data on our algorithm's performance.

6.1 System Architecture

The OASSIS prototype system is implemented in Python 2.7 and uses a MySQL 5.6 database. External libraries used include RDFLIB for handling RDF data and NetworkX for constructing the DAG that represents the partial order over assignments.[4] It uses two data repositories: an *ontology* in RDF format, and *CrowdCache*, which stores the computed assignments along with the answers collected from the crowd for each of them. When OASSIS-QL queries are executed by the system, the valid assignments (without multiplicities) are computed using the RDFLIB SPARQL engine. The assignments are then sent to a module called *AssignGenerator*, which is responsible for (lazily) computing additional assignments (as described in Section 5) and the assignment DAG. A different module, *QueueManager*, is responsible for executing our multi-user algorithm by traversing the assignment DAG and generating a queue of questions for every crowd member who is currently active in the system. *QueueManager* occasionally invokes *AssignGenerator* in a request to compute a successor of some assignment (w.r.t. $\leq$), prunes assignments from the queues if they are no longer relevant (to a concrete user or globally), and updates the collected data in *CrowdCache*.

6.2 User Interface

We developed a web UI (in PHP 5.3), which is used both by the user who poses the query, and by the crowd. For the former, the UI includes (i) an OASSIS-QL query editor, with query templates, and auto-completion for language keywords and ontology elements and relations; and (ii) a UI for browsing the obtained output. As mentioned in Section 3, our vision is to adapt existing intuitive UIs for SPARQL queries to our context. Such tools include graphical query construction (e.g., [13]), and the translation of natural language questions to SPARQL (e.g., [6]).

For the crowd, our UI tool also serves as a dedicated crowdsourcing platform. General-purpose crowdsourcing platform such as Amazon Mechanical Turk[5] did not suit our needs, which include dynamically computing the questions per crowd member based on previously collected data. In our UI, the crowd is engaged to contribute information via a social questions-game where they are asked query-relevant questions about their habits and experiences. Users are awarded stars (bronze, silver and gold) as they answer more and more questions, and can use them as virtual money either to pose queries of their own to the system or to view suggestions computed in response to previous queries. A statistics page commends the top-20 users. Questions are retrieved iteratively from the user queue and are then automatically translated into a natural language question using *templates*. These templates are domain-specific, and can be manually created in advance. E.g., the assignment φ_{17} from Example 4.2 can be presented as "How often do you engage in **ball games** in **Central Park**?", the ontology elements in bold being plugged into the template. The user answers the question by clicking an option out of "never", "rarely", "sometimes", "often" and "very often", which we interpret as support values 0, 0.25, 0.5, 0.75 and 1, respectively. To answer specialization questions, the user can choose to "specify" any part of the presented question. When the user starts typing, auto-completion suggestions that match both the entered prefix and the query are presented to the user.

In designing the interface of OASSIS, we have made two optimizations based on feedback from preliminary user experiments. First, we allow users, via a single click, to indicate that some value occurring as part of the question is irrelevant for them, in which case we infer that every assignment that involves this value or more specific values, has support 0. This optimization, called *user-guided pruning*, allows us to prune large parts of the assignments DAG for this user. Second, in specialization questions, if none of the

[4] www.rdflib.net and networkx.github.io

[5] Amazon Mechanical Turk. www.mturk.com.

auto-completion suggestions are relevant for the users, they can choose *none of these*, which assigns support 0 to all the proposed assignments; in this manner, we obtain the answers to many concrete questions at once, without incurring additional user effort. We have also added a *more* button, which opens a text input and allows the user to enter additional advice for the current assignment (corresponding to the MORE part of the query).

6.3 Results for Real Crowd

In this set of experiments, we have used a real crowd and a real-life ontology and vocabulary combining data from Wordnet [23], YAGO [30] and Foursquare [9].

We have recruited our crowd members to the experiments through social networks. The black-box used for a decision mechanism was simple: 5 crowd answers were required for classifying an assignment; if the average support exceeded the threshold, it was considered significant. We have used the answers from the crowd to simulate executing the same query with different support thresholds: note that the crowd answers are independent of the threshold. The user answers during the execution with support threshold 0.2 were cached, and then reused in evaluating the query for higher support thresholds. E.g., if following a crowd answer we obtained average support 0.2 for some assignment, the algorithm would continue to a more specific assignment for support 0.2, but in the simulation of threshold 0.4, it would stop. In the statistics below, we count for each threshold only the answers used by the algorithm out of the cached ones.

We have experimented with queries from 3 application domains. All of them were chosen to reflect situations in which data is not recorded in a centralized, systematic manner, and people are the main source of knowledge. The first domain is the *travel recommendation* scenario, where we have executed our running example query, with slight modifications to make it suitable to the target crowd (e.g., replacing NYC by Tel Aviv) and other variations thereof (e.g., asking only about sports activities and locations). The second domain is *culinary preferences*, where our queries retrieve popular combinations of dishes and drinks of different types (snacks, health food) which can be used, e.g., in composing new restaurant menus or by dieticians. The third domain is *self-treatment*, where our queries find what do crowd members take in order to relieve common illness symptoms, information which can be used, e.g., by health researchers.

In general, the execution of queries from the 3 domains exhibited similar trends, but required a different number of questions to completion, between 340 and 1416 (which we observed to be correlated with the number of MSPs, see Figures 4a-4c and Section 6.4). 248 crowd members in total have answered 20 questions on average per query to which they have contributed. As our user base kept growing between subsequent queries, a speedup was observed in finding the first MSP, which dropped from 28 minutes to less than 4, and in completing the execution, which dropped from 36 hours to less than 10. In comparison, consider common forums such as TripAdvisor Forums[6], as an alternative means for collecting new, targeted data from web users. While such forums have a much larger and devoted user base, we observed that it still typically takes tens of minutes to get the first post, and tens of hours until the last post is published. To this, one should also add the user's processing time for the posts – reading, extracting the relevant components, aggregating the suggestions, identifying consensus, etc. We provide the user with answers that are already aggregated, starting the first MSP. The answers that we provide are also structured, comprehensive and relevant, and we generally provide them faster.

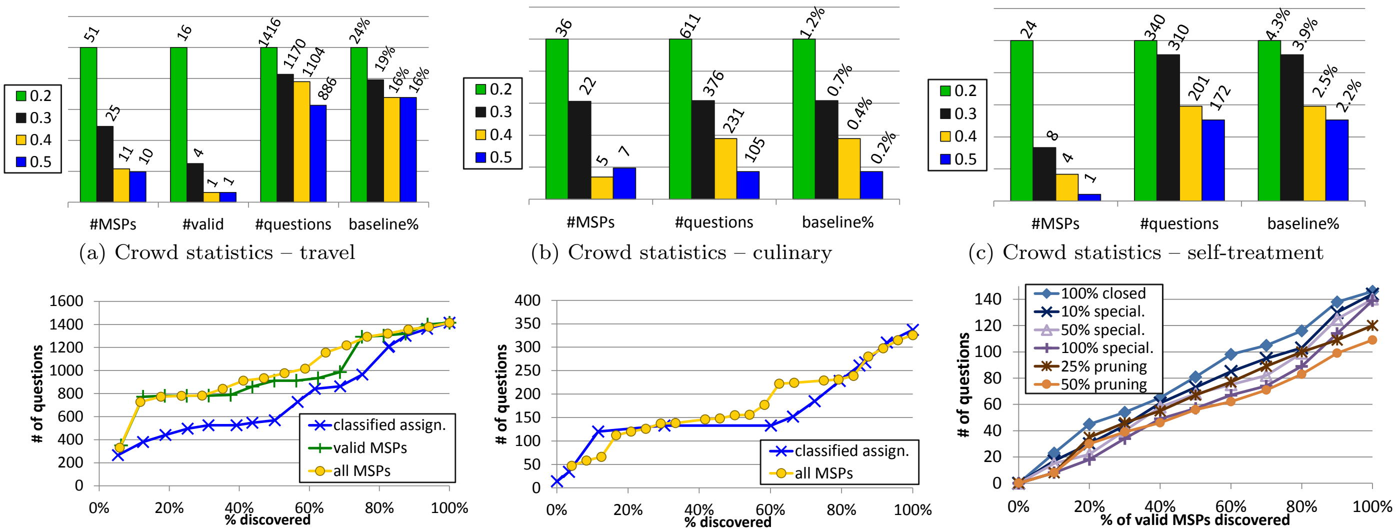
To highlight the key aspects in query execution, we further detail in Figure 4 about three example queries from the three domains, where the query in the travel domain corresponds to the running example. The selected queries exhibit two general situations: first, in the running example query, we are interested in *instances* (of places and restaurants) and hence some of the discovered MSPs may not be valid w.r.t. the query (e.g., an MSP that contains the element "Italian restaurant" rather than a specific restaurant); and second, in the two other queries we are interested in *classes*, and hence all the MSPs are valid (e.g., both "Pizza" and "Italian food" are considered as classes and can appear in a valid MSP). The queries also exhibit 3 extreme cases: the travel query required the most crowd questions to complete, the self-treatment query required the fewest questions, and the culinary query had the largest number of possible assignments in the DAG: in total, the DAGs of the three queries contained 4773, 10512 and 2307 nodes respectively (without multiplicities). Figures 4a-4c show different statistics about the 3 queries, for threshold values ranging from 0.2 to 0.5, all scaled to obtain similar bar heights. **#MSPs** and **#valid** represent, respectively, the *total number of MSPs* and the *number of valid MSPs*[7], which, as expected, generally decrease as the threshold value increases.[8] **#questions** represents the total number of questions asked *including repetitions*: unlike in Section 4, here we wish to measure the exact overall user effort, including asking multiple crowd members about the same assignment according to the multi-user algorithm. Observe that this number of questions decreases as there are fewer MSPs and larger parts of the DAG can be pruned. To illustrate the amount of questions saved by our algorithm, **baseline%** compares the number of questions we ask to a baseline algorithm, which only asks 5 questions for every valid assignment without any specific traversal order. Even when our algorithm considered an expanded set of assignments (as in Figure 4a), it asked at most 24% of the baseline algorithm questions, and this dropped to <5% in queries where such expansion was not needed (Figures 4b-4c). In addition, it provides a better user experience.

Figures 4d and 4e show, for the travel query and self-treatment query respectively, the pace of data collection for threshold 0.2 (the second query behaves similarly to the third and hence its graph is omitted). This is illustrated by the number of questions posed as a function of the % of (i) discovered MSPs (ii) discovered valid MSPs, which is only relevant to the travel domain query, and (iii) classified valid assignments (either as significant or not) at each point of the execution. Observe that generally, towards the end of the execution, classifying each remaining assignment requires more crowd answers: these are typically isolated unclassified parts of the DAG, which cannot be inferred from other

[6] TripAdvisor. http://www.tripadvisor.com/

[7] The number of valid MSPs does not appear in Figures 4b and 4c since all the MSPs there are valid.

[8] As seen in Figure 4b, this is not always the case, as by raising the threshold we may change one MSP to insignificant, turning all its immediate predecessors to MSPs.



(a) Crowd statistics – travel

(b) Crowd statistics – culinary

(c) Crowd statistics – self-treatment

(d) Pace of data collection – travel

(e) Pace of data collection – self-treatment

(f) Synthetic exp.: effect of answer types

Figure 4: Experimental results

assignments. Comparing the two graphs, the self-treatment query required fewer questions in general, but the first decisions on MSPs and classified assignments were done much more quickly. This is since our user base had grown between the executions of the two queries, and thus answers from more users and at a higher rate were obtained during the self-treatment query execution.

Out of the crowd answers, we had 12% for specialization questions, out of which half (6% in total) got a "none of these" answer, 13% user guided pruning and the rest were for concrete questions. This reflects the higher effort incurred to crowd members by specialization questions, since users preferred to answer concrete questions, and additionally shows that the optimizations we introduced (none of these, pruning) were useful to the crowd.

Multiplicities. In each of our queries, we have discovered up to 25 MSPs with multiplicities. In one of the culinary queries we found, among others, that crowd members often have a steak with fries and a coke; or, more surprisingly, that when they eat muesli with yogurt for breakfast they drink apple juice. From the *more* input we have obtained some nice tips from users, e.g., for doing push-ups in a park (an assignment for a travel domain query) it is advisable to lean on a soft surface. We note that in the statistics shown above, we fed to the naïve algorithm only the assignments with multiplicities that our algorithm had generated, for fairness. The effect of our lazy assignment generation is demonstrated by our synthetic experiments in the next section.

6.4 Results for Synthetic Data

The next set of experiments involves synthetic data, simulating the crowd answers for varying data properties. To isolate the effect of varying different properties, we used a simulation of a single user. The results were averaged over 6 trials. The properties studied are as follows.

Shape of the DAG. We have examined the effect of the assignment DAG width and depth on the algorithm performance. For that, we have used a DAG similar to the one generated in our crowd experiments with the travel query, but varied its width between 500 and 2000, and its depth between 4 and 7, by arbitrarily pruning/replicating parts of the DAG. (In comparison, the width of the DAG in the crowd experiments is around 1350 with depth 7.)

Number of (valid) MSPs. We have considered different number of MSPs ranging from 1% to 10% of the nodes. In practice, the % of MSPs is likely to be much lower than 10% and was around 1.2% in our crowd experiments.

Distribution of MSPs in the DAG. We have used three methods of generating MSPs: (1) using a uniform random distribution over assignments (while guaranteeing that the MSPs are not comparable), (2) biased towards selecting MSPs that are nearby in the DAG, i.e., are separated by at most 4 nodes, (3) biased towards selecting MSPs that are far away in the DAG, i.e., are separated by at least 6 nodes. For each such variation, we have generated MSPs in the entire DAG, or only among valid assignments.

Number and size of MSPs with multiplicities We have varied the number of MSPs with multiplicities between 0 and 5% (out of the total nodes), and their size between 1 and 4 (which was the maximal size of MSP with multiplicity in the crowd experiments).

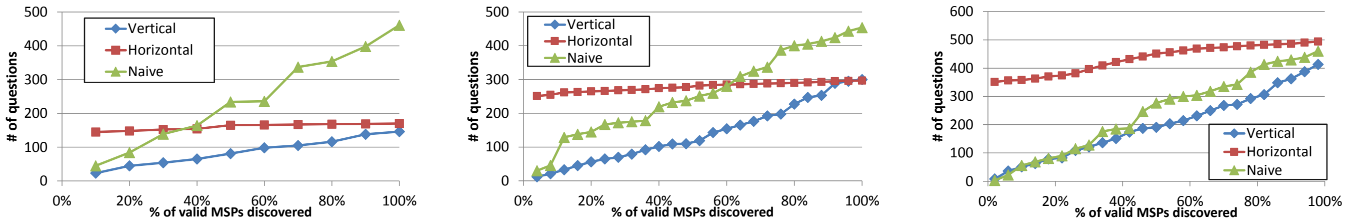
Ratio of specialization vs. concrete questions answered. We set the ratio of answers obtained to such questions in our simulation to 0%, 10% (similarly to the ratio observed in crowd experiments), 50% and 100%. These were simulated by providing the algorithm a significant successor of the current assignment.

Ratio of user-guided pruning clicks We have set the ratio of user-guided pruning clicks obtained to 0%, 25% (similarly to the ratio in crowd experiments) and 50%. A very high ratio of user-guided pruning is not interesting, since it means that there are no significant assignments.

We compare our algorithm to two alternative approaches:

- **Horizontal.** Inspired by the classic Apriori algorithm [1], this algorithm asks about assignment φ only after verifying that all of its predecessors are significant.
- **Naïve.** An algorithm that randomly chooses an assignment among the valid ones.

The alternatives use the same inference scheme as our algorithm and avoid questions on classified assignments.



(a) 2% total MSPs

(b) 5% total MSPs

(c) 10% total MSPs

Figure 5: Varying the percentage of MSPs

We have noted, in our experiments, that varying the shape of the DAG and the distribution of the MSPs in the DAG had no significant effect on the observed trends. Hence, we present here the results only for a DAG of width 500 and depth 7, and a random distribution of selecting MSPs only among valid assignments.

Varying the number of valid MSPs. Figure 5 shows the results for different numbers of chosen valid MSPs. For each such number, and for the different algorithms, the figure shows the number of questions required to discover $X\%$ of the selected valid MSPs. For example, for the case of 10% valid MSPs, the vertical algorithm found 20% of these MSPs (0.2% of the total assignments) after asking 80 questions. With respect to the horizontal algorithm, our top-down vertical algorithm starts returning answers to the query much faster (e.g., it asks fewer than 35% of the questions asked by the horizontal algorithm to discover 20% of the MSPs), which is very useful in practical scenarios since answers can be returned faster, as soon as they are identified. As a higher % of MSPs are found, the gap becomes smaller, since the vertical algorithm saves questions on significant assignments by its traversal order but may "waste" some questions on insignificant assignments in an attempt to find a significant successor. The naïve algorithm performs well only when there is a high % of MSPs and thus discovering them by a "lucky guess" is possible (which, as mentioned above, is not the case in realistic scenarios).

Varying the number and size of multiplicities. Our experiments showed that the number of questions depends on the % of MSPs, and not on whether they include multiplicities. They also show that the lazy approach for generating DAG nodes with multiplicities was proven very efficient: in all the experiment variations, `OASSIS` has generated less than %1 of the nodes, in comparison to an "eager" algorithm that generates *all* the nodes up to the same multiplicity.

Varying the ratio of answer types. Figure 4f shows the number of questions required to discover $X\%$ of the valid MSPs for different ratios of specialization questions and user-guided pruning. In all cases, a high ratio of these special types of questions improved the algorithm performance (although not by much). However, we note that as more and more assignments are pruned, the number of choices in specialization questions decreases, rendering concrete questions preferable in a real crowd setting (as they incur less user effort). Based on this and on user feedback, we propose allowing users to choose the question type, but offer a higher reward for specialization questions. Determining the exact reward is left for future work.

7. RELATED WORK

Data procurement is one of the most important and challenging aspects of crowdsourcing [8]. Some recent work (e.g., [7, 11, 19, 21, 25, 31, 32, 34]) suggests the construction of declarative frameworks which outsource certain tasks to the crowd, including the execution of common query operators such as filter, join and max, the task of populating a database, information extraction [14, 33], etc. However, the present paper is the first to propose a declarative framework for specifying *data patterns to be mined from the crowd.* The work of [16] focuses on evaluating planning queries with the crowd. While it also considers incremental selection of crowd questions, our construction of the assignment order relation renders our problem very different from theirs.

Previous work in crowd mining [3, 2], by some of the present authors, is the most related to ours. In [3], we consider mining association rules from the crowd, however (i) the approach is not based on an ontology; and (ii) it is not query-based, and thus users cannot direct the mining process. [2] studies the theoretical complexity of mining frequent itemsets from the crowd but does not consider a query language or system-related challenges, which are studied in the present work.

Mining frequent fact-sets in our setting corresponds to frequent itemset discovery, which is a fundamental building block in data mining algorithms (see, e.g., [1]). The idea of using item taxonomies in data mining, which correspond to our semantic order relation over terms, was proposed for the first time in [28]. We extend the semantic partial order over terms to further capture facts, fact-sets and variable assignments. We also mention, in this context, work on the discovery of interesting data patterns through oracle calls [20, 10]. In particular, the traversal order of assignments that we consider for the top-down algorithm is inspired by the Dualize and Advance algorithm of [10]. Similar ideas were also employed in the context of frequent itemset mining from the crowd by [2]. However, it requires further enhancements in our setting to support queries, the traversal of assignments to query variables, user sessions, etc.

`OASSIS-QL` combines capabilities from SPARQL for RDF processing [26] with ideas from data mining query languages such as DMQL [12] (see [4] for a survey), and enhances them to obtain a query language for crowd mining. Specifically, we use SPARQL-like syntax for the part of the query that involves the ontology, and constructs inspired by DMQL for specifying the form of the patterns to be mined. The idea of evaluating such queries using the crowd is new.

Finally, we mention work on mining RDF data [15, 29]. [15] considers mining of RDF concept graphs, and uses a semantic relation over facts similar to ours. However, they do not consider query-driven mining, or mining the crowd.

Adapting their techniques to our setting is an intriguing direction for future work.

8. CONCLUSION AND FUTURE WORK

In this paper, we introduced OASSIS, a system which allows users to specify data patterns of interest and mine them from the crowd. The model used in OASSIS captures both ontological knowledge and the individual history of crowd members from which significant patterns can be mined, and its query language OASSIS-QL allows users to formulate their information needs. We further presented an efficient query evaluation algorithm for mining semantically concise answers, which is implemented by OASSIS. Our experimental results, both with a real crowd and synthetic data, indicate the effectiveness of our solution.

We have only presented in this paper how to mine fact-sets, and have greatly simplified the model in order to give a precise analysis of our algorithm's complexity. Additional features can be found in the language guide [24]. Useful future language extensions include returning the top-k answers or diversified answers; selecting the crowd members, which can be done by adding a special SPARQL-like selection on crowd members to OASSIS-QL; and more. We believe that many of the same principles developed here would still apply in the context of such extensions, e.g., lazy assignment computation, and asking crowd members questions "in context".

The OASSIS engine and its UI can also be extended in many interesting ways, some of which were already mentioned throughout the paper: formulating the queries in natural language can be done by parsing a natural language sentence, constructing the where part of SPARQL as in [6], and identifying the parts that need to be mined from the crowd (subjective/personal content). Additional interesting possible extensions include employing an interactive refinement process for the OASSIS-QL query based on the collected answers; dynamically extending the ontology based on crowd answers; and so on.

9. ACKNOWLEDGEMENTS

We are grateful to the anonymous reviewers, for their insightful comments. This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, by the NSF grant III-1302212 and by the Israel Ministry of Science.

10. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.

[2] A. Amarilli, Y. Amsterdamer, and T. Milo. On the complexity of mining itemsets from the crowd using taxonomies. In *ICDT*, 2014. To appear.

[3] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowd mining. In *SIGMOD*, 2013.

[4] J.-F. Boulicaut and C. Masson. Data mining query languages. In *Data Mining and Knowledge Discovery Handbook*. Springer, 2005.

[5] N. Bradburn, L. Rips, and S. Shevell. Answering autobiographical questions: the impact of memory and inference on surveys. *Science*, 236(4798), 1987.

[6] D. Damljanovic, M. Agatonovic, and H. Cunningham. FREyA: An interactive way of querying linked data using natural language. In *ESWC*, 2012.

[7] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, 2013.

[8] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the World-Wide Web. *Commun. ACM*, 54(4), 2011.

[9] Foursquare. https://developer.foursquare.com/.

[10] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma. Discovering all most specific sentences. *TODS*, 28(2), 2003.

[11] S. Guo, A. G. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD*, 2012.

[12] J. Han, Y. Fu, W. Wang, K. Koperski, O. Zaiane, et al. DMQL: A data mining query language for relational databases. In *SIGMOD*, volume 96, 1996.

[13] F. Hogenboom, V. Milea, F. Frasincar, and U. Kaymak. RDF-GL: a SPARQL-based graphical query language for RDF. In *Emergent Web Intelligence: Advanced IR*, pages 87–116. Springer, 2010.

[14] L. Jiang, Y. Wang, J. Hoffart, and G. Weikum. Crowdsourced entity markup. In *CrowdSem*, 2013.

[15] T. Jiang, A.-H. Tan, and K. Wang. Mining generalized associations of semantic relations from textual web content. *IEEE Tran.*, 19(2), 2007.

[16] H. Kaplan, I. Lotosh, T. Milo, and S. Novgorodov. Answering planning queries with the crowd. *PVLDB*, 6(9), 2013.

[17] G. Klyne, J. J. Carroll, and B. McBride. Resource description framework (RDF): Concepts and abstract syntax. *W3C rec.*, 10, 2004.

[18] J. Le, A. Edmonds, V. Hester, and L. Biewald. Ensuring quality in crowdsourced search relevance evaluation: The effects of training question distribution. In *SIGIR*, 2010.

[19] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. CDAS: A crowdsourcing data analytics system. *PVLDB*, 5(10), 2012.

[20] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data mining and knowledge discovery*, 1(3), 1997.

[21] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. In *VLDB*, 2012.

[22] D. L. McGuinness, F. Van Harmelen, et al. OWL web ontology language overview. *W3C rec.*, 10:10, 2004.

[23] G. A. Miller. WordNet: a lexical database for English. *Comm. ACM*, 38(11), 1995.

[24] OASSIS-QL language guide. http://www.cs.tau.ac.il/~yaelamst/a/oassis-man.pdf.

[25] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. CrowdScreen: algorithms for filtering data with humans. In *SIGMOD*, 2012.

[26] E. Prud'Hommeaux, A. Seaborne, et al. SPARQL query language for RDF. *W3C rec.*, 15, 2008.

[27] V. C. Raykar and S. Yu. Ranking annotators for crowdsourced labeling tasks. In *NIPS*, 2011.

[28] R. Srikant and R. Agrawal. Mining generalized association rules. In *VLDB*, 1995.

[29] G. Stumme, A. Hotho, and B. Berendt. Semantic web mining: State of the art and future directions. *J. Web Semantics*, 4(2), 2006.

[30] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. In *WWW*, 2007.

[31] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, 2013.

[32] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, 2012.

[33] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: crowdsourcing entity resolution. *PVLDB*, 5(11), 2012.

[34] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.