Managing General and Individual Knowledge in Crowd Mining Applications

Yael Amsterdamer[1], Susan B. Davidson[2], Anna Kukliansky[1], 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 mining frameworks combine *general knowledge*, which can refer to an ontology or information in a database, with *individual knowledge* obtained from the crowd, which captures habits and preferences. To account for such mixed knowledge, along with user interaction and optimization issues, such frameworks must employ a complex process of reasoning, automatic crowd task generation and result analysis. In this paper, we describe a generic architecture for crowd mining applications. This architecture allows us to examine and compare the components of existing crowdsourcing systems and point out extensions required by crowd mining. It also highlights new research challenges and potential reuse of existing techniques/components. We exemplify this for the `OASSIS` project and for other prominent crowdsourcing frameworks.

1. INTRODUCTION

Crowdsourcing is of great current interest. Within the database community, efforts to integrate the crowd into an information system began by "plopping" a database on top of the crowd, essentially treating them as an external (and very slow, potentially unreliable) hard drive. For example, crowd workers can insert additional tuples or fill in missing attribute values in a given table, or help evaluating operators such as selection, group-by (clustering) and order-by (ranking) [6, 7, 14, 15, 16, 17]. Crowdsourcing is also used in performing specific computational tasks such as entity resolution, data cleaning, and machine learning tasks [4, 8, 11, 19, 20, 21]. Including the crowd as an additional information and processing source in these ways has therefore required extensions to the query processing and optimization components of a traditional database architecture.

More recently, *crowd mining* has been proposed for more complex information finding applications [1]. In such applications, *general knowledge* – such as an ontology, or information in a database – is combined with individual knowledge from the crowd. In contrast to general knowledge, *individual knowledge* captures habits and preferences. To illustrate, we give three examples:

1. Ann, a vacationer, is interested in finding child-friendly activities at an attraction in NYC, and a good restaurant nearby. For each such combination, she is also interested in useful related advice (e.g., what to wear, whether to walk or rent a bike, etc.)
   General knowledge: NYC attractions, restaurants, the proximity between them, names of activities
   Individual knowledge: what people like doing best with children, what restaurants they like and frequently visit, what they take with them when they go, etc.
2. A dietician wishes to study the culinary preferences in a particular population, focusing on food dishes that are rich in fiber.
   General knowledge: food dishes that are rich in fiber.
   Individual knowledge: the culinary preferences of individuals in the population.
3. A researcher wishes to study the social habits in communities where the longevity is in the top percentile.
   General knowledge: statistics about the longevity in different communities.
   Individual knowledge: the social habits of individuals.

A number of existing technologies could potentially be used in each of the examples above, e.g. searching the web, or posting a question on some forum to receive input. However, they do not enable general knowledge to be combined with individual knowledge for specific questions. For the first example, Ann could do a web search or use a dedicated website like TripAdvisor to query for child-friendly activities or for good restaurants. However, 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, filter and aggregate.

Ideally, user questions should be posed in natural language, and compiled into a declarative query language which can then be processed and optimized. However, standard query languages cannot be used as-is for questions that mix indi-

7th Biennial Conference on Innovative Data Systems Research (CIDR'15), January 4-7, 2015, Asilomar, California, USA.

vidual and general information needs, since they must enable processing each type of information in a different manner. Evaluating such queries is also complex: deciding what questions to ask next, and to how many and what type of crowd workers (statistical modeling), which is affected by the type of data in question (general or individual); determining how to aggregate answers to obtain high quality results for the different types of data; sequencing questions to crowd workers to create a good user experience; and deciding when to update general and individual knowledge based on crowd-provided information. Moreover, the system may enable personalization so that crowd members can be selected to either resemble the query issuer or diversify the individual knowledge obtained. Relevant profile data may be either obtained directly from users/crowd workers or inferred from user requests and crowd responses. Crowd mining therefore requires significantly more complicated components than those used in previously proposed crowdsourcing architectures.

In the remainder of this paper, we describe a framework architecture for crowd mining applications (Section 2) and discuss how our crowd mining system, `OASSIS` [1], employs this architecture and what research challenges remain (Section 3). In Section 4, we use the architecture to analyze other prominent crowdsourcing systems, compare them to each other and to `OASSIS` and identify extension opportunities for supporting crowd mining. We conclude in Section 5.

We note that an important enabling factor for crowd-based systems is an adequate HCI, and the study of HCI in this context leads to challenges involving many research areas (e.g., social sciences, game theory). While in [1] we have implemented a user-friendly crowdsourcing platform, we do not study these aspects here but focus instead on the *data layer*. Other important challenges when dealing with personal information about the crowd (such as preferences), involve privacy and data security. These issues, again, are out of the scope of this paper, and we may assume that a security layer is instrumented wherever personal data is recorded.

2. ARCHITECTURE

A diagram of the general architecture of a crowd mining framework is given in Figure 1. The components of this architecture are generic, and we next elaborate on the purpose of each of them. Note, in particular, that many of the components employ a separate, dedicated handling of individual and general data; and that the reuse of obtained results, which is an important optimization in a crowd-based setting, is discussed across several components.

2.1 Data Repositories

Let us start by defining the three main data repositories used by our crowd mining framework. Each of them is partitioned into a few distinguished types of data.

The *Knowledge Base* is the main knowledge repository of the framework, which stores long-term data. It consists of two main parts: first, the *input data*, gathered externally to the framework, which may integrate conventional DB data, ontological knowledge bases, document collections, etc.; second, *inferred data*, which is generated based on (automatic analysis of) crowd input. Within this inferred data, we may distinguish *general* from *individual* data. The former type may be viewed as a completion of data missing in the input, for instance, the address of a company, and must adhere to the input schema. The latter type captures data such as opinions, habits and recommendations, which may apply to a particular crowd worker but not to another. The inference process may have some error probability, and the trust in the crowd input may vary based on, e.g., the trust in the contributing crowd workers and the level of agreement between them. To reflect these uncertainties, inferred data may be annotated with provenance and confidence scores. Individual data may be more volatile then general data, as people may change their habits and opinions; hence, this data may further be annotated with expiry dates and properties of its contributing crowd workers.

The *User Profile* repository contains details per crowd worker, and is useful for the management of workers, rewarding and task assignment. Similarly to the Knowledge Base, the data here can be split into *input* and *inferred* data. The former type captures profile information that is given as input to the system (by the workers or by the crowdsourcing platform); and the latter type captures data inferred based on past tasks. For example, the worker Alice may enter her location and birthdate to the system; the system may further infer, based on her task results, that Alice is an expert in geography, and is frequently engaged in sports.

Last, the *Crowd Results* repository is of a more temporary nature and records the results of crowd tasks relevant for a particular query. It records (1) for every worker, a sequence of "raw" results, and (2) some additional inferences that may be done based on the results (see Section 2.4), which may be more convenient for analysis and output display.

2.2 Query Engine

The user information needs are formulated by a declarative query language (to be exemplified in the next section), that allows to seamlessly query general and individual knowledge. The Query Engine is responsible for computing an efficient query plan and managing its execution. The dual objective is to minimize both the machine execution time (as in standard query processing) and the crowd effort (which is often the bottleneck in terms of time, cost and availability [16]).

Some parts of the query plan can be computed based on data which is already recorded in the knowledge base – both general and individual. For the rest, the help of the crowd is required: e.g., to obtain data that is missing from the knowledge base or to re-fetch data that is stale, mistrusted, or needs to be collected from a particular target crowd, according to preferences specified by the query issuer. The Query Engine maps these information needs into *micro-tasks* for the crowd. A micro-task might be, e.g., the verification or completion of a small piece of data, by posing an appropriate question to crowd workers; and it may involve individual or general data, as determined by the query language. The unique micro-tasks that the Engine computes are then sent, along with relevant user preferences (e.g., the budget for paying the crowd, desired properties of crowd workers, etc.) to the Crowd Task Manager, which is responsible for the distribution of each unique micro-task to multiple crowd workers.

Note that to optimize the query execution, crowd tasks may be generated by the Query Engine in an *incremental* manner: the results of previous tasks may affect the necessity of subsequent tasks, and thus the Query Engine may collect some results from the Crowd Task Manager, before computing the best tasks to issue next.

2.3 Crowd Task Manager

Sending each unique task to multiple crowd workers is use-

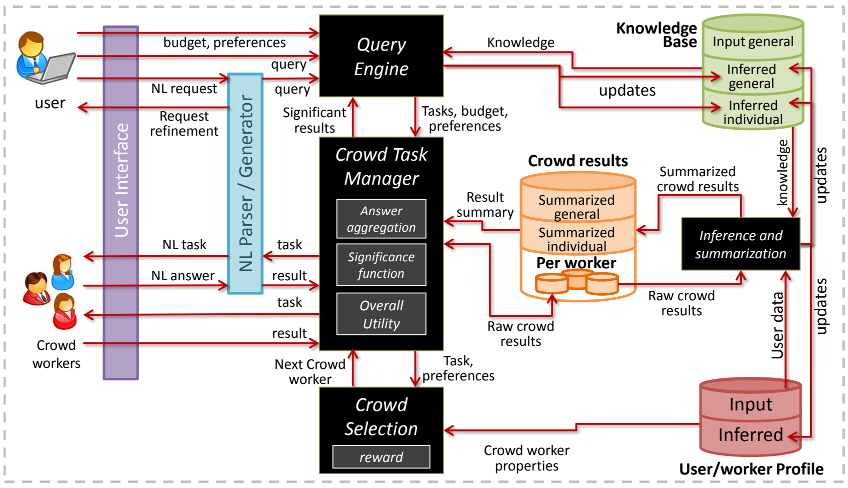

Figure 1: Crowd mining architecture

ful to overcome crowd errors or when the individual views of different crowd workers are needed. Thus, given a task, the Crowd Task Manager has to *aggregate* its results obtained from different crowd workers(by, e.g., computing the average or majority vote). The particular aggregation function may vary depending on the system or user preferences; but more importantly, it should be tied to the type of data – general vs. individual. For the former type of tasks, the aggregation should allow identifying the correct answers with high accuracy, e.g., the correct missing company address. For the latter, there is no single "correct" answer, and the aggregation should capture the (expected and even desired) variety of crowd views/habits. For each such aggregated result, the Crowd Task Manager uses a *significance function* to compute whether sufficient crowd input has been collected, and whether its result is "significant", i.e., belongs in the query output.

The Crowd Task Manager prioritizes submitting the unfinished tasks to workers based on (i) the estimated effect of more task results on the *overall utility*, e.g., the overall confidence in the query output; (ii) the profiles of the available *active workers* and their current session state/context. E.g.,we may prefer to pose to a worker a task related to/different from previous tasks she performed. In other cases, the Task Manager may choose to fetch a new worker from the pool, by issuing a request to the Crowd Selection component (see Section 2.5).

In parallel, the Task Manager may check for inferred, summarized data in the Crowd Results repository, which may render some tasks that were already generated by the Query Engine redundant. For instance, two dependent tasks may be sent to workers in parallel, until enough confidence is gained in the result of one of them. Then, the completion of this task and subsequent inferences may lead to the completion of the other. The significant among the summarized results are fed back to the Query Engine, and form a compact query output.

2.4 Inference and Summarization

This component processes the raw results obtained from the crowd, along with data from the Knowledge Base and User Profile. The inference/summarization process may be based on semantic relations between units of data, on hard or soft constraints, on predefined rules, etc. For instance, given semantic ontological knowledge that Central Park and Battery Park are both parks in New York, individual crowd recommendations about them may be summarized to determine that visiting parks in New York is recommended. We note that, here again, inference about general and individual data may have different flavors.

In a similar manner, the component may send updates, based on inferred and summarized data, back to the Knowledge Base and User Profile repositories. In this manner, subsequent queries may benefit from the execution of previous tasks. For instance, we can enhance worker profiles based on their task results. Given tasks involving general data, we can use their results to estimate the expertise of a worker in a certain domain area, since we can estimate which results were correct and which were erroneous. Given individual data tasks, we can infer the individual habits, likes and dislikes for a worker, based on his or her results.

2.5 Crowd Selection

This component searches the repository for crowd workers that are suitable to perform a certain task. The selection of workers may be done (i) according to explicit preferences given by the query issuer, (ii) by a similarity between the query issuer and the crowd worker (in tasks which involve recommendations), or (iii) by worker properties related to the task, such as the expertise area, quality and availability of the workers. The similarity between the user who issued the query and a crowd worker is a function of their overlapping input profile, as well as similarity in their results in crowd tasks regarding individual data. The latter is especially interesting, as it indicates overlapping habits and opinions between the users.

Based on the three parameters for worker selection mentioned above, the Crowd Selection component may choose the most suitable worker, fail to find a suitable worker, and/or modify the reward based on the worker requirements/availability. For example, given a task involving general data, it may be preferable to pay more for an available, high-expertise worker, rather than for many less suitable workers.

Last, another possible responsibility of this component is to perform a filtering of malicious workers and spammers. It may be possible to detect that the results of a worker are not correlated with the ground truth (in the case of general data) or that they are highly contradictory [1, 13]; such workers will not be selected for further tasks.

2.6 NL Parser/ Generator

Ideally, users should be able to formulate their information request in a natural language (NL), and this would be automatically translated to the target query language. Much previous work has been devoted to translating NL to formal queries on relational/XML/RDF data [3, 5, 12]. However, a new challenge in the context of crowd mining stems from the need to properly identify, in the parsing process, the general vs. individual parts of the user request. (We will illustrate this, for Anne's question from the Introduction, in the following section).

Analogously, for the interaction with crowd workers, the tasks that they receive, initially generated by the Query Engine, should be phrased in NL. Some tasks whose results are free text answers may also require NL parsing. The mechanism here can vary from very simple to very complex. For instance, the generation of tasks may be done using textual instruction templates, along with some rules on how to fit the task parameters into them. Or, if the free text answer of a worker should correspond to a single concept, e.g., an entity, this entity can be searched in a knowledge base.

3. OASSIS AS A FRAMEWORK EXAMPLE

We illustrate next how the architecture described above is used in the OASSIS crowd mining system, and what research challenges remain. We will use Ann's vacation query from the Introduction as our running example. We first provide a brief overview of the system, then explain how its different components fall within the described architecture.

OASSIS allows users to declaratively specify their information needs and mines the crowd for answers. The answers returned are concise, and represent significant data patterns. OASSIS uses an RDF-based model[1] to capture both the general knowledge in an ontology and individual, crowd-provided data. User queries are formulated in OASSIS-QL, a new query language that extends SPARQL[2] with crowd mining features. In the execution of a query, the ontology is used to find candidate data patterns based on general knowledge, e.g., combinations of nearby attractions and restaurants in NYC. The system then mines the crowd by automatically generating and posing questions to the crowd. In this manner, it identifies which patterns are common/significant, as well as related advice. E.g., it can ask which combinations of attractions and restaurants are typically visited together by crowd members. Semantic inference is employed to enrich the crowd-provided data, and semantic summarization yields a compact, comprehensive query output. Dynamic prioritization of crowd questions is performed to support the inference process while minimizing the number of questions asked of the crowd.

A full description OASSIS is given in [1]. We review them below in light of the architecture presented in the previous section.

3.1 The Ontology and Crowd-provided Data

The RDF ontology used in OASSIS forms its *Input General Knowledge Base*, and can be chosen from several publicly available, large knowledge bases, e.g., Yago [9], or Foursquare[3]. It contains elements of different domain areas and *general knowledge facts* in the form element-relation-element (RDF triples). E.g., the fact that "Maoz Vegetarian is nearby Central Park", can be modeled by the triple {Maoz_Veg. nearby Central_Park}. The ontology also contains semantic knowledge, such as a hierarchy of subsuming concepts, e.g., Central Park is a park, a park is an outdoors place, etc.

The data collected from the crowd – namely, pattern frequency and related advice – is recorded as raw data in the Crowd Results repository. It further undergoes an analysis, inference and summarization process (to be explained next), the result of which is also cached in the repository and is eventually used to generate the query output.

```
1  SELECT FACT-SETS
2  WHERE
3      {$x instanceOf Attraction.
4       $x inside       NYC.
5       $x hasLabel     "child-friendly".
6       $y subClassOf Activity.
7       $z instanceOf  Restaurant.
8       $z nearBy      $x}
9  SATISFYING
10      {$y doAt    $x.
11        [] eatAt   $z.
12        MORE}
13     WITH SUPPORT = 0.25
```

Figure 2: Sample OASSIS-QL Query

3.2 OASSIS-QL

We next provide an overview of the query language used in OASSIS. An OASSIS-QL query has three parts: (i) a SELECT clause, which defines the output of the query; (ii) a WHERE clause, which is evaluated against the ontology; and (iii) a SATISFYING clause, which defines the data pattern to be mined from the crowd. An example of a query $\mathcal{Q}$ is given in Figure 2, which expresses the scenario presented earlier: "Find a popular combination of activities in a child-friendly attraction in NYC, and a good restaurant nearby (plus other relevant advice)". The answer to $\mathcal{Q}$, in natural language, would include, e.g., "Go biking in Central Park and eat at Maoz Veg. (tip: rent the bikes at the Boathouse)", and "Feed a monkey at the Bronx Zoo and eat at Pine Restaurant".

We use $\mathcal{Q}$ to illustrate the semantics of OASSIS-QL; full details of the language can be found in [1].

The SELECT clause (line 1) specifies that the output should be *significant fact-sets*, i.e., sets of facts that co-occur frequently, in RDF format. (The language further allows projecting the query results only on certain facts or variables.)

The WHERE clause (lines 2-9) defines a SPARQL-like selection query on the ontology. It consists of facts over the ontology elements, such as NYC or Activity, each in a separate line. The facts also contain variables (e.g. $x, $y, ...), and the selection returns all variable bindings such that the resulting fact-set is contained in the ontology.

The SATISFYING clause (lines 10-14), to which we apply the previously found variable bindings, defines the data patterns (fact-sets) to be mined from the crowd. It further specifies a *support threshold*, which sets the minimal frequency of significant fact-sets (line 14). Consider, for example, the fact-set {[] eatAt $z. $y doAt $x}, where [] stands, intuitively, for "anything". Combined with the bindings $z $\mapsto$ Maoz_Veg., $y $\mapsto$ Sport, and $x $\mapsto$ Central_Park, this fact-set corresponds to a habit of eating (anything) at Maoz Veg. and doing a sport in Central Park. By formulating a question to the crowd, the system can ask them how frequently they engage in this habit, aggregate the answers,

[1]http://www.w3.org/standards/techs/rdf
[2]http://www.w3.org/TR/rdf-sparql-query/
[3]Foursquare API. developer.foursquare.com/.

and decide whether the pattern is significant or not. MORE is a syntactic sugar which captures related, unrestricted advice.

3.3 Query Evaluation

The OASSIS-QL user query is passed to the Query Engine of OASSIS, which is responsible for executing the WHERE clause, finding the relevant variable bindings, sending them to be evaluated by the crowd via the Crowd Task Manager, and returning significant results back to the user. Since the WHERE clause is in a SPARQL-like format, an efficient, off-the-shelf SPARQL module is used to find variable bindings. The bindings are then plugged into the SATISFYING clause to obtain candidate data patterns, each potentially corresponding to a crowd question.

Next, the Query Engine interacts with the Crowd Task Manager, employing an efficient algorithm to evaluate the SATISFYING clause using the crowd. Crowd workers may be asked two types of questions: a *concrete question* about a data pattern, e.g., "How often do you do sports in Central Park?" for the pattern {Sport doAt Central_Park}; or a *specialization* question, where the worker can specify a relevant habit, related to a previous question, e.g., specify which types of sports she does in Central Park. For the former type of question, the Task Manager polls the Engine for the next data pattern about which the worker should be asked, according to the evaluation algorithm.

The algorithm aims to minimize the number of questions asked while providing a pleasant user experience, based, respectively, on theoretical results and on preliminary user studies [1]. It turns out that these considerations dictate the *order* in which questions are asked to crowd members, to be as follows: based on the ontology semantic knowledge, a semantic subsumption partial order is defined over fact-sets; and this partial order is traversed "top-down" in our evaluation algorithm, from the most general to the most specific fact-sets. This traversal can be viewed as a search for the specific patterns that apply to a worker, which is guided by this worker's and other workers' answers. The Task Manager poses each question to multiple crowd workers, and the answers are aggregated using *average*. Statistical modeling is used to determine the result significance and the overall utility of each question. For details see [2].

Finally, the Inference and Summarization component of the OASSIS framework constantly interacts with the Task Manager during the execution of the SATISFYING clause. For instance, having discovered that a pattern is insignificant, (overall or for a particular worker), the Inference and Summarization component infers that more specific patterns (according to the partial order relation) are also insignificant overall/for this user, respectively. Analogously, when a pattern is resolved as significant, more general patterns are inferred to be significant as well. This inference scheme also entails a formal definition of *compactness*: only the *maximally specific* patterns among those which are significant and match the query are maintained as the summarized Crowd Results; the significance of other patterns can always be inferred. Consequently, the most specific patterns are also the patterns that are eventually output to the user.

3.4 Further Challenges

Describing OASSIS in terms of the crowd mining architecture in Figure 1 is useful not only for highlighting what novel components it contains but also, more importantly, for clarifying what challenges still remain for future research.

From the discussion of its data repositories, we observe that the crowd input is currently used only for obtaining individual information. However, as observed in [1], the collection and inference of *general data* from the crowd may also be very useful, e.g., in the case that missing ontology data is encountered while parsing the query or the crowd answers, or to enhance the query in general.

A second desired extension is *crowd selection*. To enable specifying preferences for crowd workers, by the user or the system, OASSIS-QL could be extended by an additional clause specifying crowd properties, with conditions on both input properties (e.g. age group, education), if available, as well as inferred information (e.g. expertise, preferences). The current short-term cache of results that is kept per-worker can easily be converted into long-term profile and used as input for such a query. Finally, the filtering of spammers by OASSIS may be done by identifying workers whose answers frequently contradict the query answer semantics [1].

Last, although OASSIS's UI facilitates query construction using a structured query-building form with auto-completion capabilities, it is an interesting challenge to allow users to specify their queries in *natural language*. Since there already exist tools which translate NL requests into SPARQL (e.g., [5]), and SPARQL forms the basis for OASSIS-QL queries, these tools could be used in a translation from NL to OASSIS-QL. However, any off-the-shelf tool would have to be extended, in order to separate the query parts related to general information from those related to individual information.

4. ADDITIONAL EXAMPLES

Our architecture allowed us to analyze the components of OASSIS and detect its capabilities vs. its challenges. In a similar manner, putting other crowdsourcing systems in terms of the architecture enables a systematic comparison of these systems, and points to extensions required by crowd mining.

Declarative Crowdsourcing Platforms. Different crowdsourcing platforms support the evaluation of queries with the help of crowd workers [7, 14, 16]. They use a standard query language such as SQL, or an enhanced variation of it, which allows, e.g., specifying special operators to be evaluated by the crowd, values to be filled in by the crowd, and/or crowd-related parameters such as the query budget. The queries are evaluated over a standard database augmented by data obtained by generating micro-tasks to the crowd. A micro-task may involve, e.g., filling a missing table value or comparing two values, and is posed to several users for redundancy. The number of users is usually fixed or bounded [7, 16] and their answers are aggregated by a fixed function such as majority vote, or by a predefined function (Resolution Rule [16], e.g., average, duplicate elimination, etc.). The Query Engine then uses standard query plan optimizations over the standard data computations, as well as *crowd-specific optimizations*. Crowd-specific optimizations are usually designed to optimize one of several parameters (the number of micro-tasks, overall cost, overall time or overall accuracy) while keeping the other parameters within a fixed range [7, 16].

Placing these crowdsourcing systems in the context of our crowd mining architecture highlights the key differences between them and OASSIS. Most notably, whereas the distinction between individual and general data is inherent to crowd mining, this distinction is missing in the other systems' data model. Consequently, their query languages do not make this distinction, and the different ways of processing individual versus general data, described throughout Section 3, is not supported. The mix of general, semantic data with individual data in OASSIS, along with its potentially huge space of data patterns, requires a significantly more careful treatment of crowd questions and answers. This is reflected in an intensive interaction between architectural components, e.g., the incremental and sequential selection of questions per user between the Task Manager, Query Engine and Inference modules. Summarization, which is not done in the other crowdsourcing frameworks, is also crucial in OASSIS to support more efficient task selection and inference.

Plug-in Components. Similar to OASSIS, the crowdsourcing systems mentioned above would benefit from some additional "plug-in" components: using a user selection mechanism; more sophisticated methods for aggregating and analyzing task results; and adopting an NL interface for user requests. Some techniques related to these components have already been developed: for crowd selection, e.g., the work of [21] considers incremental learning of crowd worker expertise for labeling data, and the joint choice of data items and annotators. The recent work of [18] considers the dynamic assignment of tasks to workers using a dedicated skill index. The work of [17] estimates worker reward on-the-fly based on task difficulty and result quality. For task result significance analysis, some solutions rely on a dynamic assignment of tasks until the result converges [10] or until enough certainty is obtained, using a model for the crowd errors [15]. Last, there exist NL parsers for different query languages (e.g., [3, 5, 12]), which may be adapted to support the enhanced syntax of the crowdsourcing platforms. Note, however, that these components only target general data and that in the case of crowd mining systems like OASSIS, a plug-in solution must account for individual data as well.

Entity Resolution. We also mention crowdsourced entity resolution, where *structural inference* is used (in contrast with the semantic inference we mentioned in Section 3.3). The transitivity of the equivalence relation between entities can be used to infer clusters of identical entities. In a crowdsourcing framework, this property can be used to minimize the number of crowd comparison tasks, and then inference can be used to complete the rest (e.g., [8]).

5. CONCLUSION

In this paper, we examine a new type of crowd mining application, which allows users to ask queries that mix general and individual knowledge. To account for the complex reasoning and crowd management in such applications, we propose a generic architecture which captures the different components used in crowd mining frameworks. We then place OASSIS and previous crowdsourcing systems within this generic architecture, to examine and compare system components and point out extensions required by crowd mining. We also highlight new research challenges and the potential for reuse of existing techniques/components. In particular, among the remaining challenges we mention crowd selection, the deployment of an NL interface, and the maintenance of crowd-provided data. These challenges become more intricate in the presence of mixed individual and general data, since each data type requires different handling within each component. E.g., inference about worker expertise w.r.t. the ground truth (general data) differs from inference based on reported habits (individual data). We summarize these challenges, forming a roadmap for ongoing and future enhancements of OASSIS and crowd mining systems in general.

Acknowledgements. This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, by the NSF grant IIS-1302212 and by the Israel Ministry of Science.

6. REFERENCES

[1] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. OASSIS: query driven crowd mining. In *SIGMOD*, 2014.

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

[3] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. *NL Eng.*, 1(1), 1995.

[4] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and prescriptive data cleaning. In *SIGMOD*, 2014.

[5] D. Damljanovic, M. Agatonovic, H. Cunningham, and K. Bontcheva. Improving habitability of natural language interfaces for querying ontologies with feedback and clarification dialogues. *J. Web Sem.*, 19, 2013.

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

[7] M. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, 2011.

[8] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.

[9] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artif. Intell.*, 194, 2013.

[10] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *SIGKDD workshop*, 2010.

[11] S. K. Kondreddi, P. Triantafillou, and G. Weikum. Combining information extraction and human computing for crowdsourced knowledge acquisition. In *ICDE*, 2014.

[12] Y. Li, H. Yang, and H. V. Jagadish. NaLIX: A generic natural language search environment for XML data. *ACM Trans. DB Syst.*, 32(4), 2007.

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

[14] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, 2011.

[15] A. G. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal crowd-powered rating and filtering algorithms. *PVLDB*, 7(9), 2014.

[16] A. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: declarative crowdsourcing. In *CIKM*, 2012.

[17] H. Park and J. Widom. CrowdFill: collecting structured data from the crowd. In *SIGMOD*, 2014.

[18] S. B. Roy, I. Lykourentzou, S. Thirumuruganathan, S. Amer-Yahia, and G. Das. Optimization in knowledge-intensive crowdsourcing. *CoRR*, abs/1401.1302, 2014.

[19] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The Data Tamer system. In *CIDR*, 2013.

[20] C. Sun, N. Rampalli, F. Yang, and A. Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *PVLDB*, 7(13), 2014.

[21] Y. Yan, R. Rosales, G. Fung, and J. G. Dy. Active learning from crowds. In *ICML*, 2011.