December: A Declarative Tool for Crowd Member Selection

Yael Amsterdamer[♣], Tova Milo[♢], Amit Somech[♢], and Brit Youngmann[♢]

[♣]Bar Ilan University, Ramat Gan, Israel

[♢]Tel Aviv University, Tel Aviv, Israel

ABSTRACT

Adequate crowd selection is an important factor in the success of crowdsourcing platforms, increasing the quality and relevance of crowd answers and their performance in different tasks. The optimal crowd selection can greatly vary depending on properties of the crowd and of the task. To this end, we present December, a declarative platform with novel capabilities for flexible crowd selection. December supports the personalized selection of crowd members via a dedicated query language Member-QL. This language enables specifying and combining common crowd selection criteria such as properties of a crowd member's profile and history, similarity between profiles in specific aspects and relevance of the member to a given task. This holistic, customizable approach differs from previous work that has mostly focused on dedicated algorithms for crowd selection in specific settings. To allow efficient query execution, we implement novel algorithms in December based on our generic, semantically-aware definitions of crowd member similarity and expertise.

We demonstrate the effectiveness of December and Member-QL by using the VLDB community as crowd members and allowing conference participants to choose from among these members for different purposes and in different contexts.

1. INTRODUCTION

Crowd-based data sourcing is a powerful data procurement paradigm that attracts web users to collectively contribute information. One important challenge, which greatly affects the performance of crowdsourcing applications, is the adequate selection of crowd members to answer questions. In this demonstration we present December (for Declarative Crowd Member Selection), a novel declarative platform for flexible crowd selection. December supports the personalized selection of crowd members via a dedicated declarative query language Member-QL. In particular, Member-QL has refined constructs for capturing the similarity and expertise of crowd members based on their profiles and/or history, through which the relevant crowd for a given context can be effectively identified.

Proceedings of the VLDB Endowment, Vol. 9, No. 10


```
1  ASSIGN TO $u
2  FROM ontology WHERE
3      {$x instanceOf Hotel.
4       $y instanceOf Restaurant.
5       $x inside NEW_YORK,NY.
6       $x near       $y}
7  FROM history($u) WHERE
8      {SELF visit NEW_YORK,NYC} WITH SUPPORT > 0.01
9  SIMILAR profile($u) TO profile(Ann)
10     WITH SIMILARITY >= 0.75
11 SIMILAR history($u) TO
12     {SELF stayAt $x.
13      SELF eatAt $y}
14     WITH SIMILARITY AS querySim >= 0
15 ORDER BY querySim LIMIT 5
```

Figure 1: Example Member-QL query Q_{NY}

Previous work in the context of crowdsourcing and related areas studied various aspects of the crowd selection problem but each focused on a specific facet. This includes, in particular, the selection of members by the relevance of their profiles to a query [4, 7, 11]; the estimation of user expertise [8]; recommendations based on user similarity [3, 5]; and so on. While these works all make a notable contribution, we argue that adequate crowd selection is often a function of many factors combined and thus a more holistic, flexible approach is required to fully capture the properties of the crowd members that should be selected.

As a simple example, assume that Ann seeks some information about hotels and restaurants in New York and decides to refer to the members of some online community she belongs to for obtaining the relevant data. The choice of which members are most adequate for Ann's question depends on Ann's specific information needs: if she seeks recommendations for herself, she may wish to select crowd members with similar profiles to herself, that ranked other hotels and restaurants similarly to herself, but also, importantly, that frequently visit NY; alternatively, if she is running a survey for a particular target audience (e.g., young professionals living in the US), she may seek crowd members whose profiles resemble that of the target and who have answered travel-related questions in the past; and so on. Identifying the relevant crowd members for each such scenario requires combining, in a task specific manner, different factors such as Ann's information needs, the profiles and histories of candidate crowd members, metrics of similarity and expertise, and many others.

To address these challenges, December uses an RDF-based data model and concept hierarchies to capture the rich semantics of the profiles and histories of crowd members. Then, Member-QL queries can be executed over such crowd mem-

ber data in combination with external RDF knowledge bases. As an example, Figure 1 shows a crowd selection query in Member-QL capturing the first scenario described above, where Ann seeks recommendations for herself. The language is SPARQL-based and we explain its syntax and semantics via this example in Section 2.2. Note that the language includes dedicated constructs for specifying user similarity and expertise, for which we provide generic, semantically-aware definitions. The new constructs are evaluated by novel algorithms that we implement in December.

While advanced users can write/edit Member-QL queries directly, we have designed an intuitive *user-guided Query Builder* to assist beginner users. This module allows constructing a crowd selection query based on the information that the user wishes to collect from the crowd, e.g., recommendations for hotels and restaurants in NY. In particular, it enables the user to choose from a menu of standard strategies for crowd selection (e.g., "choose by profile similarity to yourself"), and to refine the query by adding further constraints (e.g. considering only certain parts of the profile or crowd member history) via the UI.

Demonstration. We will demonstrate the capabilities of December by using it as a crowd selection module for the crowd-based data sourcing platform `OASSIS`[2], inviting the VLDB conference members to serve as a real crowd. For the demo, we constructed personal profiles and publications histories for the potential conference participants using online available datasets. The audience will be invited to compose questions for the crowd in various topics, then use December's intuitive Query Builder interface to construct and execute Member-QL queries in order to find adequate crowd members to provide answers. Subsequently, we will review the identified crowd members and use the December UI to discover the segments in their profile and history that led to their selection and ranking, thereby gaining an intuition on the crowd selection process. See more details in Section 4.

Related Work. As mentioned above, crowd selection has received much attention in previous work (e.g., [3, 4, 5, 7, 8, 11, 6], but each focuses on providing a dedicated solution for a specific setting. Our work provides a holistic declarative framework where such efforts can be combined and consolidated. Our query language is inspired by the crowd mining language of the `OASSIS` system [2]. Indeed Member-QL enriches `OASSIS-QL` with dedicated constructs that allow for expressing soft and hard similarity constraints on users' profiles, expertise and history.

2. TECHNICAL BACKGROUND

We next provide a brief overview of the types of knowledge that December uses regarding crowd members, the declarative query language Member-QL and its evaluation.

2.1 Knowledge Repositories

The knowledge in December is represented using an RDF-based model, which consists of *facts* in the form of entity-relation-entity triples, e.g., {`Ann graduatedFrom MIT`}. We use three types of complementary RDF repositories: *ontologies*, *member profiles* and *member histories*. Ontologies are sets of facts unrelated to specific crowd members that often include taxonomical data, e.g., {`Marriott instanceOf Hotel`} and {`Hotel subclassOf Place`}. A profile is then a

`profile(Ann):`		
`SELF`	`livesIn`	`Berlin`
`SELF`	`hasGender`	`Female`
`SELF`	`hasHobby`	`Photography`
`SELF`	`graduatedFrom`	`MIT`

Table 1: Sample profile for crowd member Ann

`history(Ann):` **Fact-set**	**Support**
`SELF visit Boston.` `SELF stayAt Marriott`	0.038
`SELF visit NYC`	0.01
`SELF collaborateWith Bob`	0.1
`SELF collaborateWith Bob.` `Paper hasTopic Crowdsourcing`	0.06

Table 2: Sample history for crowd member Ann

fact-set describing a specific crowd member, using the special entity `SELF` to refer to this member in facts as in Table 1. The profiles can be constructed either from existing profiles in crowdsourcing platforms, or by using profile builder tools based on social networks such as [6]. The history of a member also consists of fact-sets that may further be assigned *support scores* in $[0, 1]$, following the manner in which crowd answers are represented in the `OASSIS` [2]. Similarly to `OASSIS`, these scores represent the level of agreement, significance or frequency of some habit/opinion of the crowd member, which have been provided by the member in response to previous questions. E.g., the first row in Table 2 has a score of $0.038 = {}^{14}/_{365}$, meaning that Ann is visiting Boston and staying at the Marriott for about 14 days a year.

2.2 Query Language

A Member-QL query is composed of two main types of clauses, corresponding, intuitively, to *hard and soft constraints* that may be applied over the different repositories. Soft constraints are used to capture *expertise/relevance* and *similarity*. Consider, for example, query Q_{NY} in Figure 1. This query selects 5 crowd members such that (i) they frequently visit NY (at least 4 days each year); (ii) their profile resembles Ann's profile and (iii) their history is relevant to Ann's question about staying in a NY hotel and eating at a nearby restaurant. We use Q_{NY} to illustrate the query semantics, and omit the full details for lack of space.

The `ASSIGN TO` statement of Q_{NY} (line 1) defines the variable `$u`, which will be assigned names of candidate crowd members and these assignments will serve as the query output. The `FROM ... WHERE` clauses serve to apply SPARQL-like selection over chosen repositories. Since the data is represented as RDF triples (facts), the query is using triples, possibly with variables, to select repository subsets. In lines 2-6 the query seeks, in the ontology, a hotel in NY and a nearby restaurant, which are bound to variables `$x` and `$y`, respectively. In lines 7-8 the query requires that the history of the candidate crowd member `$u` includes visiting NY frequently enough (the frequency is modeled as support and restricted using the `WITH SUPPORT` condition).

The `SIMILAR... TO` clauses serve to compute soft constraints using a similarity score between two repositories (profiles, histories) or selected parts within them. Our definition of similarity is explained in Section 2.3. The first clause, in lines 9-10, selects crowd members whose profiles `profile($u)` have a similarity score ≥ 0.75 with respect to Ann's profile. The second `SIMILAR` clause, in lines 11-14, compares the history of a candidate `history($u)` to a fact-set that represents staying at the formerly selected hotel `$x` and eating at restaurant `$y`. For `$u` to be selected, `$u`'s history must

resemble this fact-set for *some* assignment of `$x` and `$y` with similarity score ≥ 0.75. The similarity score is given an alias `querySim`, which finally serves to select 5 crowd members with the highest such scores (line 15).

2.3 Query Evaluation

Since Member-QL is based on SPARQL, our query engine employs an off-the-shelf SPARQL engine (to execute the standard selections on the RDF repository), along with dedicated novel algorithms for evaluating the new `SIMILAR` clauses. There are two key challenges here:(a)formally defining an adequate similarity measure, and (b) providing efficient algorithms to compute it. Different similarity measures are considered in previous work (e.g., [3, 5, 9]), yet none of them can fully account for the semantically-rich knowledge representation in our setting. We therefore define a new similarity metric that combines two important factors: *semantic similarity* and *support similarity*. For space constraints we omit their formal definition and only give key intuition.

Semantic Similarity. The first factor of our metric, semantic similarity, uses taxonomical information in order to compare two fact-sets beyond their plain text. Consider, for instance, the fact-sets {`SELF graduatedFrom MIT`} and {`SELF graduatedFrom Stanford`}. Given a taxonomy which includes university names and categories, we can determine that these fact-sets convey similar information – e.g., graduating from a private US university

Our formal definition of semantic similarity employs existing taxonomy-based metrics [9] which are lifted to taxonomies of fact-sets using constructions from [1]. Intuitively, the metric finds a fact-set which represents the information common to two input fact-sets, and evaluates the similarity as the *information content* (IC) of this fact-set [9], which is a function inverse to its frequency of occurrence in the database. For instance, assume the common information between two fact-sets is {`SELF graduatedFrom Private_US_University`}. Such information is more specific (and less common) than, e.g., {`SELF graduatedFrom University`} and would thus have a higher IC. This captures our intuition that the first two fact-sets in the previous paragraph are more similar to each other than to the third fact-set.

It is important to note that computing the full taxonomy of fact-sets is intractable [1]. Yet we nevertheless manage to provide an optimized PTIME algorithm for the computation of semantic similarity by computing the information common to two fact-sets *directly* without materializing the full taxonomy. Details are omitted for lack of space.

Support Similarity. The second factor, support similarity, completes the semantic similarity by considering support scores. E.g., two crowd members are similar with respect to {`SELF visit NEW_YORK,NY`} only if they visit NY with similar frequency (i.e., similar support), even if this frequency is low. The support similarity of two users' histories is then defined as the average of support differences for fact-sets common to both histories, weighted by the IC of each fact-set. Intuitively, we use these weights since fact-sets with higher IC are more specific, and thus similar support for such fact-sets is more meaningful.[1] Here again we provide a PTIME algorithm for computing support similarity, employing optimization analogous to the one mentioned above.

[1] Our metric treats fact-sets that are not associated with support scores (e.g., in member profiles) as having the maximal support, 1.

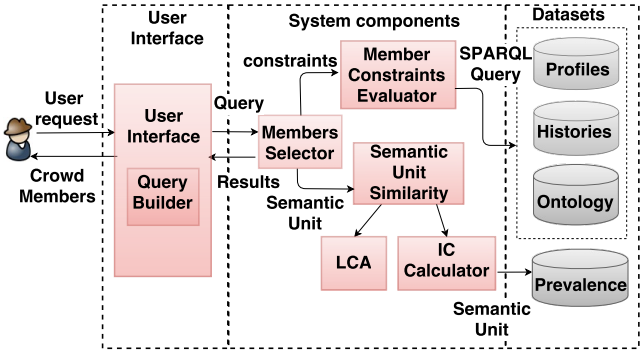

Figure 2: System Architecture

3. SYSTEM OVERVIEW

December is implemented in Java 8, uses Apache.Jena (`https://jena.apache.org`) for handling RDF data and the JGraphT (`http://jgrapht.org`) library for graph representation of taxonomies. Figure 2 depicts the system architecture. On the right are the RDF repositories over which Member-QL queries are evaluated: User profiles, histories and a domain ontology. Starting from the left, the user constructs Member-QL queries via the *User Interface*, which are then submitted for evaluation. The *User Selector* module parses the query and identifies the standard RDF selections, which are then converted by our *User Constrains Evaluator* to SPARQL queries to be evaluated by Apache.Jena. The results are used to form *semantic units* (fact-sets) over which similarity operators are evaluated according to the Member-QL query. This is performed by the *Semantic Unit Similarity* module which employs two sub-modules: the LCA (least common ancestor) module is used for computing the information common to two semantic units; the *Information Content Calculator* computes the IC of semantic units.

Query Builder UI. December has two types of query building interfaces implemented in HTML5/CSS3. First, a user-friendly UI for novice users that provides simple means to construct a crowd selection query: The users can add hard constraints on the profiles/histories of crowd members; ask for crowd members with similar profiles or histories to themselves (and add constraints to define which parts of the profiles/histories to compare); specify thresholds for support and/or similarity; and finally define the result set size and order. To further assist the users, we implemented an auto-complete feature that suggests relevant terms from the ontology when the user starts typing in the constraints text box. Second, advanced users can construct and execute complicated queries in native Member-QL via a text-based editor. In both cases, the query results are presented to the user as exemplified in Figure 4, allowing the user to view a summary of the crowd members' profiles and an explanation of their rankings. (For privacy preservation, the hosting system should allow users to specify which parts of their profile/history can be made public).

4. DEMONSTRATION

For the demo, we connected December to the `OASSIS` crowd data sourcing platform [2] that allows users to formulate questions and have them answered via the crowd. The

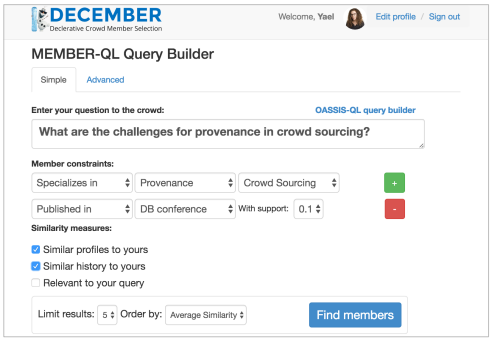

Figure 3: Query Builder

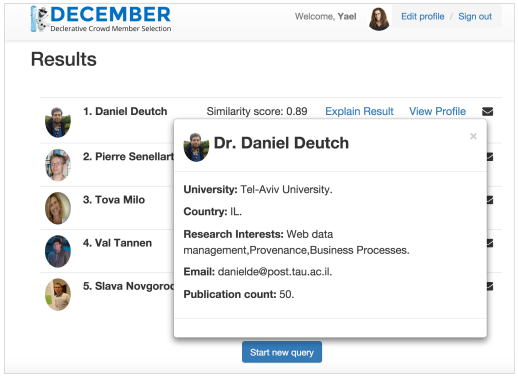

Figure 4: Results

VLDB community serves as the crowd-base of OASSIS, and every researcher who has ever published in VLDB is considered a crowd member. The conference participants will be invited to pose various questions in OASSIS on database related topics and identify for each question the relevant crowd members via December's friendly UI, and finally inspect its effectiveness on the selected crowd.

To create an initial profile for each crowd member, we use the AMiner dataset [10] which includes details about the researcher's affiliation, position, areas of research, etc. We further complete this data with details about hobbies and areas of interest obtained via crawling the public homepages of researchers. Additionally, we create a history for each crowd member which consists of publications and collaborations also extracted from AMiner. Each publication/collaboration between two or more authors is converted into a fact-set with a support value corresponding to its frequency in the dataset. As for the ontology, we use a hierarchy of categories and pages mined from DBpedia (http://dbpedia.org).

We will start the demonstration by exemplifying the use of December and its UI by creating several example Member-QL queries. Then, audience members will be invited first to edit and enrich their personal profiles, then to compose a question to the crowd via OASSIS's natural language interface (as depicted in the upper part of Figure 3), and use December's query builder interface to choose the adequate crowd for their question. Using December UI, the user may pose hard constraints on the crowd members' profiles and histories, e.g. on their areas of research as well as similarity preferences for profiles and histories (Figure 3).

To highlight the capabilities of December, we will use multiple screens and ask participants with different profiles and histories to login to the system and run the same Member-QL query in parallel. The audience will be able to see that queries that require crowd members with similar backgrounds or statures yield different results for different users, and each user will be able to verify the adequacy of the selected members. For example, it is natural for a senior leading researcher looking for some background material in a given topic to consult the top experts in the field, whereas a junior student may prefer to consult with other students or junior researchers first. This can be naturally expressed in Member-QL via similarity preferences on the profile and history of the selected crowd.

In addition, few more usage scenarios will be exemplified such as a researcher looking for possible future collaborators, or a PhD student looking for a postdoc host by selecting tenured professors that resemble her in terms of research topics, past collaborations, and even hobbies and interests.

In each case the Member-QL query will be auto generated from the UI and executed, producing a list of researchers, ranked by their relevance to answer the question asked (as in Figure 4). Our audience will be able to review the relevant data about each researcher using the *Explain* and *View Profile* buttons and thereby gain an intuition about the ranking. Finally, the IDs of the selected crowd members are passed back to OASSIS, for obtaining answers to the given question.

Finally, to provide further intuition on the definition and evaluation of the similarity constructs used by the queries, we will allow the audience to look under the hood of December via its administrator screen. We will show the computation leading to the selection of crowd members, the commonalities between the profile/history of the given user and crowd members, and the computation of similarity scores.

Acknowledgments. We thank Susan Davidson for fruitful discussions of the query language. This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 29107.

5. REFERENCES

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

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

[3] S. Ben Ticha, A. Roussanaly, A. Boyer, and K. Bsaïes. User semantic preferences for collaborative recommendations. In *EC-Web*, 2012.

[4] A. Bozzon, M. Brambilla, S. Ceri, M. Silvestri, and G. Vesci. Choosing the right crowd: expert finding in social networks. In *EDBT*, 2013.

[5] T. Di Noia, R. Mirizzi, V. C. Ostuni, D. Romito, and M. Zanker. Linked open data to support content-based recommender systems. In *I-SEMANTICS*, 2012.

[6] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux. Pick-a-crowd: Tell me what you like, and i'll tell you what to do. In *WWW*, 2013.

[7] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. iCrowd: An adaptive crowdsourcing framework. In *SIGMOD*, 2015.

[8] H. Rahman, S. Thirumuruganathan, S. B. Roy, S. Amer-Yahia, and G. Das. Worker skill estimation in team-based tasks. *PVLDB*, 8(11), 2015.

[9] N. Seco, T. Veale, and J. Hayes. An intrinsic information content metric for semantic similarity in WordNet. In *ECAI*, 2004.

[10] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: Extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008.

[11] Z. Zhao, J. Cheng, F. Wei, M. Zhou, W. Ng, and Y. Wu. SocialTransfer: Transferring social knowledge for cold-start cowdsourcing. In *CIKM*, 2014.