Answering Search Queries with CrowdSearcher

Alessandro Bozzon, Marco Brambilla, Stefano Ceri
Politecnico di Milano, Dipartimento di Elettronica ed Informazione, Via Ponzio 34/5, 20133 Milano, Italy
{alessandro.bozzon, marco.brambilla, stefano.ceri}@ polimi.it

ABSTRACT

Web users are increasingly relying on social interaction to complete and validate the results of their search activities. While search systems are superior machines to get world-wide information, the opinions collected within friends and expert/local communities can ultimately determine our decisions: human curiosity and creativity is often capable of going much beyond the capabilities of search systems in scouting "interesting" results, or suggesting new, unexpected search directions. Such personalized interaction occurs in most times aside of the search systems and processes, possibly instrumented and mediated by a social network; when such interaction is completed and users resort to the use of search systems, they do it through new queries, loosely related to the previous search or to the social interaction. In this paper we propose CrowdSearcher, a novel search paradigm that embodies crowds as first-class sources for the information seeking process. CrowdSearcher aims at filling the gap between generalized search systems, which operate upon world-wide information - including facts and recommendations as crawled and indexed by computerized systems – with social systems, capable of interacting with real people, in real time, to capture their opinions, suggestions, emotions. The technical contribution of this paper is the discussion of a model and architecture for integrating computerized search with human interaction, by showing how search systems can drive and encapsulate social systems. In particular we show how social platforms, such as Facebook, LinkedIn and Twitter, can be used for crowdsourcing search-related tasks; we demonstrate our approach with several prototypes and we report on our experiment upon real user communities.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *Search process*. H.5.4 [**Information Interfaces and Presentation**]: Hypertext/ Hypermedia.

General Terms

Design, Experimentation, Human Factors.

Keywords

Exploratory search, Multi-domain search, Information Seeking, Crowd sourcing, Social network, Search service, Search engine.

1. INTRODUCTION

Questions such as: "Which is the most romantic beach in the Caribbean?", or "Which is the most lively and trendy district in London?", which are preliminary to decisions for vacation or housing selection, cannot be responded by automatic calculations of any sort. In these and many other scenarios, professional- and consumer- oriented, the human insight or opinion is considered as more valuable than mere factual information. Currently, the only way to obtain this kind of information is through individual interactions with friends or colleagues[1], either through offline communication or via messages manually posted on social networking platforms. However, the emergence of the crowdsourcing paradigm, consisting of "asking the crowds" [2], could introduce a significant change to search behavior. Substantial research work has addressed crowdsourcing from different perspectives and within various communities, including information retrieval [3], databases [4-6], artificial intelligence, social sciences, and so on. Industry is heavily moving toward crowdsourcing, as demonstrated by the high number of new start-ups that exploit crowdsourcing, social networking, social participation, humans as sensors, and other variants of this trend.

Crowdsourcing so far is not explicitly integrated with search. People often make up their minds by combining results from search engines, investigations on vertical portals, and opinions gathered within their friends and trusted people circles [1], but there is no systematic way of linking search results to crowdsourcing. In this paper, we aim at closing the gap, by introducing CrowdSearcher, a system architecture with associated query and execution model that bridges conventional search experiences to crowdsearching and social network exploitation – a crowdsourced search activity.

The key aspect of the CrowdSearcher query model is a simple form of data sharing between a conventional search engine and a social engine; the two environments communicate through selected items which are produced by the conventional engine and proposed as the backbone of the crowdsearching activity. The CrowdSearcher query model comprises a number of structured queries over such shared data, which support the classical actions of liking/disliking, tagging, expressing preferences, ranking, inserting, deleting, correcting and so on, that on one hand can be formally specified, and on the other hand are intuitively expressed even by naïve users. Because of their nature, however, it may be inappropriate to submit such activities to "random workers"; therefore, a key aspect of crowdsearching is the possibility of using social platform for selecting the "crowd".

A CrowdSearcher query is routed to living users with human reaction times and as such is perceived by a conventional search engine as an asynchronous process; however, the results that are possibly and eventually produced by a query have a structure, and therefore can be fed again into a conventional search engine. Therefore, the proposed model fits well in a context where search is seen as a complex, long lived process [7] which may be long-lasting and may involve users in a number of interactions – such as buying a house or deciding a summer trip; such approach is typically classified as exploratory search [8]. CrowdSearcher aims at studying the way in which human computation, community feedback, and crowd opinions can be integrated in exploratory search for augmenting the answers to queries produced by purely automatic information retrieval systems.

This approach is inspired by various systems which have been recently proposed for bridging data and human sources, such as

CrowdDB [4], Turk [5] and DeCo [6]. There are, however, significant differences. First, the crowd's role in those systems is essentially to be "data producer", thereby allowing the filling of tables or table columns in a system that seamlessly integrates conventional and crowd sources. We believe that such a role significantly limits the tasks that one would naturally assign to a crowd – a typical one is to express sentiments such as like or dislike. Moreover, we think that a seamless integration is improper, as users normally must be aware of the actions which lead to crowdsearching, and should imperatively dominate such integration, by expressing its scope and limitations (including temporal ones); therefore, we believe that using optimizers for planning activities which interleave data and crowd sources, although architecturally feasible, should not be advised. We also believe that, although a user may need several crowdsearching interactions to complete a task, every interaction should be simple and focused. Therefore, in our model, simple crowdsearching sessions are spawned by users from within possibly long and complex conventional search sessions, and each crowdsearching session is independent.

One important design principle for CrowdSearcher is the independence of queries from the crowdsearching engines. This term denotes a broad class of solutions: simple queries could be orchestrated even by using email and shared documents, e.g. Google Tables. In this paper, we show that social platforms (such as Facebook, LinkedIn, Twitter) can be used for crowdsearching together with classic crowdsourcing systems (such as Amazon Turk).

Therefore, we define the *CrowdSearcher Query Language* as a mapping from an *Input Model*, including a dataset and structured queries, to an *Output Model*, which is obtained by modifying the dataset and by adding the answers to structured queries; the choice of crowdsearching engine and the selection of the crowd is performed by the user; some structured queries may not be supported by given engines. When the user queries are too complex for a specific crowdsearching task – e.g., the ordering of collections of hundreds of items, or the extraction of the "best" item from such large collections – they can be split into simpler crowd queries, whose results are then composed; such a process is transparent to the user.

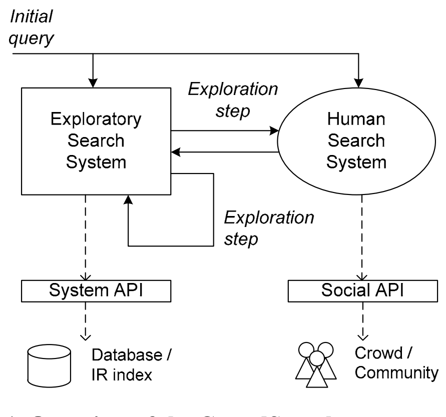


Fig. 1. Overview of the CrowdSearcher approach.

The corresponding high-level architecture is described in Fig. 1: the user submits an initial query, which can be addressed either to a traditional exploratory search system or to a human search system. If the interaction starts on the conventional search system (e.g., a vertical search system for real estate, events, travels, businesses), it interacts synchronously with data sources and produces several solutions (e.g. house offers, concerts, itineraries, restaurants, hotels). Users open the CrowdSearcher interaction by selecting some of those objects and asking questions about them; input selection and preparation is performed by the user with the help of ad-hoc web applications or wrappers. In principle, even the results produced by conventional search engines, such as Google, could be used as input sources, although in such case the user should build the input manually by extracting structured information for each selected result item. As an alternative, the user can immediately start with a human search step; in this case, the query formulation and the inputs are directly provided by the user. At the end of the CrowdSearcher interaction, results are presented to the user; in some cases, results are presented in a format that allows their acquisition by the search system, thereby enabling a seamless continuation of the search process with subsequent explorations; otherwise, data integration occurs in the user's brain, who can decide how to best operate on the initial alternatives based on the feedback from the crowd.

In our demonstrations, we use the Search Computing (SeCo) framework, and specifically its Liquid Query interface [9], which supports exploratory search upon generic data sources [10]. The SeCo framework supports exploratory search natively and produces at each exploratory step a tabular representation of the best "combinations"; therefore, SeCo is an ideal environment for supporting CrowdSearcher interactions, both to and from the chosen crowdsearching engine. The SeCo query orchestrator has been extended to support long-lived sessions, which can be suspended when the CrowdSearcher query is issued, and resumed when its results have been produced.

This paper is organized as follows. The engine-independent CrowdSearcher query language is defined in Section 2; then, Section 3 describes several technological solutions for mapping the CrowdSearcher query language to social platforms, and Section 4 shows an architecture for exploratory search which integrates the Search Computing platform with Facebook and Doodle. Section 5 illustrates some experiments and shows that, in the context of crowdsearching, social platforms have the potential of outperforming work distribution platforms. We finally relate crowdsearching to classical crowdsourch research and present our conclusions.

2. CrowdSearcher QUERY LANGUAGE

We define a CrowdSearcher query as a transformation of an input model into an output model, produced by crowdsearching or social networking engines interacting with people in real time; a mapping scheme selects the engines, the query representation for each engine, and the resources that should be used by the engine for answering the query. The model is very general so as to include many kinds of queries; some transformations are not supported by all engines, as discussed in the mapping scheme. In most practical situations, however, queries will use a small subset of the input model, thereby producing the output model through a simple transformation and mapping.

2.1 Input Model

The input of a CrowdSearcher query Q^I is a triple <C,N,S> where C is a data collection, N is a textual query expressed in natural language, and S is a collection of structured queries. Every component is optional. We next detail each component.

- C is an initial data collection which is proposed to the crowd for crowdsearching. For ease of description, we use the

relational model, and therefore C is a collection of tuples; card(C) denotes its cardinality. C is described by means of a schema sch(C), which contains the name, type, and constraints (e.g. NOT NULL) of C's attributes. We also assume that each tuple has an identifier TID. C can be sorted, in which case an attribute POS indicates the position of each tuple in the input sorting.
- N is a natural language query that is presented to the crowd. It can be mechanically generated, e.g. in relationship with specific structured queries, or instead be written by the user who starts the crowd search.
- S is a collection of structured queries that are asked to the crowd, relative to the collection C. Queries allow to express preferences about the elements of C, to rank them, cluster them, and change their content.

Preference queries correspond to typical social interactions (like, dislike, comment, tag); the other structured queries abstract simple and classical primitives of relational query languages which are common in human computation and social computation activities.

The **preference queries** include:

- **Like query**, counting the number of individuals who like specific tuples of C.
- **Dislike query**, counting the number of individuals who dislike specific tuples of C.
- **Recommend query**, asking users to provide recommendations about specific tuples of C.
- **Tag query**, asking users to provide either global tags or tags about specific attributes of C.

The **rank queries** include:

- **Score query**, asking users to assign a score (in the 1..*N* interval) to tuples of C.
- **Order query**, asking users to order the (top *N*) tuples in C.

The **cluster queries** include:

- **Group query**, asking users to cluster the tuples in C into (at most *N*) distinct groups.
- **OrderGroup query**, asking users to cluster the tuples in C into (at most N) distinct groups and then order the (top *M*) tuples in each group.
- **MergeGroup query,** asking users to merge *N* sorted groups producing a single ordering.
- **TopGroup query,** asking users to cluster the tuples in C into (at most *N*) distinct groups and then select the top element of each group.

The **modification queries** include:

- **Insert query**, asking users to add tuples to C.
- **Delete query**, asking users to delete tuples from C.
- **Correct queries**, asking users to identify and possibly correct errors in the tuples of C.
- **Connect query**, asking users to match pairs of similar tuples.

We propose 3 examples.

1. C is a set of 10 football players, presented through their Name and Photo (therefore, Sch(C)=[TID, Name:text, Photo:jpeg]). N is "Which are your favorites players?", S:Like.
2. C is a set of 3 restaurants in Brera/Milan, presented through their photo and address (then, Sch(C)=[TID, Name:text, Address:coordinates, Photo:jpeg]). N="choose restaurant with good meat + good price txs!!!!". The user issues three queries: Like, Tag, and Order.
3. C is a set of 100 students currently listed in a class, presented through their name and email (then, Sch(C)=[TID, Name:text, Email:text]). N="please add name and email if you attend the class and are not listed, and drop your name if you decided not to attend". The user issues the Insert and Delete queries to all master's students.

2.2 Output Model

The output of a CrowdSearcher query Q^O is a tuple <C',S'> where C' is a data collection and S' is a collection of structured answers. It is produced by CrowdSearcher engines and delivered asynchronously. C' is the data collection which is returned to a user after a crowdsearching task. The schema Sch(C') is obtained by adding to Sch(C) attributes which are used as slots for the answers to the structured queries S in S', i.e.:

- A counter L of people whom like each tuple.
- A counter D of people who disliked each tuple.
- A score value S representing the average score given by people to each tuple.
- A list R of character strings of people who added *N* recommendations to tuples.
- A list T of terms (simple or compound words) of people who tagged each tuple.
- A tuple identifier POS if the users ordered the tuples.
- A group identifier GID if the users clustered the tuples into groups.
- A group position GPOS if the users ordered the clusters (GPOS is repeated for each tuple in the cluster).

Tuples of C' are obtained from tuples of C after insertions, deletions, and updates of tuples in S; S' are the answers to the queries S, placed within the appropriate slots – the new attributes in Sch(C').

The result in the 3 examples above could be as follows:

1. C' is the set of players with the additional attribute Like (then Sch(C')=[TID Name:text, Photo:jpeg, Like: integer] which stores the counts of people who liked the 10 players.
2. C' is a list of 3 restaurants in Brera/Milan with Like and Tag added (then Sch(C')=[TID, POS, Name:text, Address:coordinates, Photo:jpeg, Like:integer, Tags: array(text)], POS contains the values (1,2,3), the Like attribute contains the counts of people who liked each restaurant, and the Tag attribute has an array of tags for each restaurant.
3. C' is the set of 105 students listed in a class, obtained from C by inserting 8 students and deleting 3.

2.3 Mapping Model

The mapping model specifies how given search engines can be involved in producing the output of a given query Q. A Mapping M of a query Q is a quintuple <E, G, H, T, D> where:

- E is the crowd engine or engines to be used in the query.
- G is the crowd group that should interact with engine. This could be: the user's friends, specific subsets of the user's friends, geo-localized people, expert people, workers selected on a work platform, and so on. We denote as G(E) the subset of users in G that are accessible through a given crowd engine E.
- H represents constraints in the execution of a query Q; in particular, it indicates which conditions should hold for a query to be terminated. For instance, collecting at least K answers from H different users, or lasting for three minutes.
- T is a transformation process, which applies to a query Q and transforms it into smaller queries Q' such that answering Q'

and then combining their results allows answering Q. A transformation is needed when the query Q is too complex to be directly proposed to a crowd engine.

- D is the set of templates used to present C and express structured queries for a given crowdsourced query. Templates are typically targeted to specific crowd engines; we denote as D(E) the subset of templates that can be submitted to a given engine E. Templates for displaying C have a style, e.g. textual, tabular, geo-referenced on a map, time-referenced on a time line, represented as the points of a Cartesian space, and so on.

The mapping in the 3 examples above could be as follows:

1. For the first example: E is Facebook, G are the user's friends, H is 3hrs. D and T are not needed.
2. For the second example: E is Facebook, G are the user's friends living in Milano, H is 3 mins, D is geoeferenced (a small map including the three restaurants with a small photo), T is not needed.
3. For the third example: E is a combination of conventional email and one Google Table. G are the university's master students, D is tabular. T is not needed.

3. MAPPING TO PLATFORMS

The mapping model of Section 2.3 can be deployed in many ways, due to the variety of the human and social platforms for interacting with the human responders and of the diversity of the interfaces that they expose for interaction from outside; in this Section, we classify and discuss the architectural options.

3.1 Architecture overview

A **Crowd Search Management System** (CSMS) implements the queries upon human and social platforms. The system instantiates query templates by importing information from search systems, sends queries to the social/crowd platform, gets a collection of responders involved, and gathers the results. The CSMS acts in the context provided by a given social/crowd platform user, denoted as **query master**, who is instrumental to the crowdsourcing process, by being responsible (and possibly covering costs) of tasks which are spawn to the crowd and by offering friends and colleagues as responders.

Ultimately, the social/crowd platform must import a query through a dedicated interface, and this can be achieved by either embedding the query interface as a new application of the platform, or directly using the native APIs offered by each platform, as shown in Fig. 2. In details:

- With an **embedded application**, it is possible to use the social/crowd platform for embedding a CSMS-specific client that directly interacts with the CSMS server. This option is the most powerful for supporting a description of the initial collection and of structured queries upon them; however, it requires users of the platform to install and load the embedded application. Embedding may occur either just in the context of the query master or also in the context of the master's friends or colleagues.
- With the direct use of the **external API**, the CSMS behaves as an external application that is directly using the native features of the social platform for creating queries and collecting results. This guarantees higher transparency but forces the query to be expressed sometimes in a less natural and explicit way.

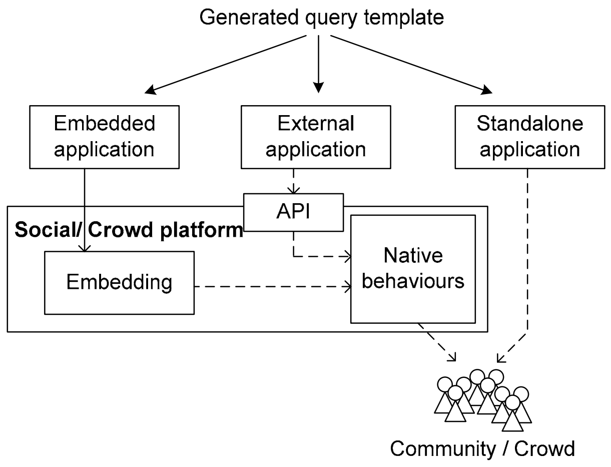


Fig. 2. CrowdSearcher architecture overview.

A CSMS could directly interact with communities and crowds without the mediation of a social platform; we omit to further discuss this option, which is however included for completeness.

The adoption of each platform by the user may be subject to an **initialization process**. For instance, MechanicalTurk requires the master user to register and to give proof of being able to cover the costs of work tasks[1]; with Facebook, the master user must initially install an application which authorizes the CSMS platform to communicate by using the master user's identity, and so on.

3.2 Execution process

The execution of one of a structured query defined in Section 2.1 may be conducted according to several execution strategies, e.g. including routing and splitting of tasks:

- **Task splitting** occurs when the collection is too complex relative to the cognitive capabilities of users or to the limitations imposed by the social platform.
- **Task routing** occurs when a task can be distributed according to the values of some attribute of the collection; for instance, if the structured query is an "Insert" for more restaurants, the attribute City can be used to route the queries to specific users living in that same City.

Splitting strategies depend on the specific social platform being used, although some fundamental problems are independent on the choice of the platform. They have been studied for join and sort operations [11] in the context of the Qurk system [5] that operates upon AmazonTurk, and they have more generally discussed in the context of work-task splitting [12-14] for that system. Task routing requires accessing profiles while selecting crowds. Presence indicators could be used for routing queries to users of a given physical location.

The concrete steps performed by the CSMS system for defining the mapping model and executing the queries are shown in Fig. 3: platform and user selection respectively define the crowd engine E to be used and the group of users G(E) that will be targeted, setting at the same time the constraints H on query termination. Based on these choices and on the input model, the query is possibly split and routed to groups of users, and then query

[1] In addition, MechanicalTurk constraints job creators to provide a US address.

template is generated by choosing among supported templates D(E).

The social platform users are then engaged by inviting them to reply to the question respond with their contribution. Finally, replies are collected by the CSMS and manipulated for generating the final output model.

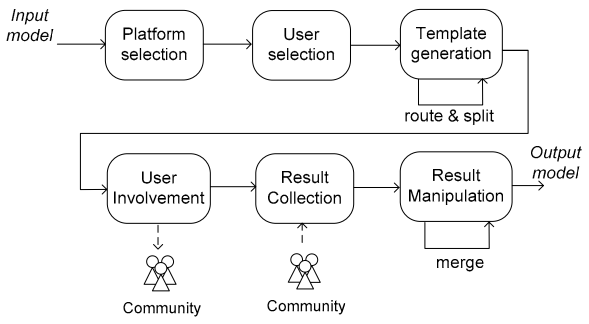


Fig. 3. Execution process for CrowdSearch query.

3.3 Mapping Table

Social platforms support the display of a population and the various structured queries at various degrees. The coverage of the structured queries by a specific social platform is influenced by two aspects: the *existing interaction paradigms* available on the platform; and the *cardinality of the data set* upon which the paradigms are supported (i.e., whether the paradigm can be enacted on a singleton or on a set). In particular, some structured queries, e.g. Like, are natively supported by given systems, e.g. Facebook; other queries, e.g. Rank, can be constructed, however less "natively", by counting the number of likes. However, even the supported interaction paradigms such as the Like may be oriented to singletons (like in FaceBook, where the like applies to single posts or comments), instead of sets (like in Doodle, where the user can select several preferences starting from a list). Considering these aspects, we have studied how the most popular social platforms support the structured queries of Section 2.1; Table 1 shows the result of the study. We classify entries as:

- **Native (N)**, when the structured query is natively supported by the platform.
- **Work around (W)**, when the structured query requires a workaround which is however still rather intuitive (e.g., ranking by counting the number of likes).
- **Indirect (I)**, when the structured query requires an indirect expression which abuses of the platform's functionality (e.g. counting dislike by counting the number of likes on a query's negation).
- **Not supported (-)**, when the structured query is not supported.

For every possible combination of social platform and structured operation listed in Table 1, a CrowdSearcher system can be configured with a platform-specific template that defines the query structure. Each template includes the basic elements of the query, such as the number of inputs, the allowed user interaction, the structure of the output, and so on. Through the template, the master user specifies the actual query parameters (textual question, query population, structured queries, involved platform, involved community), which are filled in during the query formulation phase. A single template could be used for several structured queries, but we expect this option to be rarely used because the cost for users to reuse templates may be higher than defining new ones.

The mapping of Table 1 applies to the templates which use native APIs of each social platform, while embedded applications (e.g. MechanicalTurk and Facebook apps) clearly support all features in a natural way, because the developer can build the template of the query with the maximum programming freedom (they are tailor-made web applications). Therefore, we omitted rows for MechanicalTurk and a Facebook Applications, which would appear in the table with all entries as *native*. We also omitted columns for grouping queries, as they can be achieved only by embedded applications.

This table is only of indicative nature[2], as some additional mappings might be possible by using "hidden" features of the engines – e.g., that are discovered by composing results in nontrivial ways; moreover, APIs are in continuous evolution. We are also considering GoogleTables as a "potential social engine" for collecting structured queries (in such case, the master user creates a Google table and invites interactions on the tables themselves through emails). However, the table shows the large number of queries that, once modeled through the proposed input/output model, can be responded by current social engines. In the next section we consider some specific entries in the table.

Table 1. Mapping of CrowdSearcher structured queries to social platforms using their APIs.

	Like	Dislike	Comment	Tag	Score	Order	Insert	Delete	Correct	Connect
Facebook	N	I	W	W	W	I	W	I	W	W
LinkedIn	N	I	W	W	W	I	W	I	W	W
Twitter	W	W	N	W	I	I	N	I	W	W
Doodle	N	N	-	-	-	-	-	N	-	N
Google Tables	I	I	N	W	W	W	N	N	N	I

4. EXPLORATORY USER EXPERIENCE

We have developed CrowdSearcher as a CSMS prototype that combines exploratory search with crowdsearching over social platforms. The framework currently integrates a Search Computing (SeCo) application, providing the exploratory search functionalities, with Facebook and Doodle, providing the social networking capabilities (integration with LinkedIn and Twitter is ongoing). We have used API-based solutions for accessing FaceBook and Doodle, and we have chosen Facebook for embedding a CrowdSearcher application[3].

[2] The table is a photograph of today's snapshot over an evolving world; for instance, we expect that Google+ will soon be added to this table, as the publication of the system's API is expected in short term.

[3] Embedding within Mechanical Turk is feasible along the experience of [4-6], but for crowdsearching we prefer a social platform to a working environment.

The application embedded within FaceBook consists of a set of JSP templates containing the HTML and Javascript implementing the various query features. API-supported solutions require an imperative description of the query feature (e.g., create a new post and then several comments to the post), defined according to existing APIs primitives; one different template is defined *a priori* for every query type and platform. For instance, the Like operation is concretely implemented in FaceBook by asking a question and then showing the query population, with each item associated with the classic FaceBook *Like* icon and control; in Doodle, the Like operation is defined by selecting the various items of the collection via a YES mark.

The framework covers the following phases:

- Structured query formulation by the exploratory search system;
- Crowd query formulation based on the results of the preceding exploratory step;
- Query submission to the crowd and collection of responses;
- Query result computation and rendering to the master user of the crowd result, seamlessly integrated with the possible subsequent exploration steps.

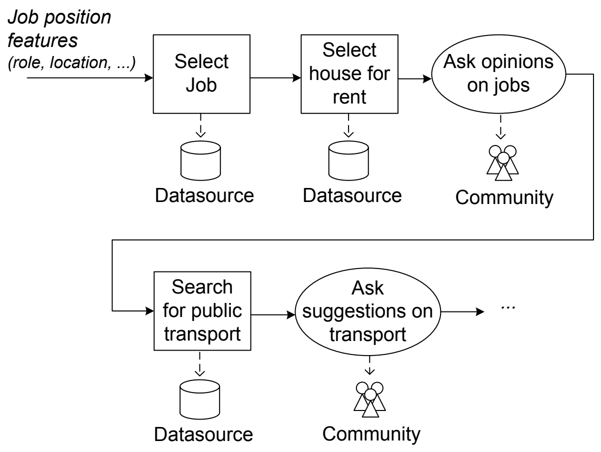
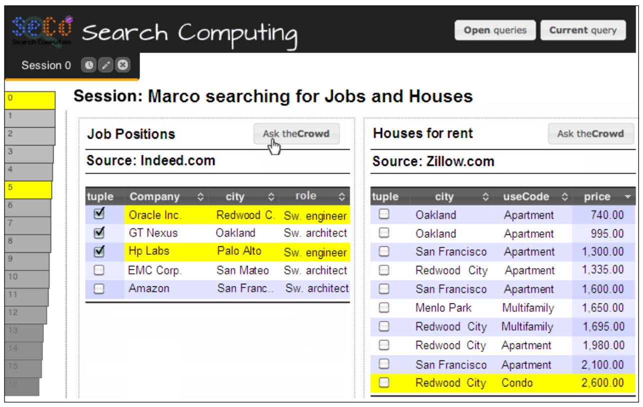


Fig. 4. CrowdSearcher experimental setting for the scenario based on exploration of jobs and houses.

For describing the user interaction we define a sample scenario based on job and house search, as shown in Fig. 4, whose exploratory part alone was demonstrated at WWW 2011 [10].
The (master) user is searching for job offers and rentals in a given area. Through the Search Computing system, the user can access job and house offers which are ranked by job affinity with the user's profile, selected by type and cost of rental, and ranked by distance to each job location; the system presents combinations and rankings. At this point, the user could decide that she likes to get advices from either the community of friends who are in the job marked about the selected job, or from the community of friends who already live in the area about the rental offices. Such step requires the interaction with a social source, e.g. LinkedIn for the former advice and Facebook for the latter advice. After getting the advices, she may refine the solution, select one job offer and a few housing offers, and then scouting the transportation options between each pair, first by using a transportation data source integrated within the application, and then asking the community, e.g. about comfort, punctuality, and so on.

4.1 Exploratory Query Step

Exploration is enabled through a SeCo user interface where the user can submit a structured query upon one or more domains and then can proceed with the exploration and selection of the items according to the Liquid Query approach described in [9]. Fig. 5 shows an example of the result set generated for the above scenario, which lists several job and housing offers as obtained through a past history of exploratory steps. At this point, the user can further explore by investigating additional domains available from within the SeCo System, or instead opt for one or more CrowdSearcher steps, thereby asking for comments, opinions or suggestions to the crowd, by clicking on the "Ask the crowd" button. We currently support domain-specific queries, as queries about combinations are more complex and generally it is better to split complex crowdsearching tasks into simple tasks [11-13]. Therefore, from within the "atom view" of the SeCo system [10], the user can invoke the CrowdSearcher application either on Jobs or on Houses, after selecting some of the items produced by the system; the selected items are imported within the crowd query formulation phase.



Session: Marco searching for Jobs and Houses

Job Positions

Source: Indeed.com

tuple	Company	city	role
✓	Oracle Inc.	Redwood C.	Sw. engineer
✓	GT Nexus	Oakland	Sw. architect
✓	Hp Labs	Palo Alto	Sw. engineer
☐	EMC Corp.	San Mateo	Sw. architect
☐	Amazon	San Franc..	Sw. architect

Houses for rent

Source: Zillow.com

tuple	city	useCode	price
☐	Oakland	Apartment	740.00
☐	Oakland	Apartment	995.00
☐	San Francisco	Apartment	1,300.00
☐	Redwood City	Apartment	1,335.00
☐	San Francisco	Apartment	1,600.00
☐	Menlo Park	Multifamily	1,650.00
☐	Redwood City	Multifamily	1,695.00
☐	Redwood City	Apartment	1,980.00
☐	San Francisco	Apartment	2,100.00
☐	Redwood City	Condo	2,600.00

Fig. 5. Exploratory Interaction Results.

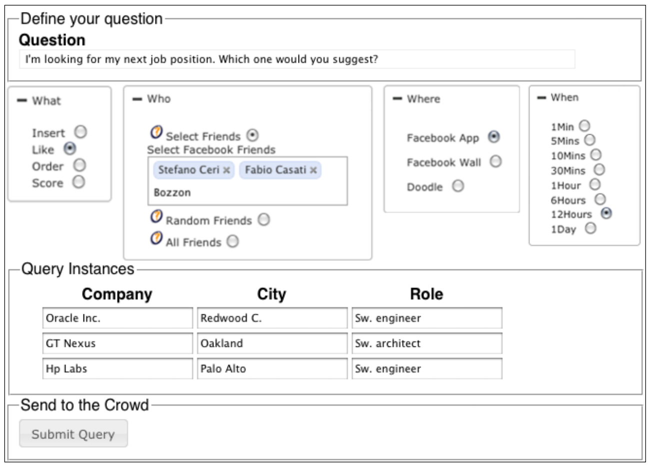
4.2 Crowd Query Formulation

The master user then defines a crowd query through the query template, which is integrated with the SeCo exploratory interface. The user can immediately see the list of results she selected in the exploratory interface and can define the textual question and then enter choices about various options:

- *What:* the type of structured query;
- *Where:* the platform of choice;
- *Who:* the people to invite;
- *When:* the duration of the crowdsearching step.

In the current prototype, options are exclusive; each query can be routed to Facebook or to Doodle, and the Facebook version can either be native or use an application developed *ad hoc*; in Facebook, a query can be routed to random or to selected friends; it can be more or less "urgent" (the time set by the user is included in the invitation. At the selected time, CrowdSearcher collects received results and summarizes them in the Query Result).

Fig. 6 shows a panel where the user can enter the textual question, then the structured query (currently just one), then the platform, the addressed users, and the expiration time of the experiment; the "crowd" of friends can either be selected randomly among all users or be enumerated by the master user[4].



Define your question

Question

I'm looking for my next job position. Which one would you suggest?

What: Insert ○ [✓ Like] Order ○ Score ○

Who: [✓ Select Friends]
Select Facebook Friends: Stefano Ceri ×, Fabio Casati ×, Bozzon
Random Friends ○
All Friends ○

Where: [✓ Facebook App] Facebook Wall ○ Doodle ○

When: 1Min ○ 5Mins ○ 10Mins ○ 30Mins ○ 1Hour ○ 6Hours ○ [✓ 12Hours] 1Day ○

Query Instances

Company	City	Role
Oracle Inc.	Redwood C.	Sw. engineer
GT Nexus	Oakland	Sw. architect
Hp Labs	Palo Alto	Sw. engineer

Send to the Crowd

Fig. 6. UI of the CrowdSearcher Question Formulation.

4.3 Crowd Query Answering

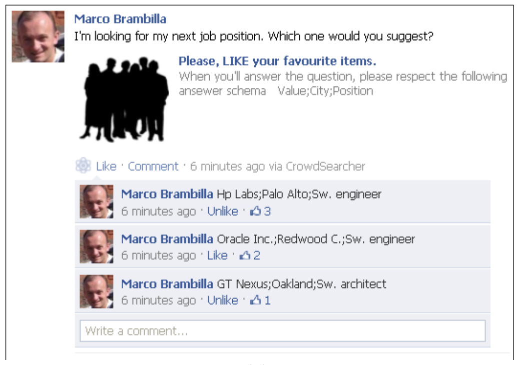
Based on the query setting, crowds receive an invitation to contribute their opinions. The way they can reply depends on the platform the user has chosen for collecting answers. Fig. 7 shows the currently implemented options in our system. In case (a), the query is posted on the Facebook's wall of the master user, and friends are invited to raise the counts of likes. This is a structured query with an intuitive, native support; other queries, such as comment, tag or score, can be easily described by work around solutions[5].

In case (b), the same query is posted as a Doodle poll, and users enter their preferences by responding YES to the poll. In case (c), a Facebook embedded application supports the ordering of the three jobs.

4.4 Result Computation and Rendering

When the community has responded to a query, the master user can inspect the query's result, which is compliant with the output model of the given query and is automatically exported back to the exploratory interface of SeCo. Fig. 8 shows that the original jobs have been augmented with a column "Like", whose content includes the counts of "Like" answers collected from the crowd.

Subsequently, the user can proceed from this state to further exploration steps and crowd questions.



Marco Brambilla
I'm looking for my next job position. Which one would you suggest?

Please, LIKE your favourite items.
When you'll answer the question, please respect the following ansewer schema Value;City;Position

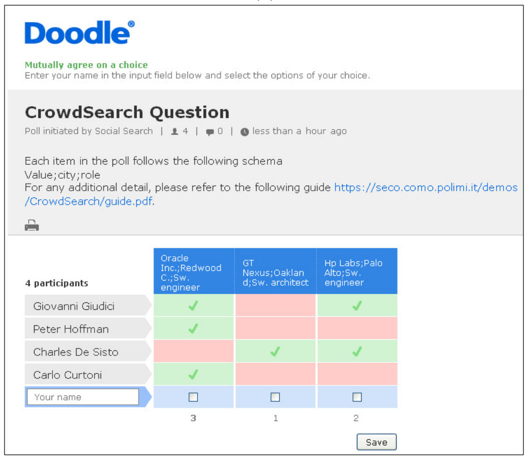
Like · Comment · 6 minutes ago via CrowdSearcher

Marco Brambilla Hp Labs;Palo Alto;Sw. engineer
6 minutes ago · Unlike · 3

Marco Brambilla Oracle Inc.;Redwood C.;Sw. engineer
6 minutes ago · Like · 2

Marco Brambilla GT Nexus;Oakland;Sw. architect
6 minutes ago · Unlike · 1

(a)



Mutually agree on a choice
Enter your name in the input field below and select the options of your choice.

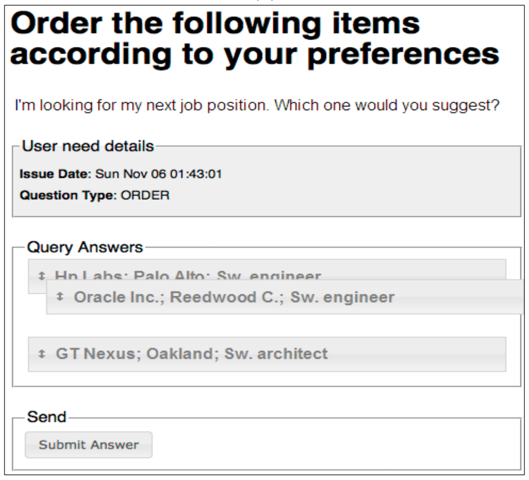
CrowdSearch Question

Poll initiated by Social Search | 4 | 0 | less than a hour ago

Each item in the poll follows the following schema
Value;city;role
For any additional detail, please refer to the following guide https://seco.como.polimi.it/demos/CrowdSearch/guide.pdf.

4 participants	Oracle Inc.;Redwood C.;Sw. engineer	GT Nexus;Oakland;Sw. architect	Hp Labs;Palo Alto;Sw. engineer
Giovanni Giudici	✓	🟥	✓
Peter Hoffman	✓	🟥	🟥
Charles De Sisto	🟥	✓	✓
Carlo Curtoni	✓	🟥	🟥
Your name	☐	☐	☐
	3	1	2

(b)



Order the following items according to your preferences

I'm looking for my next job position. Which one would you suggest?

User need details

Issue Date: Sun Nov 06 01:43:01

Question Type: ORDER

Query Answers

Hp Labs; Palo Alto; Sw. engineer
Oracle Inc.; Reedwood C.; Sw. engineer
GT Nexus; Oakland; Sw. architect

Send

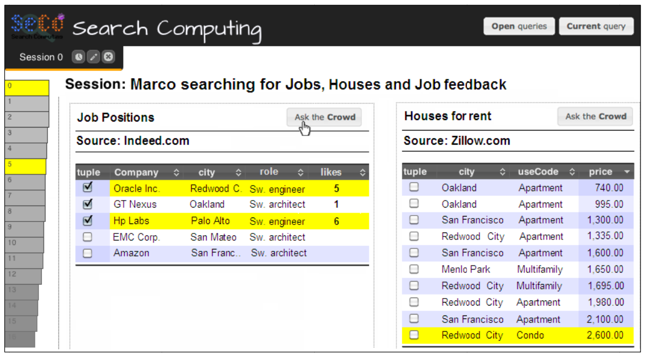
(c)

Fig. 7. Crowd Query Answering on alternative platforms: FaceBook (a), Doodle (b) and Facebook Application (c).

The system also comprises a dashboard screen, currently integrated within the SeCo UI, summarizing the status of all the open questions and respective answers for a user, shown in Fig. 9. The master user can access the results at any moment through his dashboard, by clicking on the "Open queries" button on the top-right corner of the screen. She can also use the dashboard to return to the original SeCo search contexts. This allows the management of intrinsically asynchronous queries such as the crowd ones.

[4] The Facebook application can invite a limited number of users in given time intervals.

[5] Counting and adding preferences trough Facebook Questions is currently not supported by the Facebook Graph APIs.



Session: Marco searching for Jobs, Houses and Job feedback

Job Positions

Ask the Crowd

Source: Indeed.com

tuple	Company	city	role	likes
☑	Oracle Inc.	Redwood C.	Sw. engineer	5
☑	GT Nexus	Oakland	Sw. architect	1
☑	Hp Labs	Palo Alto	Sw. engineer	6
☐	EMC Corp.	San Mateo	Sw. architect	
☐	Amazon	San Franc..	Sw. architect	

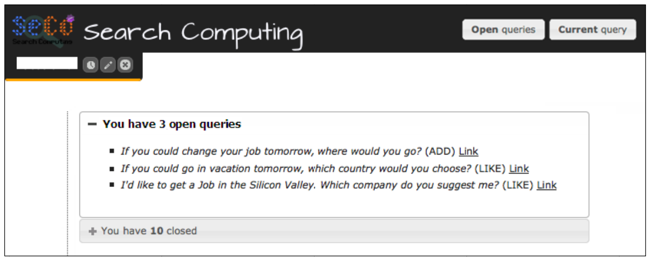
Houses for rent

Ask the Crowd

Source: Zillow.com

tuple	city	useCode	price
☐	Oakland	Apartment	740.00
☐	Oakland	Apartment	995.00
☐	San Francisco	Apartment	1,300.00
☐	Redwood City	Apartment	1,335.00
☐	San Francisco	Apartment	1,600.00
☐	Menlo Park	Multifamily	1,650.00
☐	Redwood City	Multifamily	1,695.00
☐	Redwood City	Apartment	1,980.00
☐	San Francisco	Apartment	2,100.00
☐	Redwood City	Condo	2,600.00

Fig. 8. Search Computing UI integrating the CrowdSearcher Results for a *Like* question.



− You have 3 open queries

- *If you could change your job tomorrow, where would you go?* (ADD) Link
- *If you could go in vacation tomorrow, which country would you choose?* (LIKE) Link
- *I'd like to get a Job in the Silicon Valley. Which company do you suggest me?* (LIKE) Link

+ You have **10** closed

Fig. 9. CrowdSearcher query dashboard.

5. EXPERIMENTS

In this section we present experiments that were conducted on our prototypes; 137 people were involved as master users, mostly in the 18-34 age group. While the initial group of 8 users belongs to our research group, the remaining ones were students in our classes or student's friends who voluntary offered to be part of the experiment. Each of them has associated the CrowdSearcher Facebook application[6] to the profile.

We defined two classes of queries: the former were predefined and of a general nature, with topics spanning from points of interests (e.g. restaurants in the vicinity of Politecnico), to famous 2011 songs, or to top-quality EU soccer teams; these queries presented collections of items that were retrieved as result of search steps, and then asked for preferences or additions. Each master user agreed to activate one or more randomly created query in the set and then routing it to randomly selected friends.

In addition, we asked users to build their own crowdsearching queries, by choosing a topic of their interest and by using the query specification interface shown in Figure 6; they also selected the specific friends to get involved. We regard these queries as *manual* in contrast to the *random* queries discussed above.

Users built 175 *like* and *insert* queries, sending 1536 invitations to friends. 95 questions (~55%) got at least one answer, summing up to a total of 230 collected answers (with an invitation-to-answer ratio of ~15%). The experiment has been performed during the last week of October 2011. Therefore, the results presented in this section depend on the features and limitations of the Facebook and Doodle platform as of October 2011, and on the level of involvement of our users, which could be measured (by reading the query results collected on our servers) but not controlled (each user autonomously decided to launch queries). Due to the Facebook's daily per-user API interaction limitations, the average number of allowed invitations per query was set to 15. Nonetheless, we believe that the results offered below provide a number of interesting observations about the usage of social platform to perform crowdsearching tasks.

Fig. 10a shows the mean number of answers distinguishing *random* queries from *manual* queries, and the two platforms. The histograms show that *manual* queries got in general more answers than *random* queries, and that Doodle got in general more answers than Facebook. The former comparison is an indication that participation is higher on friendship-related queries and with selected friends – as it could be expected. The second comparison is a consequence of the higher latency of queries in Doodle, which is also appreciated on Fig. 10b.

Figure 10b shows how answers, organized according to the addressed platform, accumulate over time: Facebook is most the most effective solution to get answers within the first hour from the query submission, but then Doodle becomes more effective, allowing to retrieve a higher number of answers. This is probably due to the fact that questions posted in Facebook walls camouflage with the standard information flow, thus soon becoming obsolete and less visible as new information pushes it away. Instead, given that Doodle queries are presented with dedicated visual widgets, the associated engagement is more stable, thus inviting people keep responding to invitations.

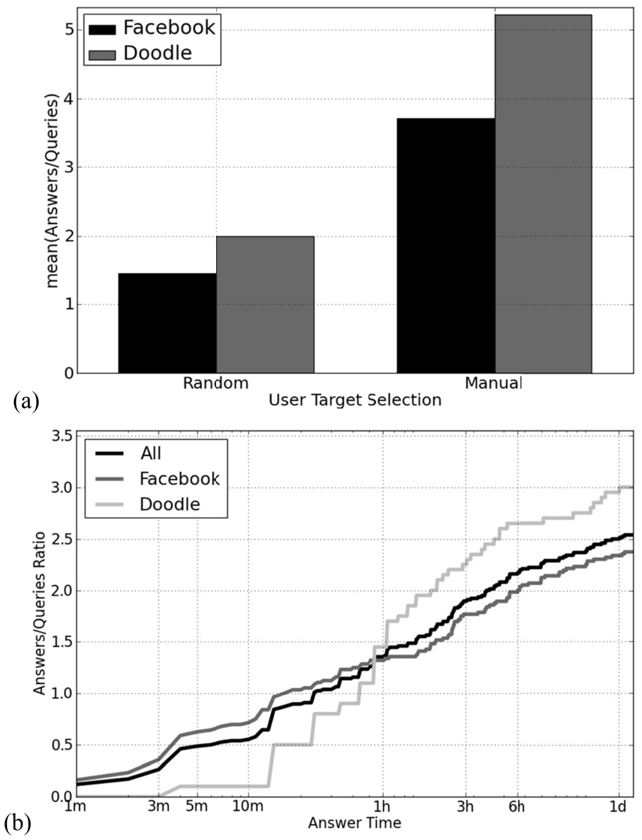
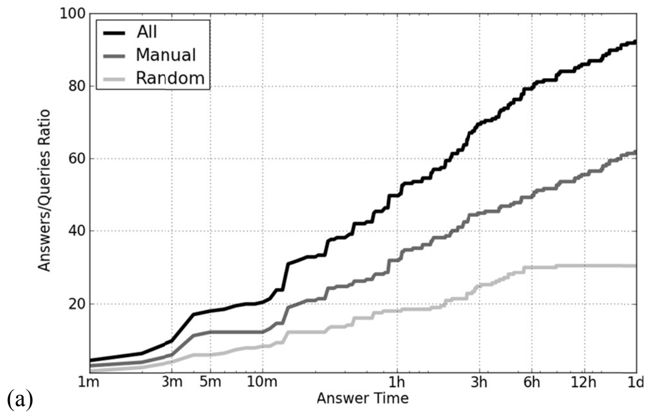
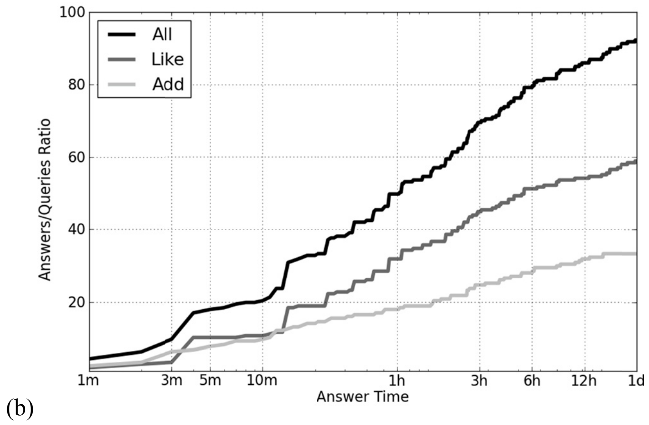


Fig. 10. (a) Average Answer-to-Question ratio *manual* vs *random* and *Facebook* vs Doodle. (b) Temporal production of answers, *Facebook* vs *Doodle* vs average of *all* answers.

[6] Available at: https://apps.facebook.com/crowd_search/



(a)



(b)

Fig. 11. Temporal production of answers: (a) *manual* vs. *random* routing of invitations; (b) *like* vs. *add* vs. *all* queries .

Fig. 11a shows that responding times to questions with *manual* routing are always and consistently shorter than to *random*ly-routed invitations. In other words, at a given time, the number of received responses to manually assigned questions is always higher than to the automatic ones. This is consistent with the aggregated view of Fig. 10a.

Fig. 11b shows that responding times to *like* and *insert* are initially very similar, but then answers to the *like* queries become predominant, probably because the *like* query better adapts to the kind of interaction which one expects on Facebook.

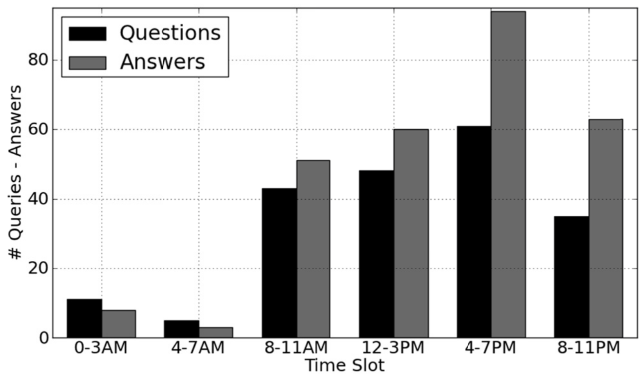


Fig. 12. Daily Distribution of questions and related amount of collected answers.

Engaging real people in providing search answers has obviously very different performance depending on the time of the day when the query is issued. Fig. 12 shows that the best time for getting answers (and consequently also the time when most queries are issued) is in the late afternoon, followed by the evening, while crowdsearch queries have very little chances to be answered during the night and early morning.

Finally, Fig. 13 shows various curves giving the percentage of responded queries (queries with at least one response) for each produced answer, clustering curves by classes of invitations. E.g., the curve corresponding to 2 invitations produces 1 answer in 80% of cases and 2 answers in 20% of cases. Curves show high variability due to the small number of elements for certain ranges of invitations; of course the number of answers increases with the number of invitations. We were surprised to find some runs with a higher number of answers than of invitations; this occurs when questions are posted on Facebook walls, as people can proactively answer them even if they were not invited to, by simply looking at the wall and deciding to comment or like one post/comment.

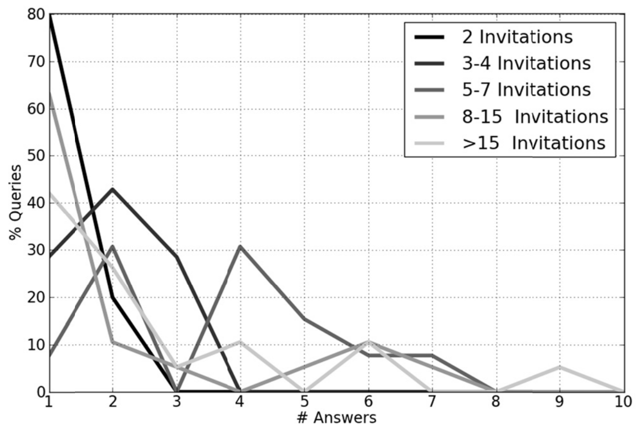


Fig. 13. Percentage of responded queries for each produced answer, classified by number of query invitations.

6. RELATED WORK

Human computation is a computational paradigm where humans interact with computers in order to solve a computational problem, usually too hard to be solved by computers alone. Crowdsourcing is a means and a facilitator for achieving human computation, but the two concepts are not equivalent: crowdsourcing focuses on mechanisms for collecting information from the masses, while human computation focuses on problem solving. Involving human computation into problem solving imposes to address a completely new set of problems related to results quality: bias, spamming, independence of the workers, role of the social network structure.

Several works have studied the impact of the design dimensions (cost per task, number of tasks allowed, and so on) upon various metrics on the results (quality of the outcome, time to response, number of participants, and so on). They all agree that cost per task has little impact on the final quality, while it has on the time needed to obtain it (and participation in general) [15, 16]. Optimization on time can be achieved by studying possible early acceptance/termination algorithms.

Some heuristics can be defined for deciding how many workers are needed for every task [12]. In case of complex tasks, these can be split or associated with new microtasks that aim at validating the complex one [13]. Experiments show that the definition of the process itself can be delegated to the users, with a map-reduce approach, like in the case of Turkomatic [14]. Unfortunately, some practical aspects must be considered too: for instance, it has been shown that workers on Mechanical Turk pick tasks from "most HITs" or "most recent" queues [17]; and HITs in 3rd page and after are not picked by workers.

There are no widespread approaches tackling the problem of applying human computation to exploratory search scenarios [1]. The most common ways of collaborating in information seeking tasks are sending emails back and forth, using instant messaging (e.g Skype) to exchange links and query terms, and using phone calls while looking at a Web browser [18]. The bottom line is that current capacity of users to exploit the actual potential of human computation for exploratory search is still very limited.

The CrowdSearcher approach directly compares with systems, such as CrowdDB [4], Turk [5], and Snoop [6], who transparently combine data and human sources. The main aspects of the comparison were anticipated in Section 2. We recall the main differences: the three systems all integrate with crowdsourcing engines (specifically Mechanical Turk) and not with social networks; they advocate transparent optimization, while we advocate conscious interaction of the query master; they involve users in data completion, while we also involve users in other classical social responses, such as liking, ranking, and tagging.

7. CONCLUSIONS

In this paper we described CrowdSearcher, a paradigm that exploits the power of human suggestions and insights for improving the quality of search results in complex, exploratory information seeking tasks. Exploiting social platforms for crowdsearching provides seamless access to broad sets of responders, at the cost of accepting a constrained interaction paradigm imposed by each platform – which are increasingly enhanced and more open. Enhanced platform programmability from external applications may lead to greater future opportunities for our approach. Of course, the chances to get good responses depend a lot on the consistency of the users' community and on the mechanisms that are exploited for inviting the users and for collecting the responses. We already demonstrated the feasibility of the approach with some prototypes and we quantitatively evaluated the impact of our ideas; we plan to conduct further experiment focused on user studies, and specifically on the adaptation of communities to their new role of responders to CrowdSearcher-enabled queries. The potential breakthrough of CrowdSearcher stands in the ambitious attempt of defining an entirely new, comprehensive formalization of search that involves human sources. The impact of this work will become more evident in the future, when online social interactions, communities, and domain-specific social applications will become even more widespread, combining an always-connected lifestyle with an increasing attitude towards sharing knowledge and participating to online social activities.

8. ACKNOWLEDGEMENTS

This research is partially supported by the *Search Computing* (SeCo) project, funded by *European Research Council*, under the *IDEAS Advanced Grants* program; by the Cubrik Project, an IP funded within the EC 7FP; and by the BPM4People SME Capacities project. We thank all the projects' contributors.

9. REFERENCES

[1] Aula, A. et al. How does search behaviour change as search becomes more difficult? In Proc. 28th international conference on Human factors in computing systems – HCI (Atlanta, GA, USA 2010), 35-44.

[2] Doan, A., Ramakrishnan R., and Halevy, A. Crowdsourcing Systems on the World-Wide Web, Communications of the ACM, April 2011.

[3] Yan, T. and Kumar, V. and Ganesan, D. CrowdSearch: exploiting crowds for accurate real-time image search. Proc. 8th Int. Conference on Mobile Systems, Applications, and Services – MOBISYS (S. Francisco, CA, 2010), 77-90.

[4] Franklin M.J. et al. CrowdDB: answering queries with crowdsourcing. In Proceedings of the 2011 international conference on Management of data (SIGMOD '11). ACM, New York, NY, USA, 61-72.

[5] Marcus, A. et al. Crowdsourced Databases: Query Processing with People. Conference on Innovative Data Systems Research. 2011 (Asilomar, CA, 2011), 211-214

[6] Parameswaran, A. and Polyzotis, N. Answering Queries using Databases, Humans and Algorithms. Conference on Innovative Data Systems Research 2011 (Asilomar, CA, 2011), 160-166.

[7] Baeza-Yates, R. and Raghavan, P. Next Generation Web Search. Search Computing Challenges and Directions, Springer-Verlag, LNCS 5950, 2010, 11-23.

[8] Marchionini, G. Exploratory Search: from Finding to Understanding. Communications of the ACM, 2006. 41-46.

[9] Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P. Liquid Query: Multi-Domain Exploratory Search on the Web. WWW 2010 (Raleigh, USA, 2010). ACM, New York, NY, USA, 161-170.

[10] Bozzon, A. et al. Exploratory Search in Multi-Domain Information Spaces with Liquid Query. Proc. WWW 2011 - Demo (Hyderabad, India, 2011), ACM, New York, NY, USA, 189-192.

[11] Marcus A., Wu E., Karger D., Madden S., and Miller R. Humanpowered Sorts and Joins, PVLDB 5(1), 2011, 13-24.

[12] Kumar, A. and Lease, M. Modeling Annotator Accuracies for Supervised Learning, Proc. Crowdsourcing for Search and Data Mining Workshop – CSDM (Hong-Kong, China, 2011).

[13] Bernstein M.S. et al. Soylent: a word processor with a crowd inside. In Proceedings of the 23nd annual ACM symposium on User interface software and technology. ACM, New York, NY, USA, 313-322.

[14] Kulkarni, A. P., Can, M., and Hartmann, B. Turkomatic: Automatic Recursive Task and Workflow Design for Mechanical Turk. Proc. Extended Abstracts on Human Factors in Computing Systems - CHI EA (Vancouver, CA, 2011), 2053-2058.

[15] Mason, W. A., and Watts, D. J. Financial Incentives and the "Performance of Crowds". KDD Workshop on Human Computation (Paris, France, 2009), 77-85.

[16] Ariely, D. et al. Large Stakes and Big Mistakes, Review of Economic Studies, 76(2), 2009, 451-469.

[17] Chilton et al. Task search in a human computation market. ACM SIGKDD Workshop on Human Computation (HCOMP '10). ACM, New York, NY, USA, 1-9.

[18] Morris, M. R. A survey of Collaborative Web Search practices. Proc. SIGCHI Conference on Human Factors in Computing Systems, (Florence, 2008) 1657–166.