Community-based Crowdsourcing

Marco Brambilla, Stefano Ceri, Andrea Mauri, Riccardo Volonterio
Politecnico di Milano. DEIB Department. Piazza Leonardo da Vinci, 32. 20133 Milano, Italy
{marco.brambilla, stefano.ceri, andrea.mauri, riccardo.volonterio}@polimi.it

ABSTRACT

This paper is focused on community-based crowdsourcing applications, i.e. the ability of spawning crowdsourcing tasks upon multiple communities of performers, thus leveraging the peculiar characteristics and capabilities of the community members. We show that dynamic adaptation of crowdsourcing campaigns to community behaviour is particularly relevant. We demonstrate that this approach can be very effective for obtaining answers from communities, with very different size, precision, delay and cost, by exploiting the social networking relations and the features of the crowdsourcing task. We show the approach at work within the CrowdSearcher platform, which allows configuring and dynamically adapting crowdsourcing campaigns tailored to different communities. We report on an experiment demonstrating the effectiveness of the approach.

1. INTRODUCTION

Crowdsourcing platforms such as Amazon Mechanical Turk are a natural environment for deploying crowd-based applications, since they support the assignment to humans of simple and repeated tasks, such as translation, proofing, content tagging and items classification, by combining human contribution and automatic analysis of results [9]. Crowds take part to social computations either for monetary rewards or for non-monetary motivations, such as public recognition, fun, or genuine will of sharing knowledge.

To get the best possible results, requestors need to dynamically adapt the behaviour of the crowd-based applications. However, in spite of the great importance of crowd adaptation and control, designing and deploying crowdsourcing applications with sophisticated controls is not well covered by existing systems, which lack methods for systematically designing complex adaptation strategies.

The CrowdSearcher system we introduced in [2, 3] brings together a conceptual framework, a specification paradigm and a reactive execution control environment for designing, deploying, and monitoring applications on top of crowd-based systems. We advocate a top-down approach to application design that adopts an abstract model of crowdsourcing activities in terms of elementary task types (such as: labelling, liking, sorting, classifying, grouping) performed upon a data set, and then we define a crowdsourcing task as an arbitrary composition of these task types. Starting from task types, we define strategies for task splitting, replication, and assignment to performers. We also define the data structures which are needed for controlling the planning, execution, and reactive control of crowd-based applications.

The main focus of this paper is leveraging **social communities** for improving the quality and cost of crowd-based information collection. By community we mean a set of people that share common interests (e.g., football club fans, opera amateurs, ...), have some common feature (e.g., leaving in the same country or city, or holding the same degree title) or belong to a common recognized entity (e.g., employee in an office, workgroup or employer; students in a university; professionals in a professional association; ...). Leveraging communities for crowdsourcing includes both the possibility of statically determining the target communities of performers, and also dynamically changing them, taking into account how the community members behave when responding to task assignments. Design-level interoperability is guaranteed by the use of a high-level, platform-independent model. Run-time interoperability is guaranteed by the use of a low-level execution model, such that the tasks can be dynamically created or rerouted in response to monitoring of community performance.

This paper is organized as follows: Section 2 describes related work and our previous work; Section 3 dwells into community-based crowdsourcing and presents our experimental scenario; Section 4 briefly describes the system implementation; Section 5 shows our experimental results; and Section 6 discusses the results and concludes.

2. BACKGROUND AND RELATED WORK

2.1 CrowdSearcher

With our approach, the tasks which constitute a crowdsourcing campaign are described in terms of an abstract model, that was initially presented in [2].

Model. The main strength of the model (represented in Fig. 1) is its extreme simplicity. We assume that each **task** receives as input a list of **objects** (e.g., photos, texts, but also arbitrarily complex objects, with a **schema**) and asks the users to perform one or more **operations** upon them, which belong to a predefined set of abstract **oper-**

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2578835.

Table 1: List of the crowdsourcing operation types.

Task Ty.	Description
Choice	Performer selects up to n items
Like	Performer adds like/unlike annotations to some items
Score	Performer assigns a score (1..n interval) to some items
Tag	Performer annotates some items with tags
Classify	Performer assigns each item to one or more classes
Order	Performer reorders the (top n) items in the input list
Ins./Del.	Performer inserts/deletes up to n items in the list
Modify	Performer changes the values of some items attributes
Group	Performer clusters the items into (at most n) groups

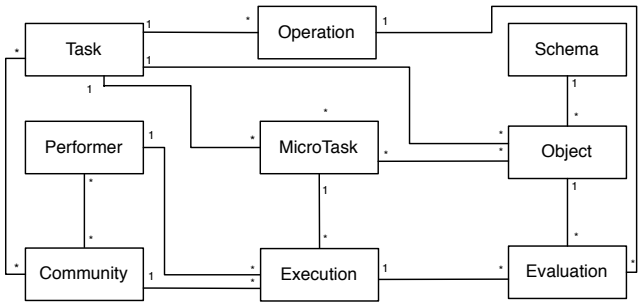

Figure 1: Data model of crowdsourcing application.

ation types. Examples of operation types are *Like*, for assigning a preference to an item; or *Classify*, for assigning each item to one or more classes. The full list of currently supported operation types is reported in Table 1. For instance, a task may consist in choosing one photo out of an input list of photos, writing a caption for it, and then adding some tags. Tasks can be assigned to one or more **communities** of **performers**. For execution purposes, a set of **microtasks** is spawned from each task. Every **execution** is recorded, with the associated performer, community and **evaluation** result.

Control. Our approach provides fine-level, powerful and flexible reactive controls whose properties (e.g., termination) have been proven[3]. We define high-level abstractions for declaring task control, as well as low-level rules for implementing such control, which typically encompasses the evaluation of arbitrary conditions on result objects (e.g., on their level of confidence and of agreement), on performers (e.g., on the number of performed tasks and their correctness, leading to the classification of performers as experts or spammers) and on tasks. Control rules are defined upon **Control Marts**, data structures designed for tracking the execution flow, analogous to data marts used for data warehousing, as its central entity represents the facts, surrounded by control dimensions. The control of objects, performers, tasks, and interoperability is performed by active rules, expressed according to the *event-condition-action* (ECA) paradigm. This paper is concerned with ***Community Control*** as a new aspect of task design.

2.2 Other approaches

Many crowdsourcing startups[1] and systems [4] have been proposed in the last years. Crowd programming approaches rely on imperative programming models to specify the interaction with crowdsourcing services (e.g., see *Turkit* [10], *RABJ* [7], *Jabberwocky* [1]).

As highlighted by [14], several programmatic methods for human computation have been proposed [10][7][1][11][12], but they do not support yet the complexity required by real-world, enterprise–scale applications, especially in terms of controlling the quality of the results. Our approach covers the expressive power exhibited by any of the cited systems, and provides fine grained targeting to desired performer profiles, as well as dynamic and adaptive control over the executions.

Recent works propose approaches for human computation which are based on high level abstractions, sometimes of declarative nature. In [14], authors describe a language that interleaves human-computable functions, standard relational operators and algorithmic computation in a declarative fashion. *Qurk* [11] is a query system for human computation workflows that exploits a relational data model and SQL. *CrowdDB* [5] also adopts a declarative approach by using CrowdSQL (an extension of SQL). *DeCo* [15] allows SQL queries to be executed on a crowd-enriched datasource. *CrowdLang* [12] supports workflow design and execution of tasks involving human and machine activities. Differently from ours, these works do not discuss how to specify the control associated with the execution of human tasks, leaving its management to opaque optimisation strategies.

Datasift [13] is a toolkit for configuring search queries so as to involve crowds in answering them, which allows deciding the number of involved human workers, the query reformulation in steps, the number of items involved at each step and their cost; in this, it is similar to Crowdsearcher. In this paper we go one step beyond and control whole communities.

In designing our system, we have been inspired by several applications of human computation. Among them, [16] compares seven strategies for improving the quality and diversity of worker-generated explanations of social analysis tools; [8] presents alternatives in allocating tasks to workers; and [6] compares some alternatives for involving Mechanical Turk users in terms of their cost and quality.

3. COMMUNITY CONTROL

In our approach, crowd-based applications can assign tasks to performers belonging to different communities. By community we mean a set of people that share common interests, have some common feature or belong to a common recognized entity or social networking group. Communities of performers can be solicited through various media, typically going beyond the classical crowdsourcing platforms, e.g., by addressing social networks, mailing lists, online forums, and so on. Being able to address communities of users is paramount for getting high quality or specialized feedback from the crowd. Results can be further improved by applying **adaptation,** i.e., any change of allocation of the tasks to their performers, based on the performers behavior.

Adaptation can be applied at **task granularity**, when the replanning or reinvitation occurs for the whole task, or at **object granularity**, when the replanning or reinvitation is focused on one (or a few) objects (for instance, objects on which it is harder to achieve an agreement among performers, with a majority-based decision mechanisms).

Adaptation at execution time is either statically or dynamically determined.

[1]E.g., CrowdFlower, Microtask, uTest.

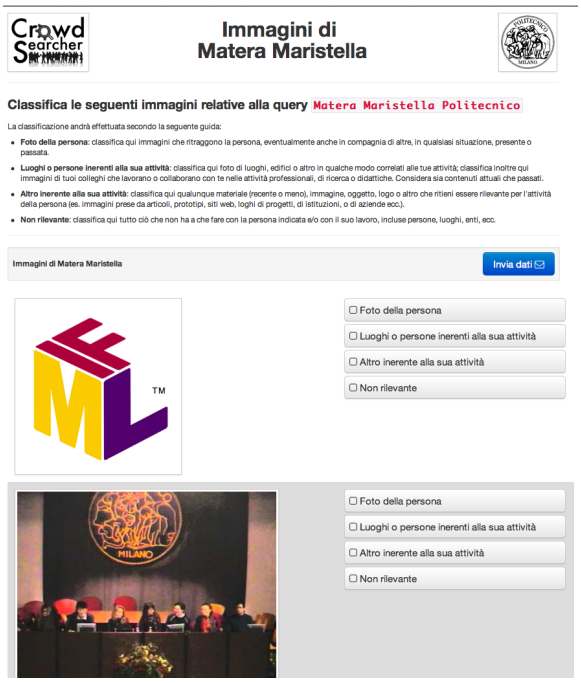

Figure 2: Customized UI for the cross-community scenario (Professors).

- With **Static adaptation**, adaptation is planned, and it occurs at a given time or after receiving a given number of task responses. E.g., an application could migrate from a community to another at a given time of the day, so as to meet lower costs or better performances.

- With **Dynamic adaptation**, adaptation occurs in reaction to specific events that are observed, such as the case of crowds which do not respond as expected.

Dynamic adaptation is quite relevant, as crowd reactions can hardly be anticipated. Thanks to dynamic adaptation, it is also possible to guarantee certain constraints or requirements on application execution:

- **Cost Constraints** can be enforced by limiting the number of tasks which are posted, or by adapting their cost to the allocated budget. This is made possible by the availability of communities (e.g., on social networks) that are willing to participate without a monetary reward.

- **Time Requirements** can be dealt with by adding more processing capability, and possibly by subsequently inviting performers from more and more communities.

- **Diversification Requirements** can be dealt with by involving different performers communities.

In our approach we implement cross-community crowdsourcing through active rules. For ease of presentation, we show them in the context of a concrete scenario.

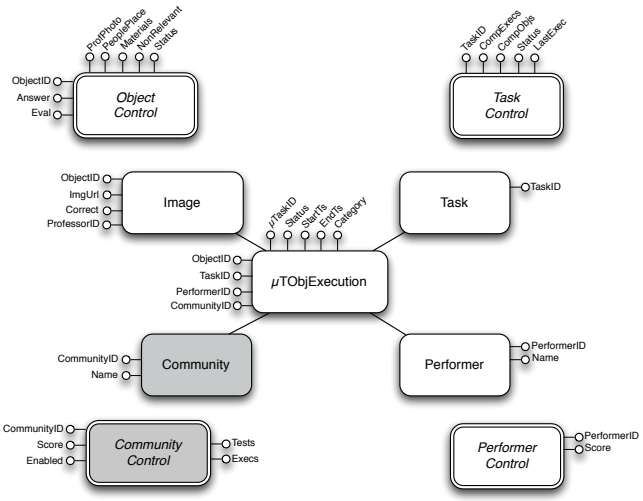

Figure 3: Control mart for the experiment.

3.1 Experimental Scenario

The scenario is concerned with *image classification*. The dataset consists of images about professors of our department retrieved through the Google Image API. In the crowdsourcing campaign we ask the performers to specify whether each image represents the professor himself, some relevant people or places, other related materials (papers, slides, graphs or technical materials), or it is not relevant at all. The experimental settings are as follows:

- **Dataset:** we selected 16 professors within two research groups in our department (DB and AI groups) and we downloaded the top 50 images returned by the Google Image API for each query (the professor's name followed by the keyword "Politecnico"); we excluded the images that were not linked or extremely small in size. We asked the professors themselves to define the ground-truth on the images, through a specific crowdsourcing task (not described here).

- **Crowdsourcing:** each microtask consisted of evaluating 5 images regarding a professor. A customized UI (in Italian) has been developed within Crowdsearcher (as shown in Figure 2).[2] Results are considered accepted (and thus the corresponding object is closed) when some agreement level on the class of the image is reached among performers. Closed objects are removed from new executions.

- **Communities:** we defined 3 types of communities as: the research group of the professor (e.g., Databases); the research area containing the group (e.g., Computer Science); and the whole department (which accounts for more than 600 people in different areas).

3.2 Community Control Implementation

We implement community control through the control mart and the adaptation rules.

[2] The experiment is up and running for evaluation purposes at the following URL (in Italian): `http://is.gd/expprofs`

Rule 1 Invitation of CS-Area community after deadline.

```
e: AFTER 2 days ON TASK[TaskID == 'Professor']
c: ---
a: SET COMMUNITY_ctrl[CommunityID == 'CS-Area'].Enabled
= true,
   reinvite('GoogleImages','CS-Area')
```

Rule 2 Maintenance of last task execution timestamp.

```
e: UPDATE for M_T_O_EXECUTION (EndTS)
c: ---
a: SET TASK_ctrl[TaskID == NEW.TaskID].LastExec = NEW.EndTS
```

Control Mart. Figure 3 depicts the control mart enabling community control. The objects of interest are images returned by Google about professors. The community dimension is represented by the *Community* and *CommunityControl* concepts.

Adaptation Rules. We report here some exemplary adaptation rules that show how the application dynamically adapts by expanding to larger communities, starting from the research group, then expanding to the area, and finally to the whole department (the whole scenario includes additional rules).

- **Rule 1** invites performers from the research area (CS-Area) after two days since the initial invitations, which were sent to a specific research group (DB-group). Note that if the task completes before two days, then the rule does not fire and the task uses just the research group. This rule implements a static adaptation determined at task granularity.
- The next two rules are used to invite the performers of a broader community when the current crowd ceases to produce answers. **Rule 2** saves the timestamp of the last execution of the current task; **Rule 3** invites performers of the broader community (CS-Area) after one hour of *idle* time, i.e. when the last execution occurred more than one hour ago in the smaller community (DB-Group). Rules implement a dynamic interoperability at task granularity.
- **Rule 4** is used for replanning the crowdsourcing task on a specific object when the performers of a community are in disagreement, e.g., if there is are vote on every category of the classify operation. In this example we assume that the invitations were initially sent to CS-Area and then are routed to the DB-Group, representing a group of experts in recognizing images about colleagues of the same group. This rule implements a dynamic interoperability determined at object granularity.

Note that Crowdsearcher offers an environment for fast prototyping of experiments which allows a progressive tuning of execution rules, as we did in the experiments reported in Section 5.

4. IMPLEMENTATION

We implemented cross-commuity rules in CrowdSearcher[3], a platform for crowd management written in JavaScript and running on *Node.js* [4] server; this is a full-fledged event-based system, which fits the need of our rule-based approach. Each control rule is translated into scripts; triggering is modelled through internal platform events. Precedence between rules is implicitly obtained by defining the scripts in the proper order.

CrowdSearcher offers a plug-in environment to transparently interface with social networks and crowdsourcing platforms. A built-in Task Execution Framework (TEF) provides support for the creation of custom task user interfaces, to be deployed as stand-alone application, or embedded within third-party platforms such as Amazon Mechanical Turk or Facebook[2]. Specific modules are devoted to the invitation, identification, and management of performers, thus offering support for a broad range of expert selection paradigms, from pure pull approaches of open marketplaces, to pre-assigned execution to selected performers.

Our platform is cloud-based and is provided with online configuration interfaces where designers can design their crowdsourcing applications through a wizard–like, step by step approach. Figure 4 shows one step of the configuration. A demonstration video of the platform is available.[5]

[3] http://crowdsearcher.search-computing.com
[4] http://nodejs.org
[5] http://www.youtube.com/watch?v=wX8Dvtwyd8s

Rule 3 Invitation of CS-Area community when the DB-Group is idle.

```
e: EVERY 1 minute ON TASK[TaskID == 'GoogleImages']
c: now() - TASK_ctrl[TaskID == 'GoogleImages'].LastExec
> 1 hour
   AND COMMUNITY_ctrl[CommunityID == 'CS-Area'].Enabled=false
a: SET COMMUNITY_ctrl[CommunityID == 'CS-Area'].Enabled=true,
   reinvite('GoogleImages','CS-Area')
```

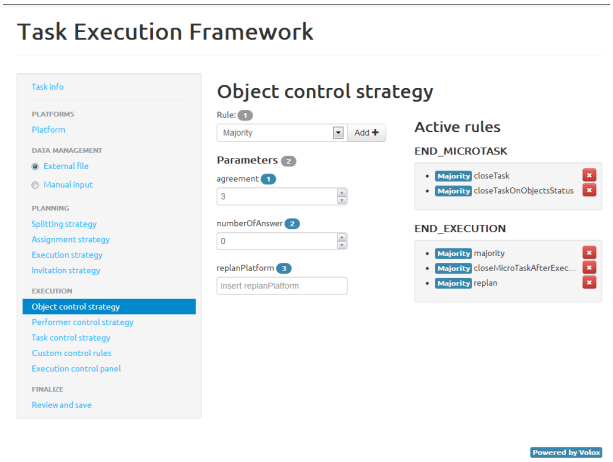

Figure 4: One step of the configuration user interface of CrowdSearcher: definition of the object control strategies, i.e., level of agreement, number of awaited answers, platform where to replan, etc.

Rule 4 Replanning of an object, by invoking an expert community (the DB-Group).

```
e: UPDATE for M_T_O_EXECUTION
c: OBJECT_ctrl[ObjectID == NEW.ObjectID].ProfPhoto >= 1 AND
   OBJECT_ctrl[ObjectID == NEW.ObjectID].PeoplePlace >= 1 AND
   OBJECT_ctrl[ObjectID == NEW.ObjectID].Materials >= 1 AND
   COMMUNITY_ctrl[CommunityID == 'DB-Group'].Enabled = false
a: SET COMMUNITY_ctrl[CommunityID == 'DB-Group'].Enabled = true,
```

5. EXPERIMENTS

We performed several experiments on the running scenario presented in Section 3. The groundtruth and results are available online.[6]

We devised two experiments: in the first one, named *inside-out*, we started with invitations to experts, e.g. people the same groups as the professor (DB and AI), and then expanded invitations to Computer Science, then to the whole Department, and finally to open social networks (Alumni and PhDs communities on Facebook and Linkedin); in the second one, named *outside-in*, we proceeded in the opposite way, starting with the Department members, then restricting to Computer Scientists, and finally to the group's members. Our assumption is that researchers that work closer to the person mentioned in the query know him better and are more able to recognize relevant images.

All invitations (except for the social networks in the first experiment) were sent by email by the system. The communities were not overlapping: every performer received only one invitation. For doing that, the members of the Department, of Computer Science area, and of the DB Group were randomly split into two sets. Invitations have been implemented as a set of dynamic, cross-community interoperability steps, with task granularity and with continuous switchovers starting one working day after a community was idle (stopped to produce results); interoperability control rules very similar to Rules 2 and 3.

Table 2 shows the number of invitations sent out, the number of performers responding, and the average precision of their evaluations. Notice that precision is decreasing when moving towards less expert people, while the social network had good precision as the invitation was posted on groups that know very well the people involved (who were their professors or advisors). For space reasons, in the paper we report only on the second experiment(*outside-in* strategy), but results and datasets are available online for both, (*outside-in* strategy).

Figure 5 shows the number of executions (a) and performers (b) by community. Again, influence of nighttime and weekend on executions is very evident. Figure 6 shows the number of closed objects vs. the number of performed evaluations. Figure 7 (a) shows the precision of evaluations by community and Figure 7(b) shows the final precision on closed objects.

Figure 7(b) compares also the precisions of the *inside-out* and *outside-in* experiments, and shows that former performs better than the latter in terms of quality of results. This is quite evident in the initial phases (when the first half of the objects close), as the performance of experts (research group) is much higher than performance of the people of the rest of department.

6. DISCUSSION AND CONCLUSIONS

This paper proposes an empowered programming and control of crowdsourcing applications across different performer communities. We describe dynamic adaptation and we show how it can be implemented through suitable active rules.

Experimental results show that our method can improve the effectiveness and efficiency of crowd-based applications, by improving quality through dynamic replanning strategies. Experiments let us collect interesting lessons learned regarding interoperability. We noticed that expert performers have a completely different attitude towards the tasks: in a sense, they felt more involved and part of a "mission", they frequently contacted us (about 30% of performers sent us messages) for providing feedback for improving the application, they way questions were asked, or even the dataset. Participants appeared more demanding than generic crowds with respect to the quality both of the application UI and of the evaluated objects. The limited number of participants implied a strong impact of the temporal aspect (responses come in more slowly than in traditional crowdsourcing systems).

[6] `http://crowdsearcher.search-computing.com/multiplatform-and-community`

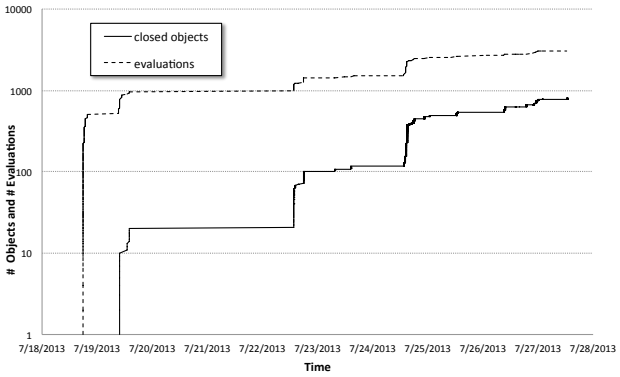

Figure 6: Number of closed objects vs. number of performed evaluations across communities.

Table 2: Cross-Community scenario statistics

Community	#Invites	#Performers	Precision
Research Group	28	13	0,68
Research Area	61	15	0,64
Department	214	34	0,58
Social Networks	N/A	9	0,65
Total	303	71	0,63

Acknowledgments. This work is supported by the Search Computing ERC Advanced Grant funded by the 7th EU FP.

7. REFERENCES

[1] S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar. The jabberwocky programming environment for structured social computing. In *UIST '11*, pages 53–64. ACM, 2011.

[2] A. Bozzon, M. Brambilla, and S. Ceri. Answering search queries with CrowdSearcher. In *21st Int.l Conf. on World Wide Web 2012*, WWW '12, pages 1009–1018. ACM, 2012.

[3] A. Bozzon, M. Brambilla, S. Ceri, and A. Mauri. Reactive crowdsourcing. In *22nd World Wide Web Conf.*, WWW '13, pages 153–164, 2013.

[4] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, Apr. 2011.

[5] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with

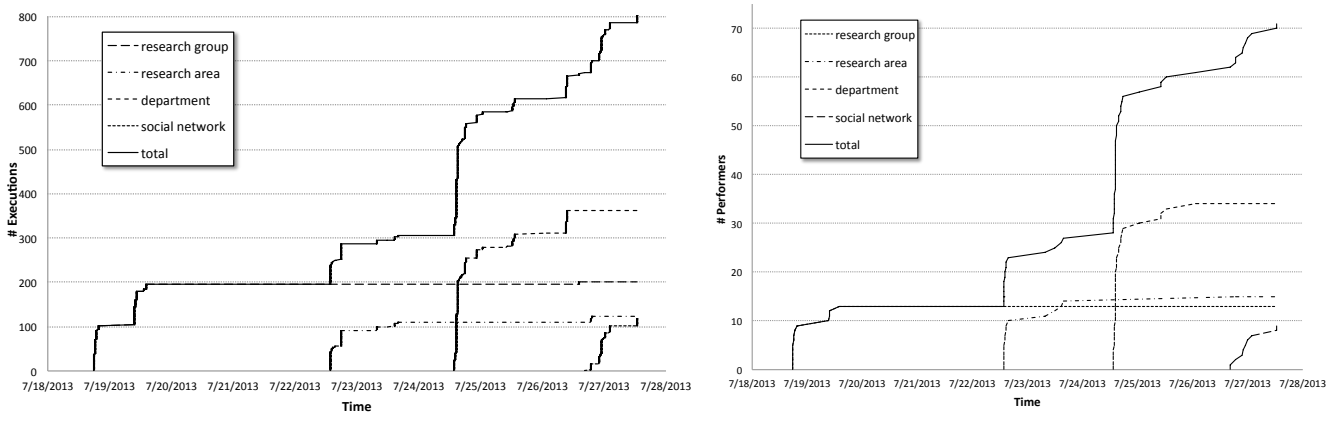

Figure 5: Number of executions (a) and performers (b) by community across the different communities.

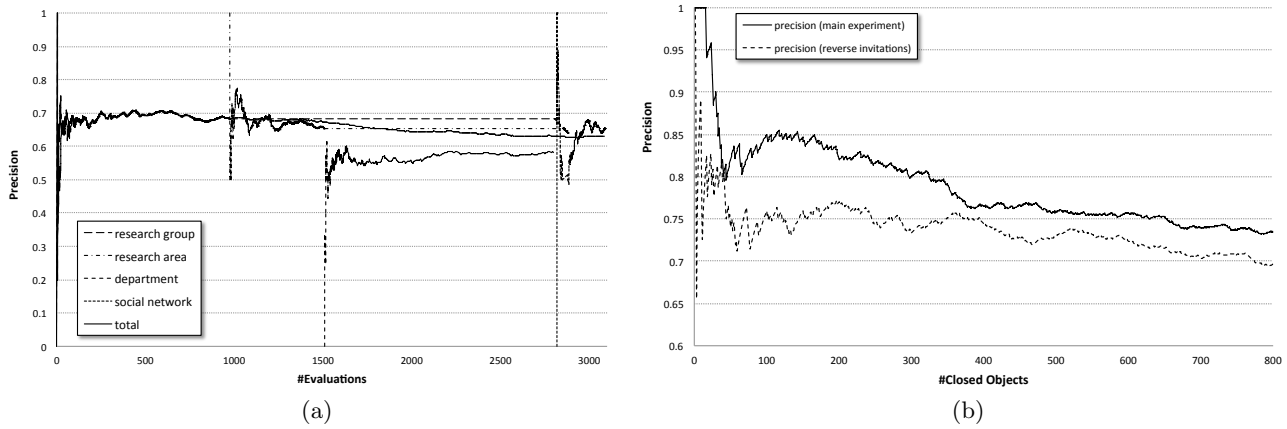

Figure 7: Precision of evaluations by community (a) and comparison of the precision for the inside-out and outside-in approaches (b)

crowdsourcing. In *ACM SIGMOD 2011*, pages 61–72. ACM, 2011.

[6] A. Kittur, E. H. Chi, and B. Suh. Crowdsourcing user studies with Mechanical Turk. In *SIG-CHI Conf. on Human factors in comp. sys.*, pages 453–456. ACM, 2008.

[7] S. Kochhar, S. Mazzocchi, and P. Paritosh. The anatomy of a large-scale human computation engine. In *HCOMP '10*, pages 10–17. ACM, 2010.

[8] M. Kosinski, Y. Bachrach, G. Kasneci, J. Van-Gael, and T. Graepel. Crowd IQ: measuring the intelligence of crowdsourcing platforms. In *Web Science Conf. 2012 (WebSci)*, WebSci '12, pages 151–160. ACM, 2012.

[9] E. Law and L. von Ahn. *Human Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2011.

[10] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. Turkit: tools for iterative tasks on mechanical turk. In *HCOMP '09*, pages 29–30. ACM, 2009.

[11] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR 2011*, pages 211–214. www.cidrdb.org, Jan. 2011.

[12] P. Minder and A. Bernstein. How to translate a book within an hour: towards general purpose programmable human computers with crowdlang. In *WebScience 2012*, pages 209–212, Evanston, IL, USA, June 2012. ACM.

[13] A. Parameswaran, M. H. Teh, H. Garcia-Molina, and J. Widom. Datasift: An expressive and accurate crowd-powered search toolkit. In *1st Conf. on Human Computation and Crowdsourcing (HCOMP)*, 2013.

[14] A. G. Parameswaran and N. Polyzotis. Answering queries using humans, algorithms and databases. In *CIDR 2011*, pages 160–166, Asilomar, CA, USA, January 2011.

[15] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 5(12):1990–1993, 2012.

[16] W. Willett, J. Heer, and M. Agrawala. Strategies for crowdsourcing social data analysis. In *SIG-CHI Conf. on Human Factors in comp. sys.*, pages 227–236. ACM, 2012.