ABSTRACT
Crowdsourcing applications are becoming widespread; they cover very different scenarios, including opinion mining, multimedia data annotation, localised information gathering, marketing campaigns, expert response gathering, and so on. The quality of the outcome of these applications depends on different design parameters and constraints, and it is very hard to judge about their combined effects without doing some experiments; on the other hand, there are no experiences or guidelines that tell how to conduct experiments, and thus these are often conducted in an ad-hoc manner, typically through adjustments of an initial strategy that may converge to a parameter setting which is quite different from the best possible one. In this paper we propose a comparative, explorative approach for designing crowdsourcing tasks. The method consists of defining a representative set of execution strategies, then execute them on a small dataset, then collect quality measures for each candidate strategy, and finally decide the strategy to be used with the complete dataset.

Categories and Subject Descriptors
H.4 [Information Systems Applications]: Miscellaneous

Keywords
crowdsourcing, social computation, empirical method

1. INTRODUCTION
A new class of software applications, called crowd-based applications, is emerging. These applications use crowds, engaged through a variety of platforms, for performing tasks; the most typical application scenarios include fact checking, opinion mining, localized information gathering, marketing campaigns, expert response gathering, image recognition and commenting, multimedia decoding and tagging, and so on.
The common aspect of these applications is the interaction between the requestor (who poses a task), the system (which organizes the computation by mixing conventional and crowd-based modules), and a potentially wide set of performers (who are in charge of performing crowd tasks and are typically unknown to the requestor). The system may take multiple forms: in addition to crowdsourcing platforms (such as Amazon Turk or CrowdFlower) or question-answering systems (such as Quora or Yahoo!Answers), a recent trend is to use social networks (such as Facebook, Twitter or LinkedIn) as sources of human labor.
Crowd-based computations undergo a new set of design principles and phases, as dealing with crowds introduces many concurring objectives and constraints. A non-exhaustive list of such objectives and constraints includes:

- Performers selection, possibly on the basis of their expertise on the task that they should perform;
- Performers rewarding, that includes the monetary aspect (how much to pay for an elementary task) as well as non-monetary ones (such as fun, self-esteem, altruism, visibility and reputation);
- Performers exclusion, the set up of explicit mechanisms for determining low-quality performers and either banning them from the computation or simply disregarding their contributions to the result;
- Convergence criteria, determining on an object basis when the number of results that have been collected is sufficient to take a final decision about the outcome of crowdsourcing for that object;
- Global time constraints, i.e. the time at which the crowd-based computation should be stopped and results should be taken as final;
- Global cost constraints, i.e. the maximum amount that should be spent on the overall crowdsourcing task;
- Object-specific cost constraints, i.e. the maximum amount that should be spent for producing the result relative to a single object;
- Performer-specific cost constraints, i.e. the maximum amount that should be given to each specific performer.

As consequence, a requestor has several design dimensions to consider when building a successful crowdsourcing task; some of these dimensions influence the first and most important decisions, i.e. the choice of the platform to be used...
with Scala, providing a rich variety of options for adaptive

2. BACKGROUND AND RELATED WORK

discuss the results and Section 5 concludes.

approach; Section 4 shows the experimental scenario and
related work; Section 3 describes in details the proposed
parameters settings in the tool.

matically generated, and therefore it is possible to create
crowd-based applications through step-by-step specifications,
no mathematical model can cover the variety of optimization
dimensions and constraints, as each model typically ad-
dresses a small set of decisions for a specific crowdsourcing
ask (e.g., [20], [2], [12], [22], see the next section); even un-
der such limitations, many mathematical models are hardly tractable, as the underlying problems fall into exponential
NP-hard classes.

In this paper, we propose a domain-independent, explo-
rative design method which uses rapid prototyping in the small in order to select the design parameters to be used
for big datasets. The method consists of defining a repre-
sentative set of execution strategies, then execute them on
a small, unbiased dataset, then collect quality measures for
each candidate strategy, and finally decide the strategy to
be used with the complete dataset. In our approach, the de-
signer has to empirically choose execution strategies which are
compared, and then empirically decide the best strategy by
looking at how the solution satisfies the various objectives and
constraints, without a formalization of these empirical
choices, as we believe that any mathematical formulation is not of practical use for this problem. However, we sug-
gest to use diversification criteria in determining the initial
space of execution strategies, so as to guarantee that the
final selection is among alternatives that yield to significant
differences along the considered dimensions.

The proposed method is of general applicability, but it is
complemented by the availability of a tool, called Crowd-
searcher [3,4], that supports the definition of multi-platform
crowd-based applications through step-by-step specifications,
where the application is initially configured and then auto-
matically generated, and therefore it is possible to create
the execution strategies in the small by simply changing the
parameters settings in the tool.

This paper is organized as follows: Section 2 describes
related work; Section 3 describes in details the proposed
approach; Section 4 shows the experimental scenario and
discuss the results and Section 5 concludes.

2. BACKGROUND AND RELATED WORK

Most crowd programming approaches rely on imperative
programming models to specify the interaction with crowd-
sourcing services (e.g., see Turkkit [10], RABJ [14], Jabber-
wocky [11], AutoMan [2] is integrating human computations
with Scala, providing a rich variety of options for adaptive
quality control, although expressed within a low-level pro-
gramming style.

Other works are centered on supporting workflows which
include humans. CrowdLang [19] supports workflow design and
execution of tasks involving human and machine activi-
ties. CrowdWeaver [13] is a system for visually manage
complex crowd work. The work [21] presents a declarative
approach to crowdsourcing workflows based on XML speci-
fications through BPEL4People.

From the databases world studies has been done for both
designing and optimizing crowdsourcing task. CrowdDB [9]
uses an extension of SQL (called CrowdSQL) for both mod-
eLLing the data and querying the crowd and it exploits query
optimization techniques. Qark [17] is a query system for hu-
man computation workflows that exploits a relational data
model and SQL, it provides a series of optimizations like:
batch predicates, run-time pricing, heuristics for reducing the
cost of the join operation, pre-filling the tables with re-
A lot of studies have been done on mode for optimizing
the design of a crowdsourcing task. Usually they focus on
the problem of task allocation and results aggregation, and
they usually use statistical methods [20]. For instance, [2]
proposes a method based on active learning for inferring the
correct solution given a set of answers. The authors in
[12] show how to use the probabilistic matrix factorization
approach in order to aggregate results of a labeling task. In
[22] the authors propose a Bayesian model for aggregating the
results of a labeling task by taking into the account the
quality of the workers. Notice that all these works focus on
a single type of operation (labeling) and address only the
problem of aggregating the results of the single evaluations
given by each performer.

On the other hand in [24] the authors propose a proba-
babilistic framework for choosing which HIT is better to send
to the crowd from a set of candidates based on the infor-
mativeness of the HIT (modeled as entropy). Furthermore
[11] proposes a model for decomposing a task into simpler
sub-task, focusing on the quality of the final results.

Other works use experiments to understand how differ-
ent task design configurations impact on the final results.
Typically they focus on a single dimension like incentives
(both monetary [15,29] or not [7]), task types [8] and task
decomposition [15].

At the best of our knowledge this is the first time that
an empirical method has been integrated in the task design
process. The results of the experiments are promptly used to
decide the configuration of the task.

3. METHOD

Our approach refers to a simple concept model shown in
Figure 1 which describes how each elementary execution of a
crowdsourcing step, called Execution, is referred to an un-
derlying operation (e.g. classifying, tagging, labeling, liking,
commenting) called Task, to a specific Platform (that can be
either a crowdsourcing marketplace or a social network),
to a specific Object of a given collection, and to a specific Performer who executes. This model is a simplified version of
the control mart presented in [4].

These concepts, in turn, are characterized by a set of prop-
erties, whose ranges of values define the design space. Typ-
ically, they should be assigned by the application designer
in order to configure the crowdsourcing tasks, either by in-
Figure 1: Concept model and parameters used by our approach

interacting with a design tool, or by using scripting languages (which in turn invokes an API where they appear as parameters), or for directly configuring the application. For instance, figure 1 illustrates a setting where we define four properties; the method is agnostic to the specific choices of properties but assumes that they can be referred to the concept model and, of course, that they can be used for configuring the execution task. They are:

- **Platform**: where the task will be executed. This is a very important dimension because each platform targets different crowds which have different skills.
- **Cardinality**, i.e. the number of object shown to the performer: this parameter controls the amount of work that a performer has to face each time. It influences the cost and time required by the task.
- **Reward**, i.e. the cost of a HIT on Amazon Mechanical Turk.
- **Agreement**: i.e., with a majority based decision for each objects, it indicates the amount of agreement needed in order to consider an object as evaluated. A high level of agreement should correspond a better quality of the results while negatively impacting on the time and cost.

This list can be extended in order to satisfy specific user needs, for instance adding a spam detection strategy, whose modeling would lead to adding a Spam flag on the performer, set to 0 or 1 to indicate its inclusion or exclusion.

Each candidate execution is thus represented by a vector \( S = \{s_1, s_2, \ldots, s_n\} \) in an \( n \)-dimensional space, where \( n \) is the number of considered parameters; for instance, an execution on Amazon Mechanical Turk showing 3 objects per HIT, requiring a 2 workers over 3 to agree on the evaluation and paying each worker 0.01$ is represented as:

\[
S = ["AMT", 3, 2/3, 0.01]
\]  

Once the design space is well defined, the designer should then choose some of the possible strategies (represented as a collection of vectors.) It is not possible to consider all possible combinations due to the cost and the required time for conducting all the small-scale experiments. It is important to choose few strategies by including interesting points in the solution space and by using as criteria parameter diversification, at the same time by avoiding to include any two solutions when one of them dominates the other. This notion is not easily formalizable, but it takes into account the correlation between parameters. For instance, it makes little sense to include two solutions such as one has a higher cost and a lower object cardinality than another one (i.e., a simpler task which is better paid).

The execution of strategies, both in the small and in the large, can be evaluated by using a set of quality measures that are computed at the end of the process by inspecting how each object has been managed by the crowd (e.g., its classification, tagging, liking and commenting). We use the following quality measures:

- **Cohen’s kappa coefficient**, a statistical measure of inter-annotator agreement for categorical annotation tasks \( \kappa \). When several performers evaluate the same objects, kappa measures the agreement among them.
- **Precision of responses**, that can be computed only when the ground truth is available; it corresponds to the percent of correct responses over the total and can be aggregated at the level of object, performer, platform, or whole task.
- **Execution time**, the elapsed time needed to complete the whole task.
- **Monetary cost**, the total amount of money spent for rewarding the crowd in order to complete the whole task.

This is only a small set of possible performance measures, and can be extended with more complex (as the ones shown in \cite{10} or application specific metrics.

Finally, our approach requires the splitting of the dataset of the objects into two subsets small and large, with \(|\text{small}| << |\text{large}|\), such that the selection of \( S \) is not biased.

Then, all the strategies \( \{S_1, S_2, \ldots, S_m\} \) are run on the small set (in the small phase) and the quality measures are collected; by analysing them, the strategy \( S_{\text{best}} \) which is associated with the best quality measures is selected.

Eventually \( S_{\text{best}} \) is run on the remaining objects of the large dataset and its results are composed with the ones obtained with the small set.

4. EXPERIMENT

We designed an image labeling crowdsourcing task in which we ask the crowd to classify pictures related to actors, telling if it represents the actor himself in a portrait, if it is a scene taken from a movie, or if it is not relevant (exclusive options); We used Amazon Mechanical Turk (AMT) as execution platform.

Using our approach, we identified the following design dimensions: number of images shown in each user task, agreement level for each picture classification, and cost of each AMT hit. Then we selected 8 different strategies (as shown in Figure 2) and we ran them on both the small and large dataset.

The experiment had the purpose of assessing the two main assumptions of our method:
In this paper, we have proposed an explorative approach for designing crowdsourcing tasks. The experimental evaluation shows that the method is applicable to this kind of problem (as there is a good correlation between the results of the experiments in the small and in the large), and that the trade-off between the cost and the added value is affordable.

Future work will focus on formalizing the process for selecting candidate strategies and on the application of the method to a wider set of design dimensions (e.g. varying also the execution platform).

1. The outcome of the experiment in the small is correlated with the result of the experiment in the large.

2. The cost of performing all experiments in the small followed by one experiment in the large is affordable and the extra-effort is well compensated by the possibility of choosing the experiment with the best quality measures in the small.

We determined an experiment of limited size but sufficient to perform such an assessment. We built a dataset consisting of 900 images related to 90 actors, retrieved from Google Images; then we selected 90 images for the phase in the small (i.e. 10 images for 9 actors, including both men and women), so that the comparison of small vs large involves an order of magnitude, which is enough to illustrate the difference between small and large cases. This setting hints to the quality of the method also when the difference between small and large cases is affordable, e.g. using datasets of increasing sizes for computing the quality measures of a restricted number of candidates. The case in the large of Table 1 can be considered an intermediate-size experiment if one has to process a dataset of millions of photos; in such case, the eight cases in the large would result from a selection starting from a larger number of experiments in the small.

One could note that case 7 is associated with a slightly higher cost of 2.70 compared to case 6 (that was selected by considering quality measures in the small), but it also exhibits a better precision in the large of 0.871 compared to case 6; such better precision is not predicted by the experiment in the small and comes as a surprise. Indeed the method incurs some unexpected differences between tests in the small and in the large due to the intrinsic statistical variability of our study; greater sizes in both small and large cases would yield to less variability.

5. CONCLUSION

The experiment was implemented using CrowdSearcher\footnote{\url{http://crowdsearcher.search-computing.org/}}; the set up of each experiment in such case is managed by the tool. CrowdSearcher is an important ingredient for our approach, because it allows to quickly define all the variants needed for the experiment and to easily collect and monitor the performance of the single strategy. Thus, setting up a sequence of experiments with CrowdSearcher requires essentially to change parameters within the tool and regenerate another crowd-based run on AMT which creates suitable HITs sets, with suitable intervals between them so as to build independent observations.

CrowdSearcher also collects statistics about each application, which allows us to read some of the quality measures (such as precision and duration of the experiment) directly from the execution controller; other quality measures, such as Cohen’s kappa coefficient, must be computed by looking at the output objects.

Table 1 summarizes the results of the experiment, by reporting the four quality measures (kappa, precision, cost and duration).

Regarding the first assumption defined in Section 4, we calculated the Pearson correlation coefficient, configuration-wise, between the experiments in the small and in the large. As one can see, correlation is almost one for the cost, that can be obtained just by considering the scale factor between small and large, but correlation is quite good also for duration, performer agreement and precision. Note that durations are longer for the small experiments than for the long ones. This reflects a known behaviour of the crowd, which tends to select tasks with higher number of executions to perform (also due to the bias introduced by crowd platforms, which show the biggest tasks first).

We next compared the strategy by looking for a trade-off between precision and cost. In particular, based upon the small-scale experiment, we selected Strategy 6, which appears to have enough precision (0.864) associated with a low cost (1.92), yielding a good price/performance ratio. The choice of Strategy 6 completes the decision making.

The designer’s choice is anyway driven by cost-benefit analysis, that however is performed in the small, e.g. the designer will be able to decide if a difference in precision from .811 of case 3 to .856 of case 5 is justified by an increase in costs from 1.40 to 4.77.

Note that we spent $22.49 for computing all the strategies in the small and $16.86 for executing the strategy number 6, for a total cost of $39.35; these two numbers are comparable, but the difference between the cost of experiments in the small and in the large increases a lot with big input data. When the task is very large, an incremental tuning is also possible, e.g. using datasets of increasing sizes for computing the quality measures of a restricted number of candidates. The case in the large of Table 1 can be considered an intermediate-size experiment if one has to process a dataset of millions of photos; in such case, the eight cases in the large would result from a selection starting from a larger number of experiments in the small.

1. \url{http://crowdsearcher.search-computing.org/}
Table 1: Quality measures and Pearson correlation of experiments in the small and in the large.

<table>
<thead>
<tr>
<th>Config.</th>
<th>Agreement kappas</th>
<th>Precision</th>
<th>Cost ($)</th>
<th>Duration (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>small</td>
<td>large</td>
<td>small</td>
<td>large</td>
</tr>
<tr>
<td>1</td>
<td>N/A</td>
<td>0.743</td>
<td>0.799</td>
<td>1.35</td>
</tr>
<tr>
<td>2</td>
<td>0.692</td>
<td>0.687</td>
<td>0.855</td>
<td>4.05</td>
</tr>
<tr>
<td>3</td>
<td>0.596</td>
<td>0.575</td>
<td>0.811</td>
<td>1.40</td>
</tr>
<tr>
<td>4</td>
<td>0.473</td>
<td>0.567</td>
<td>0.822</td>
<td>2.22</td>
</tr>
<tr>
<td>5</td>
<td>0.442</td>
<td>0.659</td>
<td>0.836</td>
<td>5.77</td>
</tr>
<tr>
<td>6</td>
<td>0.499</td>
<td>0.540</td>
<td>0.811</td>
<td>1.92</td>
</tr>
<tr>
<td>7</td>
<td>0.580</td>
<td>0.606</td>
<td>0.800</td>
<td>0.72</td>
</tr>
<tr>
<td>8</td>
<td>0.585</td>
<td>0.555</td>
<td>0.844</td>
<td>2.05</td>
</tr>
</tbody>
</table>

Correlation: 0.707 | 0.619 | 0.309 | 0.515

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

REFERENCES


