Human Computation (2015) 2:1:105-131

ISSN: 2330-8001, DOI: 10.15346/hc.v2i1.6

Crowds of Crowds: Performance based Modeling and Optimization over Multiple Crowdsourcing Platforms

SAKYAJIT BHATTACHARYA, XEROX RESEARCH CENTRE INDIA

L. ELISA CELIS, EPFL, SWITZERLAND

DEEPTHI CHANDER, XEROX RESEARCH CENTRE INDIA

KOUSTUV DASGUPTA, XEROX RESEARCH CENTRE INDIA

SARASCHANDRA KARANAM, UTRECHT UNIVERSITY, NETHERLANDS

VAIBHAV RAJAN, XEROX RESEARCH CENTRE INDIA

ABSTRACT

The dynamic nature of crowdsourcing platforms poses interesting problems for users who wish to schedule large batches of tasks on these platforms. Of particular interest is that of scheduling the right number of tasks with the right price at the right time, in order to achieve the best performance with respect to accuracy and completion time. Research results, however, have shown that the performance exhibited by online platforms are both dynamic and largely unpredictable. This is primarily attributed to the fact that, unlike a traditional organizational workforce, a crowd platform is inherently composed of a dynamic set of workers with varying performance characteristics. Thus, any effort to optimize the performance needs to be complemented by a deep understanding and robust techniques to model the behaviour of the underlying platform(s). To this end, the research in this paper studies the above interrelated facets of crowdsourcing in two parts. The first part comprises the aspects of manual and automated statistical modeling of the crowd-workers' performance; the second part deals with optimization via intelligent scheduling over multiple platforms. Detailed experimentation with competing techniques, under varying operating conditions, validate the efficacy of our proposed algorithms while posting tasks either on a single crowd platform or multiple platforms. Our research has led to the development of a platform recommendation tool that is now being used by a large enterprise for performance optimization of voluminous crowd tasks.

1. INTRODUCTION

Crowdsourcing has become a prevalent method for sourcing work with a variety of supporting platforms and business models (Saxton et al., 2013). However, the dynamic nature of crowd poses an interesting problem for requesters who want to schedule a (large) set of tasks on a given platform. Unlike organizational environments where workers have fixed timings, known skill sets and performance indicators which can be monitored and controlled, the available crowd is dynamic, and hence so are the patterns of expertise and quality of work on a given platform. Further, often there is little or no control a platform can exert over their worker population. In such an uncontrolled environment, platforms exhibit varied performance in terms of the observed performance characteristics, e.g., completion time, accuracy, task completion rates etc. However, this performance is not strictly due to noise – underlying patterns can be detected (although, as our data exhibits, the patterns themselves are also dynamic). When considering use cases such as enterprise crowdsourcing where large corporations have a steady stream of work, understanding and exploiting these complexities becomes crucial in order to satisfy requirements and improve quality of work.

This paper studies two interrelated facets; 1) understand the performance dynamics exhibited by crowd platforms, and 2) leverage this understanding to enhance the performance via intelligent scheduling of tasks. For the former, statistical models of dynamic crowd platforms reveal interesting trends and provide a basis for building systems that can predict crowd performance. They additionally serve to simulate platform behavior, which becomes a testbed for our optimization algorithms. For the latter, we show that performance can be enhanced by intelligent scheduling of tasks; we determine posting strategies (varying factors such as batch size, time of posting, price of tasks) on platforms in order to maximize the performance. Naturally, not all platforms behave equally, and we both consider optimizing a schedule on a single platform, and scheduling across many depending on the nature of our requirement. The goals of the paper are the following:

– Understand the performance of crowd platforms and build robust statistical models of crowd performance based on data collected by posting tasks on crowd platforms.
– Design an algorithm for automatically modeling the crowd performance that can be used to generate and update models on any platform with dynamically changing crowd composition that can be used to forecast performance.
– Design and test scheduling algorithms on a variety of such platforms, comparing them via their accuracy and completion time.
– Demonstrate the benefits of adaptive scheduling, in order to maximize performance, on single platforms as well as across multiple platforms.

1.1. Overview and Contributions

In this section we provide an overview of the paper and highlight the specific contributions of our research.

For the purpose of this paper, we consider three different data sets, which we refer to as *platforms* from real experiments on crowd platforms. Note that the task is the same for all three data sets; namely, digitization of fields taken from a healthcare form. For each of the data sets we have a series of measurements of the performance of the crowd at regular (roughly hourly) intervals, where the performance metrics are *accuracy* and *completion time*. Of particular interest is the fact that the

performance varies not just noisily, but rather certain patterns are detectable. This type of regular variation is what our performance models as well as scheduling algorithms attempt to exploit. More details on data collection and methodology can be found in section 2.

Using this data, our aim is to build statistical models for the accuracy and completion time of tasks on each platform. We would like to emphasize that in contrast to most previous models, our models (of the data as well as those intrinsic to our scheduler) are based on only *Externally Observable Characteristics* (EOCs) of the platform, i.e., properties that can be collected/measured from the platform when a task is submitted. Examples of such EOCs include task accuracy, completion time, cost, date and time when the task was completed. An obvious advantage of this approach is that these parameters can be easily observed and recorded for any platform without requiring any details about the internal architecture of the platform.

Previously, EOC-based models were successfully used (Dasgupta et al., 2013) to design a classification based approach for platform recommendation. We extend this work by building longitudinal models of performance and also by designing a system that can automatically build such models. Further, our system in this paper is validated on a larger (four more weeks) dataset.

Section 3 describes our statistical models of performance (accuracy and completion time). We first show that standard time series models (like ARMA) can be used to model performance on a platform adequately. Such models can be used to forecast performance on a platform. They can also be used to select a platform for a given task with the best expected performance – by choosing that platform which has the best performance forecast (on a given day or time). Given the dynamic nature of these platforms it is entirely conceivable that the performance trends visible over a short period (e.g. a few weeks) may change significantly with time. To avoid repeated modeling exercises and to enable scalable modeling of multiple platforms, we build a system that automatically tunes model parameters with the arrival of new data and changes in crowd performance. The use of such automated modeling systems can be envisaged in applications where a large number of platforms are analyzed repeatedly (e.g. in platform recommendation systems) which would require multiple dynamically changing performance models. We empirically demonstrate that our system adaptively models platform performance over several weeks. Thus our statistical models are robust to variations in working situations like periodicity e.g. varying performance over different hours of a day, or between weekdays and weekends.

These statistical models provide the foundation for a simulation testbed used to evaluate various algorithms as a "proxy" crowd for experimentation. This is particularly useful since going to the crowd directly may come at a significant cost. An application of such a testbed is readily seen when we evaluate our scheduling algorithms.

The forecasts (and recommendations) provided by the longitudinal models are adequate when a small batch of tasks are to be posted on a crowd platform. However in many enterprise settings, a large batch of tasks are provided to one or more crowd platforms for which responses may be received over a span of days or weeks. Given the temporal variations observed in crowd performance, it is then possible to design adaptive scheduling algorithms that can post the tasks in a manner that gives us much better performance (in terms of accuracy and/or completion time) than when the entire batch is posted at one–shot.

These scheduling algorithms are studied in section 4. In particular, we focus on measurable per-

formance metrics such as accuracy and completion time when we vary the size of a batch (i.e., the number of tasks posted simultaneously on a given platform). Naturally, this kind of analysis could be extended to other independent variables (eg. amount paid for each task), and we expect similar techniques to hold.

First, we consider different kinds of schedulers across different platforms. In this process we establish that adaptive scheduling algorithms generally outperforms other algorithms, including a naive one which always posts a batch of the same size on a given platform. Hence, we take this algorithm for comparison in our next set of experiments where our goal is to see if we can have better performance by scheduling across several different platforms. We use Thomson Sampling based algorithms on each platform individually, and then on the three at once, and indeed show using more platforms is preferable.

To summarize, our contributions in this paper are:

– We show that temporal patterns in crowd performance (accuracy and completion time) can be modeled using statistical time–series models.
– We design an automated system for modeling crowd platforms that can adapt to changing performance trends.
– Using these statistical models, we design a simulation testbed that can serve as proxy for real crowd platforms and be used to evaluate algorithms.
– We design and evaluate adaptive scheduling algorithms that can be used to schedule large batches of tasks over one or more crowd platforms. These algorithms leverage dynamic temporal trends observed in the platforms to optimize performance obtained from the crowd.

2. DATA

We choose *digitization* as a task category for our data collection exercises. In particular, we consider an insurance form of a popular healthcare provider. A corpus of approximately 10000 hand-filled forms was used to create crowd tasks[1]. For the reported experiments, a crowd task (HIT) required six specific fields to be extracted from the forms (circled in Fig. 1). If all six fields are not extracted, the task is deemed incomplete and the worker is not paid for it. Upon completion of the task, the worker is paid 1 cent per task. We observe that number of tasks completed ranges from 2 to 12 tasks per minute; the *effective* hourly wages earned ranges from $1.44 (at mean completion time of 25 seconds per task per worker) to $7.2 (at mean completion time of 5 seconds per task per worker). Note that these are simple tasks that consume very less time and a worker could choose to undertake multiple tasks – both from our set of tasks (i.e. extract data from multiple forms) or from other tasks posted on the platforms.

Data is obtained from two crowd platforms that are free marketplaces. For privacy reasons, we do not disclose the actual names of the platforms. Workers in these platforms are a heterogeneous mix and consist typically of non-experts. Previous works analyzing worker characteristics can be found in (Kulkarni et al., 2012; Ipeirotis, 2010; Difallah et al., 2015). While knowledge about the workers will definitely help us in better understanding as well as modeling these dynamic systems, in crowdsourcing platforms like Amazon Mechanical Turk (AMT), CrowdFlower, MobileWorks

[1]Note that volunteers have hand-filled the forms with fictitious data.

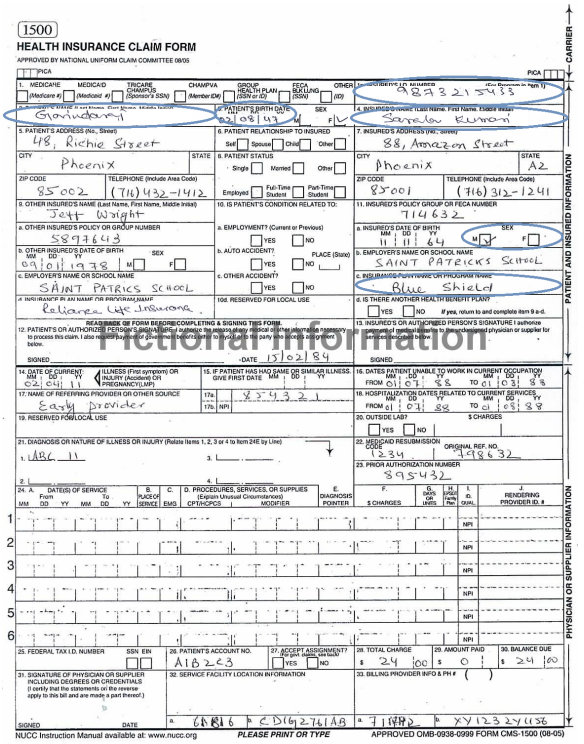


Figure 1. ***Health Insurance Form.***

and others, very little is revealed about the workers. The premise of our models and algorithms is that "externally observable characteristics" of crowd platforms like task accuracy and completion time can be modeled and used in scheduling algorithms effectively *without* internal knowledge of the platforms, like worker characteristics, which are hard to obtain. Analysis of worker behaviour is outside the scope of our present work but will definitely be a very useful to understand and analyze our models and algorithms in the future.

We conduct two data collection exercises. In the first exercise, batches of 50 forms, randomly selected from the corpus are posted to the crowd, every hour of the day, for all days of the week for one week. Further, each batch is simultaneously posted on two crowd platforms: Platform 1 and Platform 2. The data obtained from this exercise is used to build a preliminary model that characterizes the performance of the platforms. In the second phase, we use a similar experimental design, however for a longer period of 4 weeks and across three platforms: Platform 1, Platform 2 and Platform 3.

We record the hour of day (Hour 0 to Hour 23), day of week (Monday to Sunday) and the week (1 to 5) in which the task was posted. We measure task completion time from the time of the task was posted to the time the result was received. We also measure task accuracy by comparing the crowd response with actual values on the form. Accuracy is defined as 1-L/N where L is the Edit Distance (minimum number of string operations required to transform a string to another) and N is the length

of string.

Fig. 2 shows the mean task accuracy and mean completion time obtained each hour on two of the platforms.

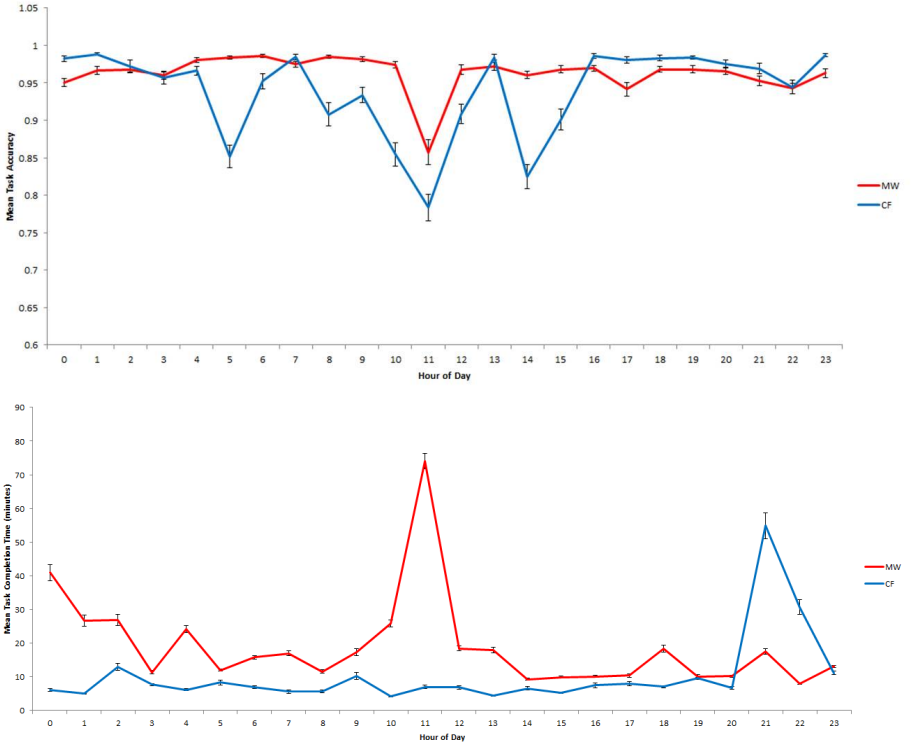


Figure 2. *Task accuracy (above) and completion time (below) on two platforms averaged every hour*

3. STATISTICAL MODELS OF CROWD PERFORMANCE

The temporal patterns seen in Fig. 2 *within* each platform are consistent across the four weeks – shown by the low variance at each hour – and the patterns are remarkably distinct between the two platforms. For example it clearly seen that the completion time in the 11^{th} hour on platform 2 is much higher than at other times. Similarly the completion time in the 21^{st} hour on platform 1 is much higher than at other times. The reasons behind such temporal variation could be many: workers with similar performance characteristics may be regularly logging in at the same time each day or the response time may be much slower around lunch/dinner. Without additional information about the workers and platforms, we cannot ascertain the reasons behind these regular patterns. However we can leverage the periodicity of the patterns to build statistical models of performance. Depending on the data available, the granularity of the model can be in hours, days or even weeks.

This forms the motivation for using longitudinal models of performance. A longitudinal model has

an underlying time series where some observations (for our case, accuracy and mean completion time) are repeated for the same variables (for our case, the workers) over long periods of time (for our case, over five weeks). To model the time series, we consider the Auto-Regressive Moving Average (ARMA) structure. An ARMA (p,q) model has two components, p auto-regressive terms and q moving average terms. Notationally, for a time series data x_t $(t = 1,2,...)$ the ARMA (p,q) model can be written as

$$x_t = A + \sum_{i=1}^{p} \lambda_i x_{t-i} + \sum_{i=1}^{q} \gamma_i \varepsilon_{t-i}$$

where λ_i-s are the auto-regressive parameters, γ_i-s are the moving average parameters, ε_t-s are the white noise components and A is a constant. For more details, please see (Hannan and Deistler, 1988).

In section 3.1 we show that ARMA models fit the data very well thereby demonstrating that crowd performance can be modeled in this manner. Such models can be used to forecast performance on a given platform. They can also be a useful tool in platform selection: given performance models of multiple platforms, we can forecast the performance of each of the platforms and select the platform with the best forecast. The dynamic nature of these platforms do pose a problem over longer time periods: since the worker composition or platform characteristics may change over time, there is reason to believe that the performance trends we observe may also change. So, a model that captures the temporal trends over a few weeks may no longer be valid after a few months or whenever there are internal changes in the platform. Hence, the model must be adapted to the changing platform characteristics. To create such adaptive models, we automate the model-building process. Section 3.2 describes such an automated model building process. We also empirically demonstrate that this process automatically adapts to the changing performance characteristics of the platforms.

Finally, in section 3.4 we use these models to build a simulation testbed that can be used to test algorithms in a controlled environment.

3.1. Model with manual intervention

We start with the first data set. We view task-completion time as longitudinal data, starting at hour 0000 on Monday, since 50 measurements (one per task) are taken each hour. Allowing for heterogeneity we fit a different ARMA model to each hour and choose the best model using the Bayesian Information criterion. The fit is sufficiently good: the pseudo R^2 value (Long, 1997) is 0.975.

As the tasks we sent are simple, the accuracy of the responses are high: around 80% of the responses have 100% accuracy leading to no apparent time-series structure. If Y is the accuracy value, then $1-Y$ becomes a sparse response, where most of the observations are zeroes. We perform a sparse regression on the data $1-Y$ with day, hour and number of tasks as the predictor variables, pseudo R^2 for which, comes out to be 0.998.

3.2. Automated modeling

To reduce manual intervention as much as possible, we build an automated model building system that can explore the correlation structure within the data, find out the best structure by information

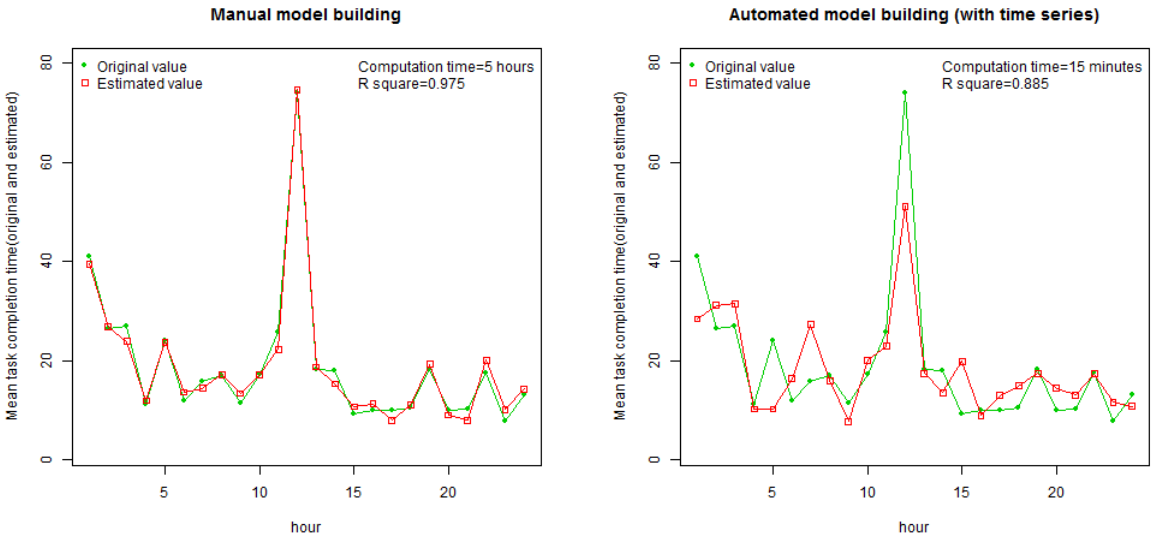


Figure 3. ***Observed and estimated mean task completion times computed from statistical models built by (left) manual intervention and (right) automated system.***

theoretic criteria and build the model by cross-validation so as to avoid overfitting. To do that, the system searches for the random factors and then constructs a longitudinal model for each factor. The final model is a convex combination of all the models corresponding to the respective grouping factors. Fig. 6 shows the algorithmic steps of the automated system to build a longitudinal model. The system finds the grouping factors and then splits the data in training and test sets in an automated way. For each split the system finds the best ARMA model, and the models are linearly combined, where the coefficient of combinations are also discovered by the system. The best set of models and the best set of linear coefficients are found using the Akaike Information Criteria (AIC) (Akaike, 1974) and the lowest mean squared error. On the dataset we use, the best model found is a first order autocorrelation structure for both time and accuracy. Notationally, the model is

$$Y_t = \alpha + \beta X + \varepsilon_t$$

where Y_t is the performance (time or accuracy) at time t, X is the set of independent variables, α and β are regression coefficients and ε_t is the error part such that $\varepsilon_t = \phi\varepsilon_{t-1} + \gamma$ where γ is the white noise and ϕ is the AR(1) parameter.

The adjusted R^2 for the above model 0.88 for time and 0.79 for accuracy. The time taken to compute the best model is around 15 minutes.

The plots in Fig. 3 show the original and estimated mean task completion times per hour for these two methods, i.e. manual and automated. Clearly when the model is created manually the fit is better (0.975) but it requires 5 hours of dedicated time by a data scientist. To repeat this model building process at regular intervals (e.g. every week) and scale it to multiple platforms (up to

hundreds) it is more realistic to use an automated process. The automated system is a reasonable compromise between goodness-of-fit of model (0.88) and the time it takes to build it (15 minutes) both of which are acceptable.

3.3. **Automated modeling on four weeks of data**

We use the automated method to validate the system on the second set of four weeks' data. Two issues arise in the context of automated modeling:

– As more data is available, can the system auto-tune its parameters? What impact will it have on goodness-of-fit of the model?
– It is likely that observations from the distant past have actually no effect on the predictive capability of the model. Can the system decide automatically whether to keep all data, or to remove some of it? Also, if it decides to remove, can the system automatically decide how much to remove?

To address the first issue, we divide the data into four parts, each corresponding to one week. Starting with the first week, the dataset is incremented one week at a time. At every step, the system is run, a new model is built, whose goodness-of-fit (closeness between estimated and original observations) is compared with the previous model. Fig. 4 shows the mean task completion time per hour, both estimated and original. From the plots, we can see that goodness-of-fit of the model, measured by adjusted R^2, increases and thus the model learns better with more data.

To address the second issue, we start with all the observations of four weeks, and delete one week at a time starting from the farthest week (week1). At every step, just like we did earlier, the system is run, a new model is built, whose goodness-of-fit (closeness between estimated and original observations) is compared with the previous model. As with the manual model-building case, the goodness-of-fit is measured by the adjusted R^2 value (Long, 1997). Fig. 5 shows the mean completion time per hour, both estimated and original. From the plots, we can see that the goodness-of-fit of the model is highest with all four weeks of data (0.81) and lowest with only the latest week (0.13). The model seems to lose goodness-of-fit as more and more past data is removed. In other words, the model learns from historical data. By comparing the goodness of fit of various models from the past, the system can ascertain how much data to discard. For example, if the model built from past 5 weeks' data has a lower goodness of fit than a model built from past 4 weeks' data, then 4 weeks can be taken as the threshold and all data before the threshold can be discarded.

Fig. 6 shows the algorithm of the automated model building system. The system imputes the missing values, finds the grouping factors and then for each grouping factor splits the data in training and test sets in an automated way. For our present experiment we set the system to take a random 75% of the total days as training and the rest 25% as test data. For each split the system finds the best autoregressive moving average (ARMA) model, and linearly combines the models corresponding to the grouping factors. The best set of models and the best set of linear coefficients are found using the best Akaike Information Criteria (AIC) (Akaike, 1974) and the lowest mean squared error, as shown in Fig. 6.

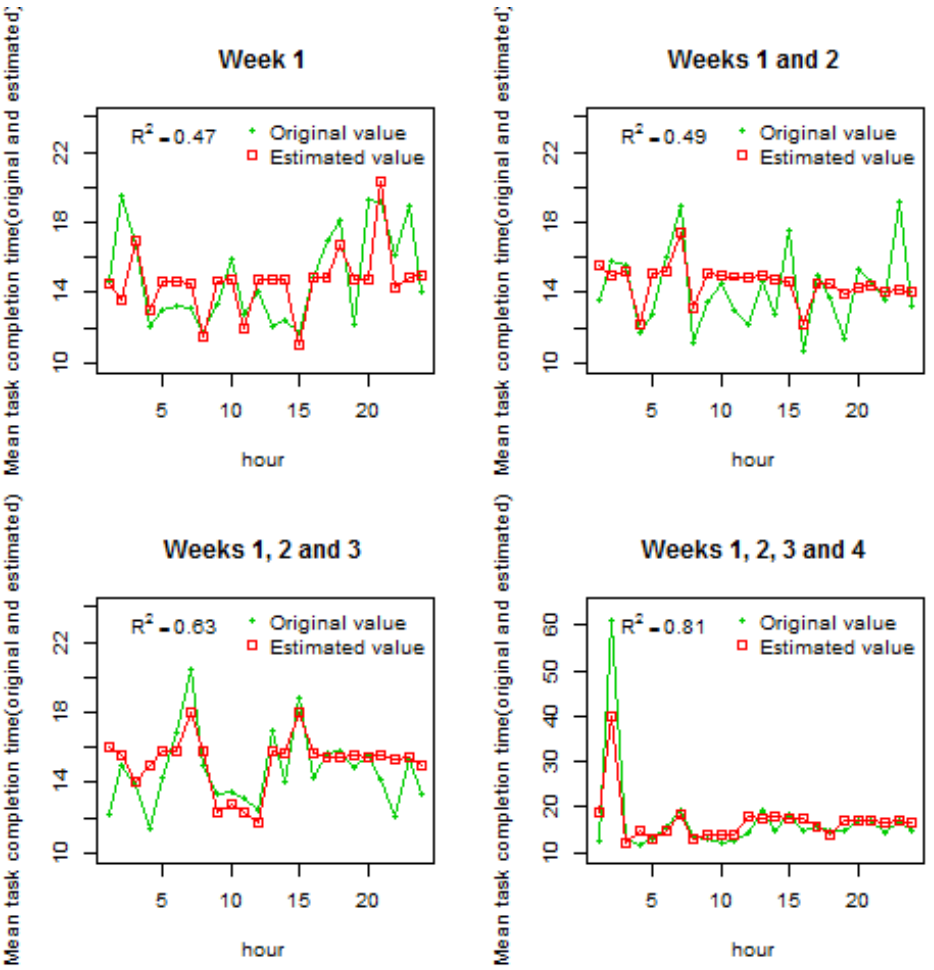


Figure 4. ***Original and estimated mean task completion time per hour starting with week 1 and adding more observations each week. We divide the data into four parts, each corresponding to one week. Starting with the first week, the dataset is incremented one week at a time. At every step, the automated model building system is run, a new model is built, and its goodness-of-fit, measured by adjusted R^2, is calculated. We see that goodness-of-fit of the model is the highest (0.81) when the model learns from four weeks of data and the lowest (0.47) when the model learns from only the first week.***

3.4. Simulation testbed

In order to evaluate algorithms, we create a testbed simulating the performance of a crowd. Using these simulations we can generate performance data for several weeks (from real data of five weeks). The simulations allow us to study the behavior of the algorithms under carefully controlled experimental conditions. These simulations are generated by combining additional models over the

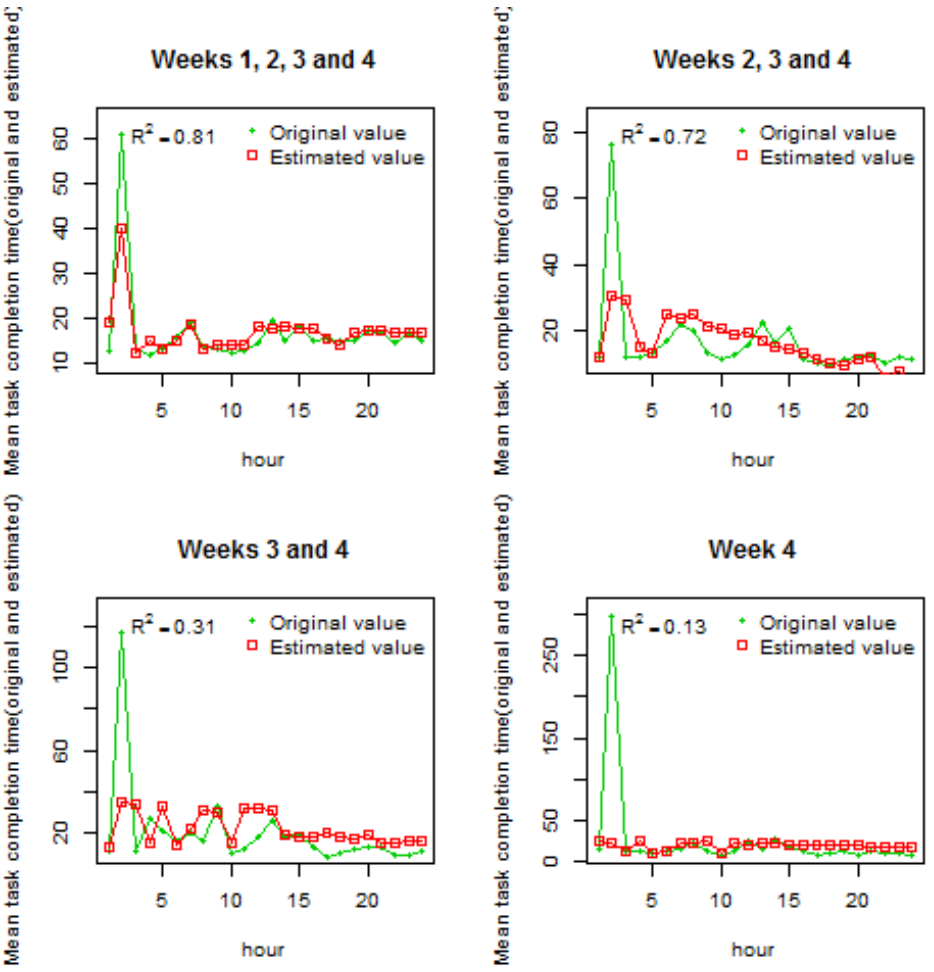


Figure 5. ***Original and estimated mean task completion time per hour starting with four weeks data and then removing one week's data at a time. We start with all the observations of four weeks, and delete one week at a time starting from the farthest week (week 1). At every step, after deleting a week, the system is run, a new model is built, and its goodness-of-fit (measured by adjusted R^2) is calculated. We see that the goodness-of-fit of the model is highest with all four weeks of data (0.81) and lowest with only the latest week (0.13). The model seems to lose goodness-of-fit as more and more past data is removed. In other words, our automated system learns better with more historical data.***

model described in the previous section.

Assuming correlations within each hour (as modeled in the previous section) and correlations between hours and between days, we use a simulation model generated by three (independent) time series structures:

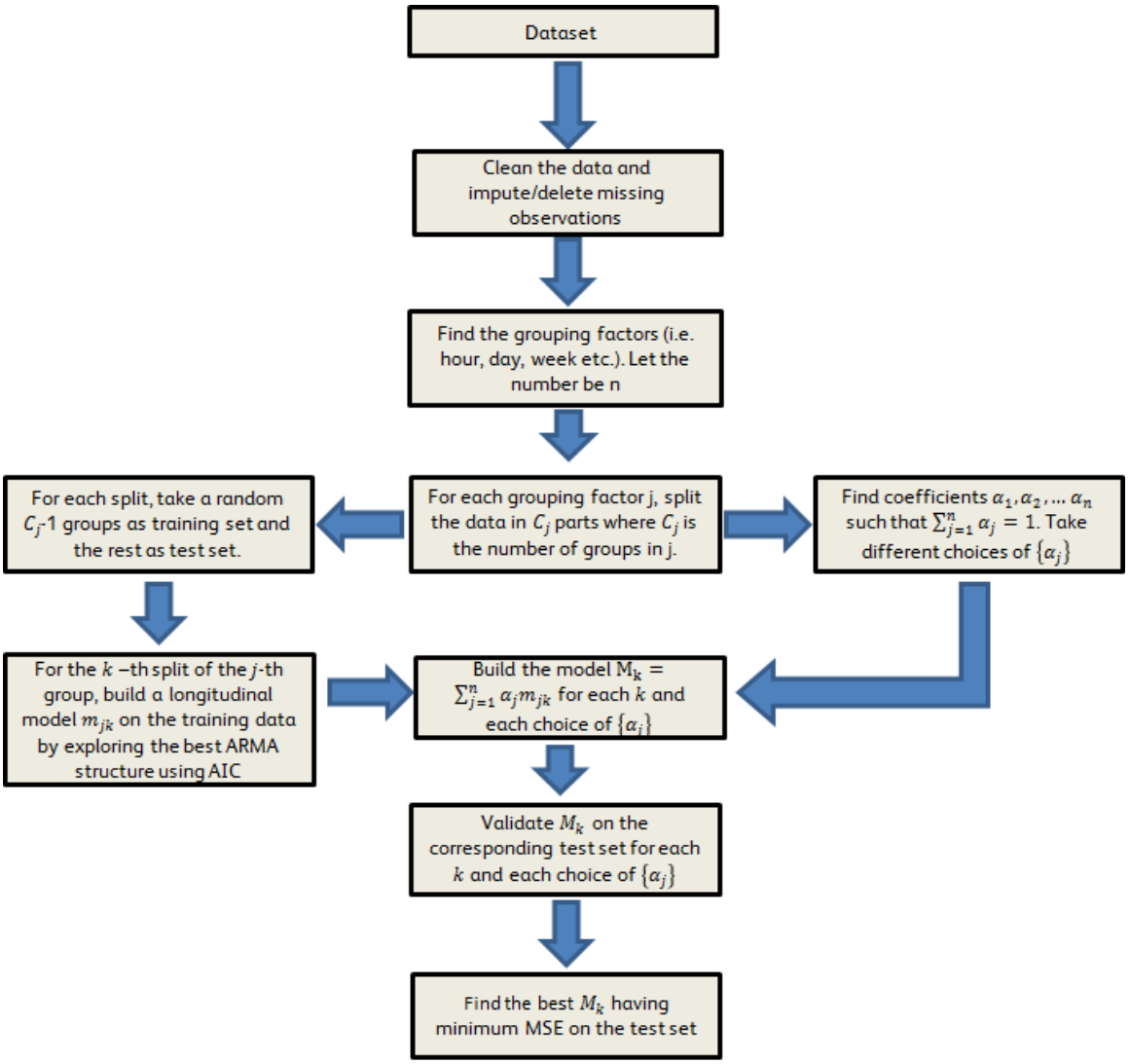


Figure 6. Automated modeling algorithm

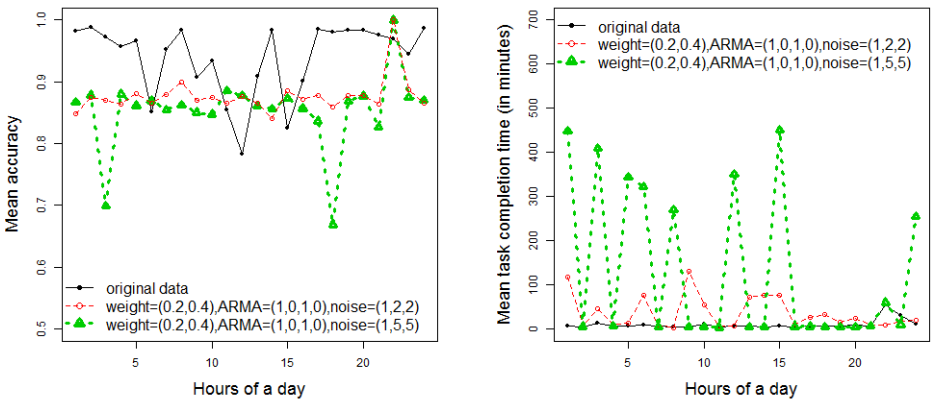
- T_1 : For each hour, $Y_t = \phi_1 (Y_{t-1}, Y_{t-2}, \ldots, Y_1)$ for $t \in [1, 50]$.
- $T_2 : Y_{i\times 50} = \phi_2 \left(Y_{(i-1)\times 50}, Y_{(i-2)\times 50}, Y_{50}\right)$ for $i \in [1 : 168]$.
- $T_3 : Y_{i\times 50} = \phi_3 \left(Y_{(i\times 50-1200}, Y_{i\times 50-2400}, \ldots, Y_{50}\right)$ for $i \in [1 : 168]$.

Y_t denotes performance (completion time/accuracy) at time t. In the simulations, we consider the weighted linear combination of these three structures. The weights can be varied to get different types of simulations. Within each hour (i.e., for T_1), the performance values are fitted using an

ARMA(p,q) structure where the choices of p and q are obtained using the AIC/BIC criterion (see (Schwartz, 1978) and (Akaike, 1974) for details about information theoretic criteria). For T_2 and T_3, however, we use a generalized ARMA structure for each time series where the coefficients can be varied to get different simulations. The model is assumed to be of the form $\alpha T_1 + \beta T_2 + (1 - \alpha - \beta)T_3$ where $0 \leq \alpha, \beta \leq 1$ and $\alpha + \beta \leq 1$ where α and β are the weights. We also use noise parameters to control the variance within each model. A list of the parameters and the values used in the simulations is shown in table 1.

Parameter	Values
Weights of T_1 and T_2 (α, β)	(0.33,0.33) (0.2,0.4) (0.4,0.2)
ARMA structure of T_2 (p_1, q_1)	(1,0), (1,1)
ARMA structure of T_3 (p_2, q_2)	(1,0), (1,1)
Noise parameters $(\sigma_1, \sigma_2, \sigma_3)$	(1,2,2) (1,3,3) (1,4,4) (1,5,5)

Table 1. ***Parameters of the simulation model, and the values used to generate simulations***



(a) (b)

Figure 7. ***Plot of (a) average completion time and (b) accuracy over 24 hours of a day: simulations and original data. When noise is small (1,2,2), the model has small deviation from the original one (red line). When noise is large (1,5,5), the model has significant deviation (green line).***

The simulation parameters are chosen to capture a large variety of real-life scenarios. For example, if hourly variation and daily variation in workers' performance are both important, we give equal

weights to T_1, T_2 and T_3, i.e. $\alpha = \beta = 0.33$. For other choices of α and β, we assign more importance to any two of the three time series models and less importance to the other one, i.e. $\alpha = 0.2$ or $\beta = 0.2$. The auto-regressive and moving average factors of the ARMA structure are varied between (1,0) and (1,1). A structure (1,0) indicates an AR(1) model. A structure of (1,1) is a more complicated model where the time series has an inherent smoothing factor with gap 1. These two structures together capture a wide variety of situations. We do not use more complicated structures like ($p > 1, q > 1$) because such models are rarely used in reality. However, the model can easily incorporate generalized structures. The choices of σ_2 and σ_3 are similarly chosen, in order to capture all kinds of deviation from the original model, from small to large. E.g. $\sigma_2 = \sigma_3 = 2$ indicates a negligible deviation and $\sigma_2 = \sigma_3 = 5$ indicates a significant deviation.

For each of the 48 models, we simulate four weeks' data. Note that for the real data (in the previous section), we had modeled accuracy using sparse regression. However, here we use a common time–series model for both accuracy and completion time in the simulations since the fit, for accuracy, using this model (T_1) on the real data is also very good: the pseudo R^2 value is 0.913.

In order to get a sense of how the simulated data varies from the original data, we show two sample simulations along with the original data for the first week in Fig. 7. The weight parameters of the simulations are $(\alpha = 0.2, \beta = 0.4, (1 - \alpha - \beta) = 0.4)$, thus being the 'farthest' from the original data since time series T_2 and T_3 have higher weights than the original model. Also note that the variance changes dramatically when the noise parameters are increased from 2 to 5.

The simulation testbed allows us to test crowd performance based algorithms using a small amount of data from a real crowd platform. Since crowd performance varies dynamically with the composition of the platform, no real world experiment in a limited time period can capture all the variability present in the long run. By varying different parameters of the simulation, we can extensively test the performance of algorithms before they can be deployed. While the underlying models allow us to test our algorithms on crowd performance as seen in the past, by varying parameters of the simulations, we can extensively test the algorithms on cases when there are significant departures from the model. In the following sections we use the simulation testbed to test the performance of scheduling algorithms on crowd platforms.

4. SCHEDULING TASKS ON CROWD PLATFORMS

Our scheduler uses a multi–armed bandit framework to schedule a (large) batch of tasks on a platform by simultaneously learning the behavior of the platform and optimizing the performance obtained through the crowd. The framework can be extended to schedule tasks over multiple platforms and thereby direct tasks to the best performing platform at any given time. In the following sections, we review our scheduling algorithm and extend it to schedule tasks over multiple platforms. Experiments on our simulation testbed follow.

The multi–armed bandit framework (Berry and Fristedt, 1985) has been used in many real–world problems to model online decision making scenarios that require simultaneous exploration (acquiring knowledge) and exploitation (optimizing). In this framework each arm represents a different option resulting in a random reward specified by an unknown reward function. Bandit algorithms seek to simultaneously learn the reward functions and optimize the received rewards. Multi armed bandit frameworks have also been considered in the context of crowdsourcing (Tran-Thanh et al.,

2012; Celis et al., 2013). However, the focus has been on learning optimal parameters of either individual workers or platforms as opposed to our more complex problem of dynamically scheduling batches of tasks.

We can classify the Externally Observable Charachteristics (EOCs) into three categories depending on their role: Response, Adjustable and Limit parameters. These three sets of parameters are not mutually exclusive. *Response parameters* are parameters that the user wants to optimize. These include accuracy, completion time and cost. Note that the values of these parameters depend, directly or indirectly, on the task, the time and the composition of the crowd. They may change from one worker to another and may also be different for the same worker at different times. *Adjustable parameters* are those that the scheduler is allowed to vary within given limits. Examples include batch size, cost of each task, and number of judgements. *Limit parameters* are those that the scheduler is not allowed to vary; these set upper or lower limits on various parameters for the scheduler. For example, the maximum cost or completion time of the entire batch of tasks can serve as limit parameters.

The scheduler divides the input batch of tasks into subsets of tasks and sends the subsets to the crowd by varying only the adjustable parameters. The execution of the entire batch of tasks must be completed within the limits set by the limit parameters. It iteratively estimates the values of the response parameters from the responses obtained from the crowd and schedules the tasks in a way that optimizes the response parameters for the input batch of tasks. We send the selected subset of tasks, in each iteration to one or more platforms, depending on whether scheduling is done on one or multiple platforms. A schematic of the scheduler is shown in Fig. 8. To schedule over multiple platforms, tasks are sent to multiple platforms and the parameter set now contains information from all the platforms.

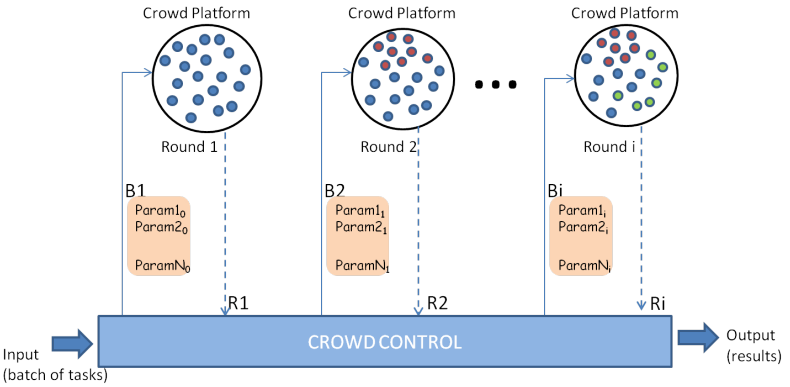


Figure 8. Schematic of our Scheduler.

In this study we restrict ourselves to the following parameters.

– Two response parameters, tested independently:[2]

[2]The algorithms are described assuming a maximization setting; the changes for minimization are obvious.

  - mean completion time, and
  - mean–to–variance ratio of accuracy.
- One adjustable parameter: batch size.
- One limit parameter: Total completion time.[3]

Thus in each of the algorithms we assume an input t (timeout) that is the time limit for each round. Such a time limit is also practically useful. Previous studies (Wang et al., 2011; Faridani et al., 2011) have shown that completion times have a heavy tailed distribution: a majority of the jobs get completed fast whereas the remaining (the tail) take inordinately long time. Depending on the type of task, the batch size and the platform used, the timeout parameter can be set. Although fixed in the following algorithms, we can also vary the timeout (with the batch size) for each round. The algorithms can be generalized to include multiple adjustable, limit and response parameters.

4.1. Scheduling Algorithms

We describe four scheduling algorithms, two based on multi-armed bandit algorithms, and two other simpler algorithms provided for baseline comparisons. The general template of our algorithm, that schedules tasks over multiple platforms is shown in algorithm 1.

Algorithm 1 Scheduler over Multiple Platforms

Input: Tasks T, Platforms $P_1, P_2, \ldots, P_n$, Objective function f, Historical Data H
Initialize: Completed Tasks, $C = \phi$; Incomplete tasks, $I = T$; Timeout t
repeat
  Select b tasks from I: $R \subseteq I, |R| = b$
  Distribute R tasks over n platforms: $R_1, R_2, \ldots, R_n$
  Send R_i to platform P_i
  At completion or after t time steps: withdraw incomplete tasks, let R_c be the set of completed tasks
  Update: $C = C \cup R_c$, $I = I - R_c$,
until $C = T$

The main difference between the algorithms lies in the way the adjustable parameters (b_i) are selected in each round. The algorithms typically use an objective function (such as mean completion time or mean–to–variance ratio of accuracy) that is to be optimized for the entire batch. Based on the response received (R_c) in each round, the value of this function ($f(R_c)$) is computed and the value of the adjustable parameter (b) is determined for the next round. We now describe these details for the algorithms we evaluate. Note that in the case of scheduling over a single platform, $n = 1$, and a single batch size is selected in each round.

Optimistic Scheduler (OS)

This simplest possible scheduler, which forms a baseline for our comparison with other algorithms, simply sends a pre–decided fixed number of tasks in every round. Note that previous studies (Wang

[3]This is used to set the time limit for each round of the scheduler.

et al., 2011; Faridani et al., 2011) suggest that this strategy itself will significantly improve crowd performance compared to the case when all tasks are sent to the crowd at one-shot, thus providing a competitive standard for the remaining algorithms.

GP-UCB based scheduler (GPS)

This scheduler is based on a Bayesian Optimization framework which we briefly describe, following the presentation in (Srinivas et al., 2010). The goal of Bayesian Optimization is to sequentially optimize an unknown function $f : D \mapsto \mathbf{R}$. In each round i, we observe the function value (with noise): $y_i = f(x_i) + \varepsilon$ at a chosen point $x_i \in D$. A Bayesian Optimization solution attempts to sample the best possible x_i from the domain D at each step with the aim of optimizing $\Sigma_{i=1}^{T} f(x_i)$ (over T rounds). For our scheduler, we model the response parameters, R, as a function of the adjustable parameters, A. At each round, the scheduler samples from the space of adjustable parameters in order to optimize the response parameters for the entire batch.

The key ingredients of a Bayesian Optimization technique are the modeling assumptions about the unknown function f and the sampling rule used to sample from the domain. In the GP–UCB algorithm which we use, f is modeled as a Gaussian Process. In a Bayesian setting, the posterior from a GP prior is again a GP distribution and can be computed using simple analytic formulas (see (Srinivas et al., 2010) for details). In our experiments we obtain the first prior by training the GP on historical data. The UCB sampling rule is as follows. Let x be the vector (of values for the adjustable parameters) that is chosen in each round of the algorithm. We choose x_i in round i such that $x_i = arg\ max_{x \in D}\, \mu_{i-1}(x) + \sqrt{\beta_i}\sigma_{i-1}(x)$, where μ_{i-1} and σ_{i-1} are the mean and covariance functions of the GP, respectively, at the end of round $i-1$ and β_i is a constant. Intuitively, the method samples from known regions of the GP that have high mean (function values closer to the known maxima) and also from unknown regions of high variance, thus simultaneously optimizing and learning. As recommended in (Srinivas et al., 2010), for finite domain D, we set $\beta_i = 2\log|D|i^2\pi^2/6\delta$. In our experiments we set $\delta = 0.1$ and use a squared–exponential kernel in the GP.

Algorithm GP–UCB based Scheduler (GPS)

- Select batch sizes from the UCB rule.
- Perform Bayesian update on μ, σ using $f(R_c)$ in each round

Thompson Sampling based Scheduler (TS)

A simple and efficient bandit algorithm is Thompson sampling (Chapelle and Li, 2011) which is based on probability matching. The key idea of the method is to randomly draw each arm according to its probability of being optimal. The reward for each arm is assumed to come from a parametric distribution and after each round, as more knowledge is gained, the distributions are updated in a Bayesian manner.

Our scheduler fits into this framework where the arms are chosen from the domain of the adjustable parameters (the batch sizes) and the rewards are obtained through the chosen objective function determined by the crowd response. We model the problem as a standard K–armed Bernoulli bandit with arms consisting of n-tuples of batchsizes, the mean reward of the ith arm modeled as a Beta

distribution, the conjugate of the Binomial distribution. Each time the crowd response is better than the previously seen objective function value, it is counted as a *success* for the arm. If the crowd response is worse than the previously seen value, it is counted as a *failure*. We initialize the Beta distribution priors to $\alpha = 1, \beta = 1$ and maintain success and failure counters S_i, F_i for each arm.

Algorithm Thompson Sampling Scheduler (TS)

• Batch size selection:
for $i = 1, \ldots, k$ **do**
  Draw θ_i from Beta$(S_i + \alpha, F_i + \beta)$
end for
Select batch size (arm) $(b_1, b_2, \ldots, b_n) = arg\ max_i\ \theta_i$
• Update distribution parameters:
if $f(R_c) > f_{prev}$ **then**
  $S_i = S_i + 1$
else if $f(R_c) < f_{prev}$ **then**
  $F_i = F_i + 1$
end if

Gradient-based Scheduler (GS)

The principle of this algorithm is simple: it starts with the minimum batch size and the minimum value of the objective function. Each time the crowd response is better than the function value in the previous round, it doubles the batch sizes. If the crowd response is worse than the previously seen value, it halves the batch size. In the pseudocode below, we denote the n-tuple (vector) of batchsizes by $\mathbf{b}$ and all vector operations are performed coordinate-wise.

Algorithm Gradient based Scheduler (GS)

• Batch size Selection:
Initialize: $\mathbf{b} = \mathbf{b}_{min}$
In each round:
if $f(R_c) > f_{prev}$ **then**
  $\mathbf{b} = min(2\mathbf{b}, \mathbf{b}_{max})$
else if $f(R_c) < f_{prev}$ **then**
  $\mathbf{b} = max(\mathbf{b}/2, \mathbf{b}_{min})$
end if

5. EXPERIMENTAL RESULTS

We describe our experimental results in this section. The first set of experiments are conducted to compare the performance of our algorithms for scheduling over a single platform. These experiments use the single–platform version of the scheduling algorithms, as presented in (Rajan et al., 2013), but on a much larger dataset of four weeks. In the next set of experiments we schedule tasks

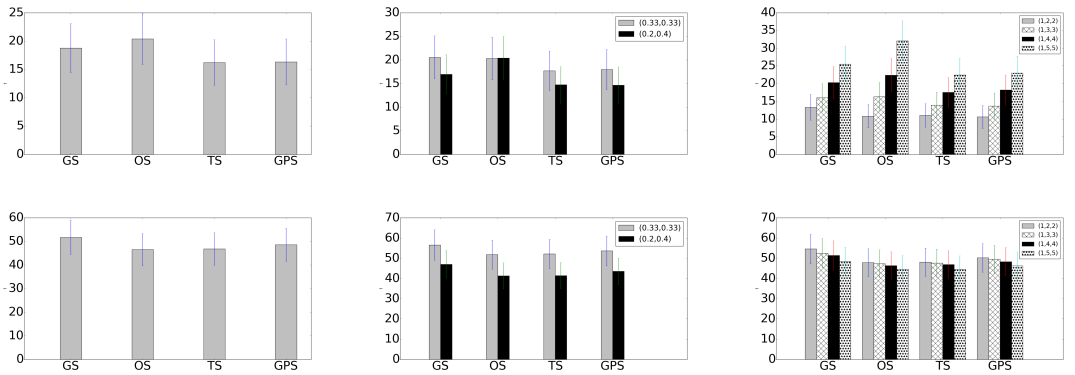


Figure 9. ***Average Performance of Algorithms: mean completion time (top row), mean–to–variance ratio of accuracy (bottom row). Left column: averages over all simulations; Middle column: two bars, one for each weight setting, averaged over all other simulation parameters; Right column: four bars, one for each noise setting, averaged over all other simulation parameters.***

over multiple platforms, using our algorithms described in the previous section and observe a significant improvement in performance. All experiments are conducted using the simulation testbed described in section 2.2.

5.1. Scheduling on a single platform

We test the algorithms using the following parameter settings:
$b_{max} = 25, b_{min} = 5, A = \{5, 10, 15, 20, 25\}, k = |A| = 5$. The (real) data collected from the crowd is used as historical data and various simulations are used as *future* data for which the algorithms create schedules for a batch of 20000 tasks. Given the day of the week, the time of the day and a batch size, the simulator provides the batch completion time and mean accuracy of the batch.

We test two objective functions independently: the average completion time and the mean–to–variance ratio of the accuracy. Note that we want to minimize the average completion time and maximize the accuracy. Note that the accuracy values in the data (and hence simulations) are already high, so we chose the the mean–to–variance ratio, which incorporates an aspect of reliability, as the objective function instead of simply the mean accuracy. The performance of an algorithm is by the value of the objective function for the entire batch of input tasks. These values, shown in Fig. 9 for different simulation settings, are averages (with standard deviation bars) over 1680 runs: 10 runs of each scheduler executed for 24×7 starting timepoints in a week.

The first column of Fig. 9 (top row: completion time, bottom row: mean-to-variance ratio of accuracy) shows the performance of the algorithms averaged over all the simulation parameters. Adaptive algorithms, TS and GPS performs the best with the least mean completion time and the highest mean–to–variance ratio for accuracy. The improvement in performance, with respect to other algorithms, is higher for completion time where the variance in the data (and simulations) is higher.

The middle column of Fig. 9 shows the performance of the algorithms separately for each of the weights in the simulations. For each algorithm there are two bars, one for each weight setting, which represents the algorithm's performance averaged over all other simulation parameters. GPS performs the best, with TS a close second in the case of accuracy and equal in the case of completion time. Note that the performance is better in the case where the weights of the second time–series structures are higher.

The last column of Fig. 9 shows the performance of the algorithms separately for each of the four noise parameters in the simulations. For a single algorithm there are four bars, one for each noise level, which represents the algorithm's performance averaged over all other simulation parameters (weights). While the differences in the performance are not much when there is less noise in the data (the first bar, parameter $(1,2,2)$), the differences become significant with increase in noise (the last bar, parameter $(1,5,5)$), once again showing that with more noise, GPS and TS adapt faster and creates better schedules.

Overall, both the adaptive algorithms perform consistently well, GPS better than TS when mean–to–variance ratio of accuracy is optimized and GPS comparable to TS when mean completion time is optimized. The performance of GPS on the present dataset of four weeks is much better than its performance on one week's data shown in (Rajan et al., 2013). This shows that using past data is beneficial in improving scheduling performance, which may be further improved by fine tuning the parameter values of the GP. However, the simplicity and efficiency of TS makes it the most attractive choice for scheduling.

5.2. Scheduling across multiple platforms

In this section we compare the performance of the scheduling algorithms in two scenarios: when scheduling is done on a single platform for each of the three selected platforms versus when scheduling is done over all three platforms simultaneously using our algorithm. We present the results only with respect to completion time as the performance metric. Results for accuracy display similar trends and are not presented. Our results show that by scheduling over multiple platforms (as opposed to scheduling over a single platform), we can further increase the overall performance and achieve better accuracy and completion time.

We consider the TS algorithm across multiple platforms, and show marked improvement as displayed in Fig. 10. We measure the performance across different platform simulations with varying weights and noise settings as in our first experiment. Note that in this case, our comparison is still the average time a *single* task takes to complete, hence we show true performance gain, not simply improvement due to parallelization.

The performance of TS over multiple platforms (MULT), averaged over all simulations, is found to be significantly better than over any of the single platforms: P1 (p–value: 0.0004), P2 (p–value:0.0139) or P3 (p–value: 0.0451). A two–sample one–sided t–test is conducted to determine whether the mean is significantly lower at 95% confidence[4].

[4]For two samples with sizes n_1 and n_2, means μ_1 and μ_2, and variances σ_1^2 and σ_2^2, the null hypothesis is $H_0 : M_1 = M_2$ and the alternate hypothesis is $H_1 : M_1 < M_2$ (M_1 and M_2 are population means). The test statistic is $T = (\mu_1 - \mu_2)/\sqrt{(\sigma_1^2/n_1 + \sigma_2^2/n_2)}$, and the p–value is the probability $P = 1 - P(t(k) > |T|)$ where $t(K)$ is a t-distribution with degrees of freedom $k = (\sigma_1^2/n_1 + \sigma_2^2/n_2)^2/[(n_1 - 1)\sigma_1^2/n_1 + (n_2 - 1)\sigma_2^2/n_2]$. If $P > .05$, we accept the null hypothesis (i.e. there is no significant difference between the means), else we

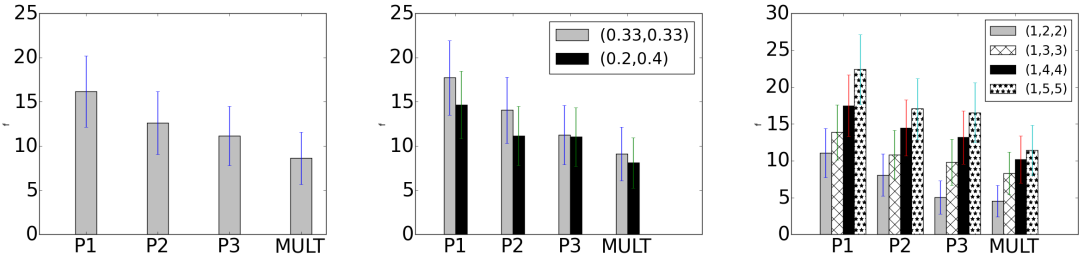


Figure 10. ***Performance (mean completion time) of the TS algorithm on each of the three simulated platforms, P1, P2, P3, individually, and across all three (MULT). Left column: averages over all simulations; Middle column: two bars, one for each weight setting, averaged over all other simulation parameters; Right column: four bars, one for each noise setting, averaged over all other simulation parameters.***

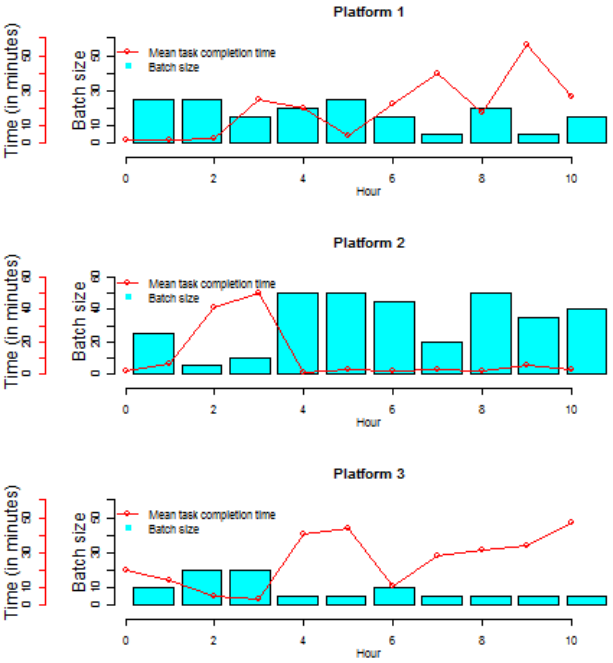


Figure 11. ***Performance of the TS algorithm across three platforms.***

A visual depiction of how algorithm TS works is presented in Fig. 11: the algorithm learns the performance of different platforms, and exploits their variability, placing more tasks on the best

reject the null hypothesis (i.e. there is a significant difference between the means).

performing platform at that given time. For example, one can observe that at hour 2, the majority of tasks are placed on Platform 2, which gives the best performance at that time. In a similar but different vein, few tasks are places on Platform 2 outside of hours 2 and 4 since performance on this platform is significantly worse at those times. Note that this is all done in an automated fashion, and the model is constantly being updated with the new observed parameters, hence an enterprise can simply use such a scheduler without requiring any tuning or interference. Moreover, we expect an improvement in the algorithm's performance to only increase as the number of available platforms increases.

Similar experiments are conducted with algorithms GPS, GS and OS on single and multiple platforms. Figures 12, 13 and 14 compare the performance of these algorithms on the three individual platforms and when all three platforms are used simultaneously. For the GPS algorithm, performance over multiple platforms (MULT), averaged over all simulations, is significantly better than that over platform P1 (p–value: 0.0042) but not over P2 (p–value: 0.1129) and P3 (p–value: 0.2564). For the GS algorithm, performance over multiple platforms (MULT), averaged over all simulations, is significantly better than that over platforms P1 (p–value: 0.0013) and P2 (p–value: 0.0075) but not over P3 (p–value: 0.0513). For the OS algorithm, performance over multiple platforms (MULT), averaged over all simulations, is significantly better than that over all three platforms: P1 (p–value: 0.0017), P2 (p–value: 0.03) and P3 (p–value: 0.04). In most cases, scheduling over multiple platforms has better performance than scheduling over any single platforms but it also depends on the scheduling algorithm being used, with TS giving the best performance.

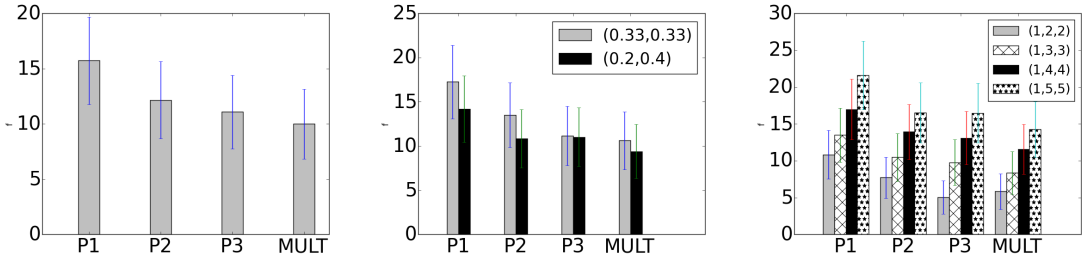


Figure 12. ***Performance (mean completion time) of the GPS algorithm on each of the three simulated platforms, P1, P2, P3, individually, and across all three (MULT). Left column: averages over all simulations; Middle column: two bars, one for each weight setting, averaged over all other simulation parameters; Right column: four bars, one for each noise setting, averaged over all other simulation parameters.***

Fig. 15 repeats the average performance (over all simulations) for all the four algorithms, shown in the previous four figures, for the case when scheduling is done over multiple platforms. We observe that the performance of GPS is similar to that of TS whereas OS has the worst performance. GS is better than OS but is not as good as TS or GPS. TS is significantly better than both GS (p–value: 0.0278) and OS (p–value: 0.0001) that are not bandit–based adaptive algorithms.

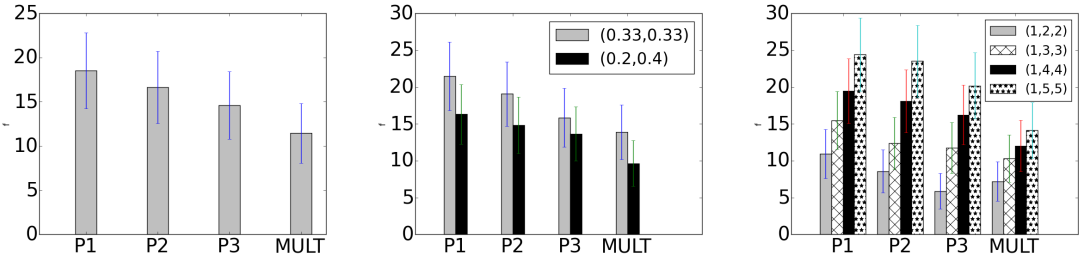


Figure 13. ***Performance (mean completion time) of the GS algorithm on each of the three simulated platforms, P1, P2, P3, individually, and across all three (MULT). Left column: averages over all simulations; Middle column: two bars, one for each weight setting, averaged over all other simulation parameters; Right column: four bars, one for each noise setting, averaged over all other simulation parameters.***

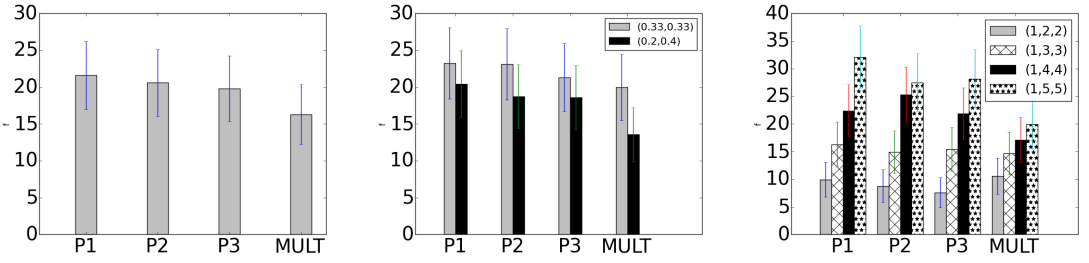


Figure 14. ***Performance (mean completion time) of the OS algorithm on each of the three simulated platforms, P1, P2, P3, individually, and across all three (MULT). Left column: averages over all simulations; Middle column: two bars, one for each weight setting, averaged over all other simulation parameters; Right column: four bars, one for each noise setting, averaged over all other simulation parameters.***

6. RELATED WORK

In this section we describe related work and comment on how the present study differs from them.

Many researchers have focused on modeling platforms using, e.g., multi-agent systems (Bücheler et al., 2011) or probabilistic (Hossfeld et al., 2011) and adaptive modeling (Roy et al., 2013) or the effect of task design on performance (Eickhoff and Vries, 2013). However, these models require knowledge of platform internals, such as worker skill or task complexity to make the characterization of platform performance. In (Wang et al., 2011), the completion time of tasks on the AMT platform is modeled using a survival analysis model. (Dasgupta et al., 2013) envisions a characterization of crowd platforms based on Externally Observable Characteristics (EOCs). None of these works has generically modelled and tested the dynamics of crowdsourcing platforms.

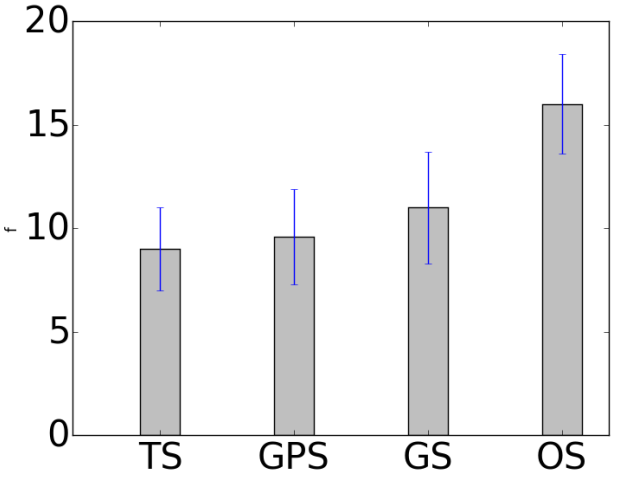


Figure 15. ***Performance (mean completion time) of the algorithms TS, GPS, GS and OS when scheduling over three platforms, averaged over all simulation conditions.***

The problem of task assignment to meet requester constraints has been well-studied in the crowdsourcing literature using a wide variety of techniques, including pushing tasks to specific workers based on performance and interest (Difallah et al., 2013), allocating via a combinatorial algorithm (Minder et al., 2012), using retainer models from queuing theory (Bernstein et al., 2012), using data analytics on historical worker performance (Liu et al., 2012), or greedy scheduling taking the best available workers (Khazankin et al., 2012, 2011). While (Little et al., 2010) demonstrates a programming model that facilitates the incorporation of human computation algorithms in Mechanical Turk,(Dai et al., 2010) presents a decision-theoretic planner that dynamically optimizes iterative workflows to achieve the best quality/cost tradeoff in crowdsourcing scenarios. Quality assurance is the focus of (Jung, 2014), which predicts unobserved workers' performance on a new task based on matrix factorization models. A survival analysis model is used in (Faridani et al., 2011) to design a pricing policy which is a trade-off between completion time and price. In (Heidari and Kearns, 2013) considers efficient algorithms and structures that minimize both the maximum workload of any worker, and the number of workers. (Bernstein et al., 2011) proposes Adrenaline, a crowd-powered camera application, which uses a retainer model for mobilizing synchronous crowds and rapid refinement for early identification of crowd agreement. However, all these works pertaining to scheduling strategies, or crowdsourcing workflows, rely heavily on prior knowledge of worker skills or preferences, and often depend on push models which lets the requester select the desired worker. In our work, we simply use the EOCs of platforms to schedule tasks.

The AUTOMAN system (Barowy et al., 2012) is a fully automatic crowd-programming system that facilitates requesters by allowing them to specify the quality and budget; while the system automatically yet transparently manages all details necessary for scheduling, pricing, and quality control. Unlike in the scheduling mechanism proposed in our work, (Barowy et al., 2012) uses a simple doubling strategy for budgeting and the time allotted for the task. AUTOMAN does not incorpo-

rate optimality in its solution and focuses on the programmability aspects of human computation. Quality control is ensured in AUTOMAN by iteratively computing the minimum number of additional responses required to meet the confidence levels posed by the requester; unlike in (Rajan et al., 2013) where accuracy is incorporated in the cost function of the scheduling algorithm. (Rajan et al., 2013) considers the problem of optimally scheduling batches of tasks on a *single* platform. Neither of these works consider *multi-platform* solutions or *automatic model building and selection of platforms*.

7. CONCLUSIONS

In this paper we model the temporal behavior of crowd performance using time series models. Further, we build a system that can automatically and continuously model crowd performance, by auto-tuning the parameters of the models, and can be used to predict crowd performance. We also propose a novel framework for optimally scheduling a large batch of tasks on multiple crowdsourcing platforms which are tested on a simulation testbed constructed from the automated system.

Our experiments show that adaptive learning algorithms outperform other algorithms that do not learn or rely on past data alone. In particular, Thompson Sampling based scheduling performs well in a wide variety of experimental conditions and we recommend this approach. We extend our approach to optimally schedule batches of tasks on multiple platforms, simultaneously. Our results show that multi-platform scheduling significantly outperforms scheduling only on a single platform. Principles described in this paper has been used to design a recommendation tool for enterprise users of popular crowd platforms.

In both the single as well as multi-platform approaches, the scheduling algorithms have been evaluated with simple parameter settings; multiple adjustable parameters and more complex objective functions need further testing. In the future, we will explore the effects of varying costs and task categories, on the platform models and scheduling algorithms.

REFERENCES

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Trans. Automat. Control* 19, 6 (1974), 716–723.

Barowy, D. W, Curtsinger, C, Berger, E. D, and McGregor, A. (2012). AutoMan: a platform for integrating human-based and digital computation. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications (OOPSLA '12)*. ACM, New York, NY, USA, 639–654. DOI: http://dx.doi.org/10.1145/2384616.2384663

Bernstein, M. S, Brandt, J, Miller, R. C, and Karger, D. R. (2011). Crowds in Two Seconds: Enabling Realtime Crowd-powered Interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 33–42.

Bernstein, M. S, Karger, D. R, Miller, R. C, and Brandt, J. (2012). Analytic Methods for Optimizing Realtime Crowdsourcing. *CoRR* abs/1204.2995 (2012).

Berry, D. A and Fristedt, B (Eds.). (1985). *Bandit problems: sequential allocation of experiments*. Chapman and Hall.

Bücheler, T. A, Lonigro, R, Füchslin, R. M, and Pfeifer, R. (2011). Modeling and Simulating Crowdsourcing as a Complex Biological System: Human Crowds Manifesting Collective Intelligence on the Internet. In *ECAL 2011. The Eleventh European Conference on the Synthesis and Simulation of Living Systems (ECAL)*, Tom Lenaerts, Mario Giacobini, Hugues Bersini, Paul Bourgine, Marco Dorigo, and Renï£¡ Doursat (Eds.). MIT Press, Boston, Mass., 109–116.

Celis, E, Dasgupta, K, and Rajan, V. (2013). Adaptive Crowdsourcing for Temporal Crowds. In *3rd Temporal Web Analytics Workshop, Rio de Janeiro, Brazil Accepted, to appear*.

Chapelle, O and Li, L. (2011). An empirical evaluation of thompson sampling. In *Neural Information Processing Systems (NIPS)*.

Dai, P, Mausam, , and Weld, D. S. (2010). Decision-Theoretic Control of Crowd-Sourced Workflows.. In *AAAI*, Maria Fox and David Poole (Eds.). AAAI Press.

Dasgupta, K, Rajan, V, Karanam, S, Ponnavaikko, K, Balamurugan, C, and Piratla, N. M. (2013). CrowdUtility: Know the crowd that works for you. In *CHI Extended Abstracts*. 145–150.

Difallah, D. E, Catasta, M, Demartini, G, Ipeirotis, P, and Cudre-Mauroux, P. (2015). The Dynamics of Micro-Task Crowdsourcing – The Case of Amazon Mechanical Turk. In *Proceedings of the 24th International World Wide Web Conference*.

Difallah, D. E, Demartini, G, and Cudré-Mauroux, P. (2013). Pick-a-crowd: tell me what you like, and i'll tell you what to do. In *WWW*. 367–374.

Eickhoff, C and Vries, A. P. (2013). Increasing cheat robustness of crowdsourcing tasks. *Inf. Retr.* 16, 2 (April 2013), 121–137. DOI:http://dx.doi.org/10.1007/s10791-011-9181-9

Faridani, S, Hartmann, B, and Ipeirotis, P. G. (2011). What′s the right price? pricing tasks for finishing on time. In *Proc. of AAAI Workshop on Human Computation*.

Hannan, E. J and Deistler, M. (1988). *Statistical theory of linear systems: Wiley series in probability and mathematical statistics.* John Wiley and Sons, New York.

Heidari, H and Kearns, M. (2013). Depth-Workload Tradeoffs for Workforce Organization.. In *HCOMP*, Björn Hartman and Eric Horvitz (Eds.). AAAI.

Hossfeld, T, Hirth, M, and Tran-Gia, P. (2011). Modeling of crowdsourcing platforms and granularity of work organization in future internet. In *Proceedings of the 23rd International Teletraffic Congress (ITC '11)*. ITCP, 142–149. http://dl.acm.org/citation.cfm?id=2043468.2043491

Ipeirotis, P. G. (2010). Demographics of mechanical turk. (2010).

Jung, H. J. (2014). Quality Assurance in Crowdsourcing via Matrix Factorization Based Task Routing. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion (WWW Companion '14)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 3–8.

Khazankin, R, Psaier, H, Schall, D, and Dustdar, S. (2011). QoS-Based task scheduling in crowdsourcing environments. In *Proceedings of the 9th international conference on Service-Oriented Computing (ICSOC'11)*. Springer-Verlag, Berlin, Heidelberg, 297–311. DOI:http://dx.doi.org/10.1007/978-3-642-25535-9_20

Khazankin, R, Schall, D, and Dustdar, S. (2012). Predicting qos in scheduled crowdsourcing. In *Proceedings of the 24th international conference on Advanced Information Systems Engineering (CAiSE'12)*. Springer-Verlag, Berlin, Heidelberg, 460–472. DOI: http://dx.doi.org/10.1007/978-3-642-31095-9_30

Kulkarni, A, Gutheim, P, Narula, P, Rolnitzky, D, Parikh, T, and Hartmann, B. (2012). Mobileworks: Designing for quality in a managed crowdsourcing architecture. *Internet Computing, IEEE* 16, 5 (2012), 28–35.

Little, G, Chilton, L. B, Goldman, M, and Miller, R. C. (2010). TurKit: Human Computation Algorithms on Mechanical Turk. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 57–66.

Liu, X, Lu, M, Ooi, B. C, Shen, Y, Wu, S, and Zhang, M. (2012). CDAS: a crowdsourcing data analytics system. *Proc. VLDB Endow.* 5, 10 (June 2012), 1040–1051. http://dl.acm.org/citation.cfm?id=2336664.2336676

Long, J. S. (1997). *Regression Models for Categorical and Limited Dependent Variables*. Thousand Oaks: Sage Publications.

Minder, P, Seuken, S, Bernstein, A, and Zollinger, M. (2012). CrowdManager - Combinatorial allocation and pricing of crowdsourcing tasks with time constraints. In *Workshop on Social Computing and User Generated Content in conjunction with ACM Conference on Electronic Commerce (ACM-EC 2012)*. Valencia, Spain, 1–18.

Rajan, V, Bhattacharya, S, Celis, L. E, Chander, D, Dasgupta, K, and Karanam, S. (2013). CrowdControl: An online learning approach for optimal task scheduling in a dynamic crowd platform. In *ICML Workshop: Machine Learning Meets Crowdsourcing*.

Roy, S. B, Lykourentzou, I, Thirumuruganathan, S, Amer-Yahia, S, and Das, G. (2013). Crowds, not Drones: Modeling Human Factors in Interactive Crowdsourcing. In *DBCrowd*. 39–42.

Saxton, G, Oh, O, and Kishore, R. (2013). Rules of Crowdsourcing: Models, Issues, and Systems of Control. *Inf. Sys. Manag.* 30, 1 (Jan. 2013), 2–20. DOI: http://dx.doi.org/10.1080/10580530.2013.739883

Schwartz, G. (1978). Estimating the dimensions of a model. *Annals of Statistics* 6 (1978), 461–464.

Srinivas, N, Krause, A, Kakade, S, and Seeger, M. (2010). Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2010)*. 1015–1022.

Tran-Thanh, L, Stein, S, Rogers, A, and Jennings, N. R. (2012). Efficient crowdsourcing of unknown experts using multi-armed bandits. In *European Conference on Artificial Intelligence*. 768–773.

Wang, J, Faridani, S, and Ipeirotis, P. (2011). Estimating the completion time of crowdsourced tasks using survival analysis models. *Crowdsourcing for search and data mining (CSDM 2011)* 31 (2011).