CrowdControl: An online learning approach for optimal task scheduling in a dynamic crowd platform

Vaibhav Rajan VAIBHAV.RAJAN@XEROX.COM
Sakyajit Bhattacharya SAKYAJIT.BHATTACHARYA@XEROX.COM
L. Elisa Celis ELISA.CELIS@XEROX.COM
Deepthi Chander DEEPTHI.CHANDER@XEROX.COM
Koustuv Dasgupta KOUSTUV.DASGUPTA@XEROX.COM
Saraschandra Karanam SARASCHANDRA.KARANAM@XEROX.COM
Xerox Research Centre India, Bangalore, India

Abstract

The dynamic nature of crowd platforms poses an interesting problem for users who wish to schedule a large set of tasks on a given platform. Although crowd platforms vary in their performance characteristics, certain temporal patterns can be discerned and statistically modeled. Methods that can learn these patterns and adapt as the patterns change can schedule "the right number of tasks with the right price at the right time" which can have significant implication on how well the tasks are completed. To address the problem, we propose *CrowdControl*: a novel online approach that controls and co–ordinates the execution of crowd tasks, in real–time, involving simultaneous learning of crowd performance and optimization based on the learning. We design and compare several algorithms in this framework. We also describe dynamic statistical models of crowd performance based on real data and a simulation testbed for evaluating CrowdControl algorithms. Our experiments show that algorithms that schedule jobs by adaptively learning current crowd performance can significantly outperform other algorithms that do not learn or rely on past data alone.

ICML Workshop: Machine Learning Meets Crowdsourcing, Atlanta, Georgia, USA, 2013.

1. Introduction

The dynamic nature of crowd platforms poses a very interesting problem for requesters who want to schedule a (large) set of tasks in a given platform. Unlike an organizational environment, where workers have timings, known skill sets and performance indicators that can be monitored and controlled, most crowd platforms leverage the capabilities of "fleeting" workers who exhibit dynamically changing work patterns, expertise, and quality of work (Ipeirotis, 2010). Further, there is little or no control a platform can exert over the worker population. In such an uncontrolled environment, platforms exhibit wide variance in terms of the observed performance characteristics such as completion time, accuracy, and task completion rates. In (Dasgupta et al., 2013; Karanam et al., 2013), the authors conduct experiments with digitization tasks (extracting handwritten fields from a form) crowdsourced to a platform[1] and measure the variations in performance characteristics. Figure 1 shows the hourly trends in mean accuracy and completion time, averaged over several days. While the variance of the entire data set is large, the variance we observe at a specific time of day is small.

In addition to hourly (or daily) variations, factors such as payments and size–complexity–redundancy of tasks are also shown to affect the performance obtained from a platform. The results strongly suggest that assigning "the right number of tasks with the right price at the right time" has significant implications on how well the tasks are completed. Poor decision making, especially by enterprises, can lead to Service Level Agreement (SLA) violations such as larger completion times, increased costs and/or inferior accuracy. This obser-

[1]Name undisclosed to protect privacy

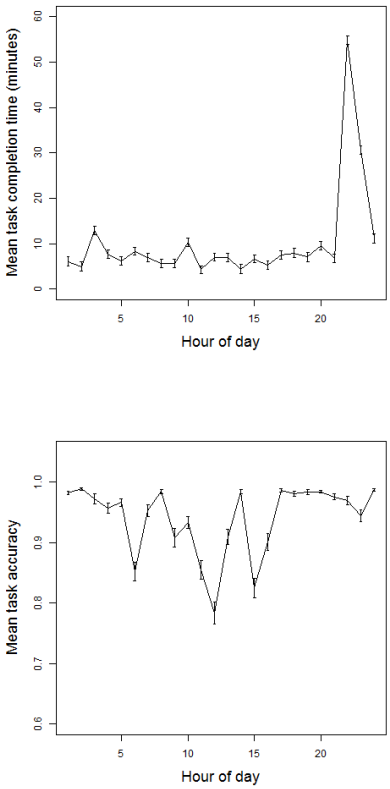

Figure 1. Hourly completion time (above) and accuracy (below) distributions on a real crowd platform averaged over a week

vation provides the motivation for the optimal task scheduling problem described in this paper.

1.1. Towards optimal task scheduling in dynamic crowd platforms

Users who have a large batch of tasks (to be crowdsourced) would like to have performance guarantees in terms of cost, completion time and accuracy. The scheduling problem in this case is to control and co–ordinate the execution of the tasks with the aim of optimizing the performance criteria.

A one–shot assignment of the entire batch cannot learn and dynamically explore different parameter settings to improve performance at different points of execution of the batch - for example, by offering more payments, relaxing time constraints, or tightening accuracy requirements. Another approach, illustrated in (Dasgupta et al., 2013) uses historical data to recommend the day and time which is most likely to fulfil the performance requirements of the user for the input batch of tasks. However, any predictions based on past data alone may or may not hold in runtime - the observed variations in real–time data (e.g., figure 1) indicate that performance could be significantly different over longer periods of time.

To address the problem, we propose CrowdControl - a novel online learning approach that controls and co–ordinates the execution of crowd tasks, in real–time, to meet user requirements. Task scheduling in CrowdControl progresses in rounds - where in each round a smaller subset of tasks is submitted to the crowd with a stipulated set of requirements (e.g., on cost/time/accuracy). Observations made in a previous round are used to refine the parameter settings for the next round such that requirements are further optimized. Note that the composition of the workers are bound to change at different points of time (rounds) and so do the responses received at different instances. The key to the proposed method thus involves simultaneous learning of crowd performance and optimization based on the learning.

1.2. Performance data and models

Performance in a crowd platform has been the subject of numerous previous studies (see section 1.4). However, almost all of prior art assumes complete control, knowledge of the workers' availabilities and abilities and/or knowledge of the internal architecture of the platforms. Clearly, such knowledge is not available to a requestor when working with third party platforms. Even when there is historical data with a platform owner, it might have little or no correlation with current operating conditions.

In contrast to previous models, our models (of the data as well as those intrinsic to CrowdControl) 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 its generality - 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 have been successfully used (Dasgupta et al., 2013) to design a classification based approach for platform recommendation. The more complex situation of adaptively switching between platforms based on observed performance has been studied in (Celis et al., 2013). Note that, while we introduced and utilized EOC–based models in our previous work none of them addresses the problem of optimal task scheduling on crowd platforms.

In this paper, we also present new EOC–based performance models of accuracy and completion time of real crowd tasks submitted over one week to a crowd platform. We extend these models in numerous ways to generate simulations of crowd performance over several weeks. This enables us to evaluate CrowdControl algorithms under various controllable experimental conditions.

1.3. Summary of our contributions

- A framework for scheduling tasks on a crowd platform in real–time that leverages the dynamic performance characteristics of the crowd. We design and evaluate of novel algorithms, in the framework, that simultaneously learn crowd performance and schedule tasks effectively.

- Dynamic statistical models of crowd performance based on real data and a simulation testbed for evaluating algorithms.

1.4. Related Work

Karger et al. discuss a general model of crowdsourcing tasks and design an algorithm for task assignment to workers and for inferring correct responses which outperforms simple majority voting (Karger et al., 2011). The aim is to minimize the budget and achieve a level of reliability characterized in terms of worker reliabilities. Tran–Thanh et al. also address the problem of assigning tasks to a set of workers with varying working costs (Tran-Thanh et al., 2012). They take into account the utility of each task to a worker, the quality of a worker and the limits on the number of tasks a worker performs – finally tasks are assigned to maximize overall utility. Minder et al. and Khazankin et al. also provide combinatorial optimization frameworks for the offline task allocation problem under price/QoS constraints (Minder et al., 2012; Khazankin et al., 2012). In the online version of the problem, workers arrive in an online manner and must be assigned to a task. The goal is to allocate tasks to workers and optimize the benefit of the requester. Ho and Vaughan extend the solution of the online adwords problem to solve this problem (Ho & Vaughan, 2012). Faridani et al. use a survival analysis model to find the right price for each worker assigned to a task (Faridani et al., 2011). None of these solutions address the problem of scheduling tasks in the presence of rapidly changing characteristics of the underlying platform. Further, none of these solutions work when there is little or no knowledge of the internals of the platforms.

2. The Crowd Control Scheduler

Based on the role each parameter assumes in scheduling, we classify the EOC parameters into three categories: 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 judgments. *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.

In this study we restrict ourselves to the following parameters. Two response parameters, tested independently: mean completion time and the mean–to–variance ratio of the accuracy. (The algorithms are described assuming a maximization setting; the changes for minimization are obvious.) Batch size is the single adjustable parameter. Total completion time is a limit parameter, which is used to set the time limit for each round of the scheduler. 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.

2.1. Scheduling Algorithms

The general template of a CrowdControl algorithm is shown below.

Algorithm CrowdControl Scheduler

Input: Tasks T, Platform P, Objective function f, Historical Data H
Initialize: Completed Tasks, $C = \phi$; Incomplete tasks, $I = T$; Timeout t
repeat
  Select b tasks from I: $R \subseteq I, |R| = b$
  Send R to platform P
  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 parameter (b) is 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.

2.1.1. Optimistic Scheduler (OS)

The 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. We call it the Optimistic Scheduler (OS).

2.1.2. Best guess scheduler (BGS)

The Best Guess Scheduler (Algorithm BGS) uses historical data (performance of previously sent tasks) to determine the distribution of batch sizes to be used for the schedule. Let b_{max} denote the maximum batch size to be sent in any round of the scheduler. Let f_{max} be the maximum value of the objective function. We call the pair (day of week, time of day) a *timepoint*. The distribution is determined as follows. We sort the timepoints of the past in descending order of the average value of the objective function seen at the timepoint. Let f_t be the objective function value at timepoint t. For each timepoint in the sorted list, if the timepoint occurs within the limits of completion time, we assign in our schedule a batch size $b = b_{max}\frac{f_t}{f_{max}}$ (i.e., a batch size proportional to the performance seen in the past). After iterating over the sorted list, if there are more batches to be assigned to the schedule, then batch sizes are allocated randomly in the schedule until the entire batch is scheduled.

2.1.3. Gaussian Process Upper Confidence Bound 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): $y_i = f(x_i) + \epsilon$ 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 a CrowdControl 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). For CrowdControl it is also possible to obtain the first prior by training the GP using 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 size from the UCB rule.
- Perform Bayesian update on μ, σ using $f(R_c)$ in each round

2.1.4. Thompson Sampling based Scheduler (TS)

The multi–armed bandit framework (Berry & 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). 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 tasks.

A simple and efficient bandit algorithm is Thompson sampling (Chapelle & 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.

The CrowdControl scheduler naturally 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 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) $b = arg\ max_i\, \theta_i$
• 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

2.1.5. 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 size. If the crowd response is worse than the previously seen value, it halves the batch size.

Algorithm Gradient based Scheduler (GS)

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

3. Data and Models

In this section we describe the data and present statistical models based on the observed performance characteristics. While the models are of independent interest, they also provide the foundation for a simulation testbed which can be used as a proxy crowd for experimentation. This is particularly useful since going to the crowd directly may come at a significant cost.

Our data consists of the performance data (completion time and accuracy) on a platform taken hourly over a one week period. In Section 3.1 we model this data as a longitudinal time series. Then, in Section 3.2, we add noise and further variability in an informed manner, based on observed correlations in the sample data, in order to simulate the performance characteristics of a platform into the future. The noise and time series parameters are tunable, which allows for a wide range of simulation models. Data simulated in this manner is used to evaluate CrowdControl algorithms.

3.1. Performance models based on real data

The data is collected by posting batches of 50 forms to the crowd, every hour of the day, for all days of a week leading to 8400(= 50 × 24 × 7) observations. A crowd worker has to digitize one or more fields (alphanumeric strings) in the forms. Since the true answer is known to us (we only use gold data), the accuracy of the response is computed easily. Task Accuracy is given by $1 - L/N$ where L is the edit distance (minimum number of string manipulations, i.e. insertions/deletions/substitutions, required to transform one string to other) and N is the length of string. Task completion time is measured from the time of

posting the task to the time when results are received. For more details of the data collection procedure see (Karanam et al., 2013). In our modeling, we consider three independent variables: day of week (Monday to Sunday), hour of day (0000 to 2300) and batch size (1 to 50). The dependent variables are task completion time and accuracy.

Completion time is viewed as longitudinal data, starting at hour 0000 on Monday, since 50 measurements (one per task, batch of 50) are taken each hour. Allowing for heterogeneity, we fit a different ARMA model to each hour and choose the best using the Bayesian Information criterion. The fit is sufficiently good: the pseudo r–square value (Long, 1997) is 0.975.

As the tasks we sent were simple, the accuracy of the responses were high: around 80% of the responses had accuracy 1 (i.e., 100%). There is no apparent time–series structure since the values remain mostly unchanged over time. Since most of the accuracy values are 1, we fit a sparse regression model. 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. The pseudo r–square value is 0.998.

3.2. Extending performance model to simulations

In order to evaluate CrowdControl algorithms we create a testbed that simulates the performance of a crowd. Using these simulations we can generate performance data for several weeks (from our real data of one week). Moreover, 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 model described in the previous section. Although not validated by real data, the additional models are based on autocorrelations that are strongly suggested by the real data.

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:

- T_1 : For each hour, $Y_t = \phi_1\left(Y_{t-1}, Y_{t-2}, ..., Y_1\right)$ 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 : $Y_{i\times 50} = \phi_3\left(Y_{(i\times 50-1200}, Y_{i\times 50-2400}, ..., Y_{50}\right)$ 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. 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 $0 \le \alpha, \beta \le 1$ and $\alpha+\beta \le 1$ 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 3.2. 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–square 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 (repeated from figure 1) in figure 2. 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.

4. Experimental Results

We test the algorithms using the following parameter settings: $b_{max} = 50, b_{min} = 5, A = \{5, 10, 15, \ldots, 40, 45, 50\}, k = |A| = 10, t = 60$ min-

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

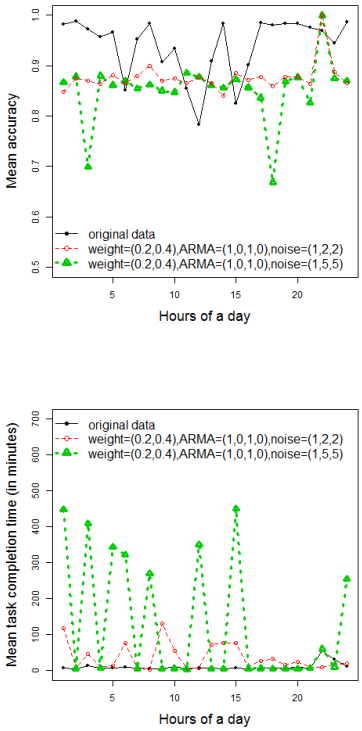

Figure 2. Plot of the average completion time (above) and accuracy (below) over 24 hours of a day: two simulations and the original data.

utes. 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 15000 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. Since most of the accuracy values for our tasks (and simulations) were high, we use the mean–to–variance ratio (instead of mean accuracy) as the objective function. The performance of an algorithm is measured by the value of the objective function for the entire batch of input tasks. These values, shown in figure 4 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.

In figure 3 we show an execution of two algorithms – GS and TS – over 100 rounds for a single simulation. The bars represent the batch sizes selected

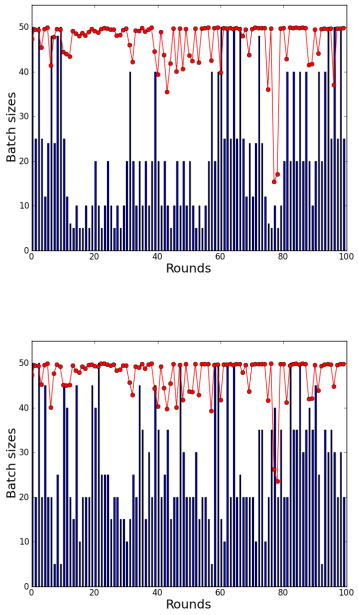

Figure 3. An execution of two algorithms – GS (above) and TS (below) – over 100 rounds for a single simulation. The bars represent the batch sizes selected in each round. For better visualization we plot the negative of the completion time with an offset of 50

in each round. For better visualization we plot the negative of the completion time with an offset of 50: thus, the higher the value on this curve, the better the (simulated) performance of the crowd is. Thompson sampling based scheduler, TS, is clearly more reactive, adapts quickly to varying completion times and creates a schedule with higher performance. As we see subsequently, algorithm TS has the best performance in nearly every experiment we conduct.

The first column of figure 4 (above: accuracy, below: completion time) shows the performance of the algorithms averaged over all the simulation parameters (weights, ARMA coefficients and noise parameters). Algorithm TS 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 figure 4 shows the performance of the algorithms separately for each of the three weights in the simulations. For each algorithm there are three bars, one for each weight setting, which represents the algorithm's performance averaged over all other simulation parameters (ARMA coefficients and noise parameters). Once again, TS performs the best.

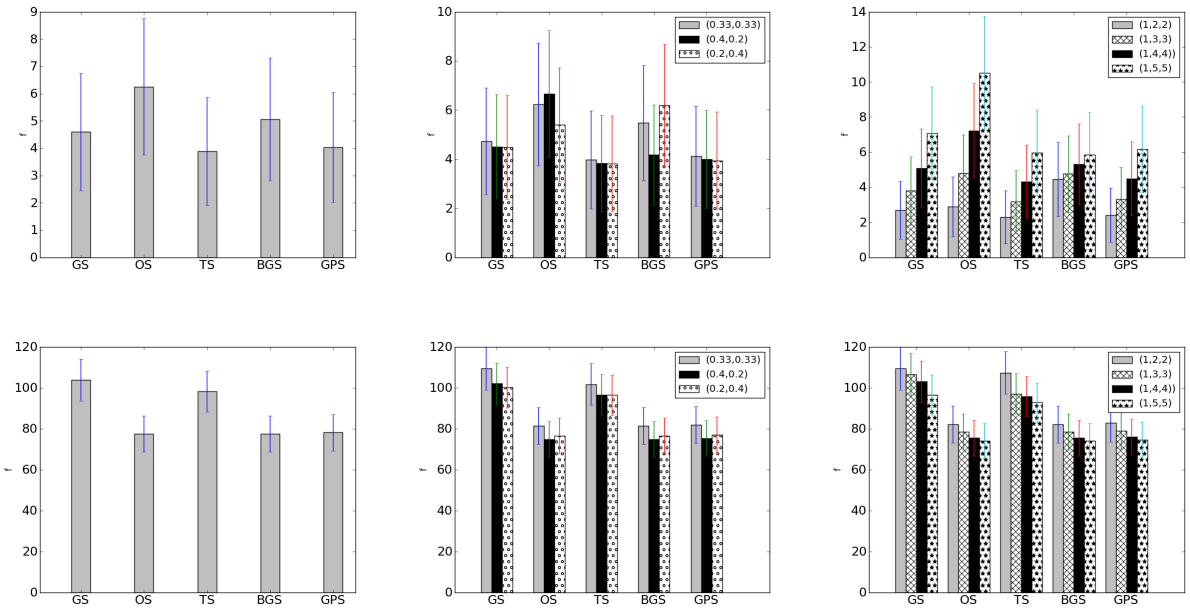

Figure 4. Average Performance of Algorithms: mean completion time (above), mean–to–variance ratio of accuracy (below). Left column: averages over all simulations; Middle column: three bars, one for each weight setting, averaged over all other simulation parameters; Right column: four bars, one for each noise setting, averaged over all other simulation parameters.

Note that the performance is better in the case where the weights of the second and third time–series structures are higher. Thus with larger deviations from the model, as might be expected in the real world, TS is able to adapt faster and schedule better than other methods.

The last column of figure 4 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 and ARMA coefficients). 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, TS adapts faster and creates better schedules.

Overall, algorithms GS and GPS also perform well, GS comparable to TS when mean–to–variance ratio of accuracy is optimized and GPS comparable to TS when mean completion time is optimized. We also compute the performance of the globally *optimal schedule* (OPT) The average mean–to–variance ratio of OPT is 710.22 (due to very low variance) and the average completion time is 0.39 minutes. Note that OPT, that has complete knowledge of the performance of the crowd in advance, is impossible in reality and can only be computed in simulations.

5. Conclusion

In this paper we propose CrowdControl: a novel framework for scheduling a batch of tasks on a crowd platform which aims at simultaneously learning crowd performance and optimizing the performance achieved on an input batch of tasks. We also describe statistical models of performance based on real data which are extended to create a novel simulation testbed for evaluating our algorithms. Our experiments show that adaptive learning algorithms can significantly outperform other algorithms that do not learn or rely on past data alone. In particular, Thompson Sampling based scheduling performed well in a wide variety of experimental conditions and we recommend this approach.

In the future, we intend to validate both the models and algorithms on more real data from the crowd. The algorithms have been evaluated with simple parameter settings (e.g., only batch size is varied in each round); multiple adjustable parameters and more complex objective functions need further testing. The effects of varying costs and task categories on the models and algorithms is also worth exploring.

References

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

Celis, Elisa, Dasgupta, Koustuv, and Rajan, Vaibhav. Adaptive crowdsourcing for temporal crowds. In *3rd Temporal Web Analytics Workshop, Rio de Janeiro, Brazil Accepted, to appear*, 2013.

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

Dasgupta, Koustuv, Rajan, Vaibhav, Karanam, Saraschandra, Balamurugan, Chitralekha, and Piratla, Nischal. Crowdutility: Know the crowd that works for you. In *Extended Abstracts, ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2013.

Faridani, Siamak, Hartmann, Björn, and Ipeirotis, Panagiotis G. Whats the right price? pricing tasks for finishing on time. In *Proc. of AAAI Workshop on Human Computation*, 2011.

Ho, Chien-Ju and Vaughan, Jennifer Wortman. Online task assignment in crowdsourcing markets. In *AAAI Conference on Artificial Intelligence*, 2012.

Ipeirotis, Panos. A computer scientist in a business school, 2010. URL http://www.behind-the-enemy-lines.com/.

Karanam, Saraschandra, Rajan, Vaibhav, and Dasgupta, Koustuv. Understanding dynamic performance variability across multiple crowdsourcing platforms. In *Short Paper, ACM Web Science, Accepted, to appear*, 2013.

Karger, David R, Oh, Sewoong, and Shah, Devavrat. Iterative learning for reliable crowdsourcing systems. In *Neural Information Processing Systems*, 2011.

Khazankin, Roman, Schall, Daniel, and Dustdar, Schahram. Predicting qos in scheduled crowdsourcing. In *Advanced Information Systems Engineering*, pp. 460–472. Springer, 2012.

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

Minder, Patrick, Seuken, Sven, Bernstein, Abraham, and Zollinger, Mengia. 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 (JUN 2012)*, pp. 1–18, 2012.

Srinivas, Niranjan, Krause, Andreas, Kakade, Sham, and Seeger, Matthias. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2010)*, pp. 1015–1022, 2010.

Tran-Thanh, Long, Stein, Sebastian, Rogers, Alex, and Jennings, Nicholas R. Efficient crowdsourcing of unknown experts using multi-armed bandits. In *European Conference on Artificial Intelligence*, pp. 768–773, 2012.

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