Explicit Preference Elicitation for Task Completion Time

Mohammadreza Esfandiari[#1], Senjuti Basu Roy[#1], Sihem Amer-Yahia[#2]
{me76,senjutib}@njit.edu,Sihem.Amer-Yahia@imag.fr
[#1] New Jersey Institute of Technology, USA, [#2] Univ. Grenoble Alpes, CNRS, France

ABSTRACT

Current crowdsourcing platforms provide little support for worker feedback. Workers are sometimes invited to post free text describing their experience and preferences in completing tasks. They can also use forums such as Turker Nation[1] to exchange preferences on tasks and requesters. In fact, crowdsourcing platforms rely heavily on observing workers and inferring their preferences implicitly. On the contrary, we believe that *asking workers to indicate their preferences explicitly* will allow us to improve different processes in crowdsourcing platforms. We initiate a study that leverages explicit elicitation from workers to capture the evolving nature of worker preferences and we propose an optimization framework to better understand and estimate task completion time. We design a worker model to estimate task completion time whose accuracy is improved iteratively by requesting worker preferences for task factors, such as, required skills, task payment, and task relevance. We develop efficient solutions with guarantees, run extensive experiments with large scale real world data that show the benefit of explicit preference elicitation over implicit ones with statistical significance.

ACM Reference Format:
Mohammadreza Esfandiari, Senjuti Basu Roy, Sihem Amer-Yahia. 2018. Explicit Preference Elicitation for Task Completion Time. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271667

1 INTRODUCTION

The main actors of a crowdsourcing platform are requesters and tasks, and workers who complete them. Understanding quality indicators in crowdsourcing has been a recent research focus [6, 12, 13, 20]. Some work focuses on estimating quality indicators such as engagement and motivation [19, 21, 24], and on revisiting this estimation periodically in an implicit manner. An important open question however is, **can we improve the estimation of quality indicators by seeking explicit preferences from workers?**

On Amazon Mechanical Turk[2] or Prolific Academic,[3] a task has factors such as *type* (e.g., image annotation, ranking, sentiment analysis), *payment* and *duration*, i.e., the time allotted to complete a task. Workers are characterized by their *preferences for task factors* [1, 27]. Our first contribution is to propose *an optimization framework* `ExPref`, within which an individual *worker model*, that captures *the preference of workers for task factors*, is learned and maintained to estimate *task completion time. Worker Model* is "supervised" in nature and it is initialized by deriving principles from *active learning* [5]. Indeed, worker preference on task factors, such as, payment, task types, are indicators of how much time the she needs to complete the task. Task completion time is an important quality indicator in crowdsourcing platform, as deeper understanding and analysis of task completion time benefits customization of payment strategies [12, 13], task assignment, and appropriate recruitment of workforce for crowdsourcing platforms [17, 18, 24, 28].

Unless `ExPref` is updated periodically, it is likely to become outdated, as worker's preferences evolve over time (e.g., a worker's skills improve as she completes tasks). To update the model, we advocate the need to **explicitly elicit from a worker her preferences**. **That is a stark departure from the literature where workers are observed and their preferences computed implicitly**. An additional challenge arises to address the following question: *in what fashion these preferences should be extracted and used to produce a scalable and accurate model?*. To that end, we present **Question Selector** for optimizing preference elicitation that asks a worker to rank the k task factors that lead to minimize error in the model. For example, a worker may be asked *"Rank task relevance and payment"*. A higher rank for payment will indicate the worker's preference for high paying tasks over those most relevant to her profile. We prove that optimally selecting k questions, i.e., k task factors, for explicit worker preference is NP-hard, even when the *Worker Model* is linear. Consequently, we develop an efficient alternative using an iterative greedy algorithm that has a provable approximation bound. Once the worker provides her preference, the *next challenge is how to consume that feedback to update Worker Model in a principled manner.* We present **Preference Aggregator** to update the *Worker Model* with the elicited preferences. To ensure that `ExPref` does not necessarily incur additional burden, worker's response can be : *a total order over those k factors*; or *a partial* preference over a subset of those k factors, when the worker does not/can not provide total ordering. Of course, an extreme case is that the worker does not provide any preference and in that case the *Worker Model* is updated implicitly. We formulate those choices as a constrained optimization problem and develop efficient solutions.

We run experiments that measure the quality and scalability of `ExPref`. We use 165, 168 real micro-tasks from CrowdFlower involving 58 workers hired from Amazon Mechanical Turk. We measure the accuracy of `ExPref` against multiple baselines, including existing ones that leverage implicit preference computation [24]. We show that **soliciting preferences explicitly and using them to update the model achieves greater stability in a short number of iterations and outperforms implicit preference based solutions with statistical guarantees**. We also show that task

[1]http://turkernation.com/
[2]https://www.mturk.com/
[3]https://www.prolific.ac/

CIKM '18, October 22–26, 2018, Torino, Italy
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271667

Notation	Definition
n, m	# tasks, # task factors
$\vec{t}$	task t represented by a vector of m factors
$\mathcal{T}$	Task factor matrix
$\vec{w}$	worker w's preference vector
Q^k	a set of selected k questions
y_t	completion time of task t
$\vec{Y}$	vector representing y_t over a set of tasks
$\mathcal{F}$	Worker Model

Table 1: Table of important notations

completion time is highly correlated with the quality of the completed tasks - hence a deeper analysis on task completion time helps improve crowdsourced task quality. We present case studies as anecdotal evidence and show that `ExPref` truly captures worker preference. We finally measure scalability and demonstrate that our proposed solutions scale well. CrowdCur [9] is the real world implementation of these research ideas.

In summary, our contributions are:

- `ExPref`, a framework that elicits explicit worker preference to better estimate task completion time. `ExPref` has a *Worker Model* that captures worker preferences for task factors.
- A formalization of two core problems: **Question Selector** that asks a worker to rank k task factors, and **Preference Aggregator** that updates the model with elicited preferences.
- An in-depth analysis and solutions with provable guarantees for the *Worker Model*, the **Question Selector**, and the **Preference Aggregator**.
- Extensive experiments that corroborate that explicit preference elicitation outperforms implicit preference computation [24] and that our framework scales well.

2 FRAMEWORK AND FORMALISM

We present our proposed framework and formalize the problems.

2.1 `ExPref` Framework

We propose an iterative framework `ExPref` (refer to Figure 1) that is designed to ask personalized questions to a worker to elicit her preferences. The rationale is that while task factors are stable, a worker's preference evolves as workers undertake tasks [16, 24]. We propose a *Worker Model* that consumes task factors and predicts for a task, how long will the worker spends on the task, *by inferring her preferences.* However, unless the *Worker Model* is refreshed or updated periodically, it is likely to become outdated, as worker preference evolves over time [1, 16, 24]. To update the model, one has to periodically invoke an explicit preference elicitation step, called **Question Selector** that selects a set of k task factors and asks worker w to rank them. Once the worker provides her preference, the *Worker Model* is updated by the **Preference Aggregator**.

This information could be used in many places to characterize the workforce of a crowdsourcing platform and enable several improvements such as the analysis of workers' fatigue [16] and motivation, and better task assignment to workers [17, 18, 24, 28].

Two computational problems form the heart of this framework. **1. Question Selector:** - when invoked, selects the best set of k questions to elicit a worker's preference for task factors. **2. Preference Aggregator:** - takes a worker's preference to the questions into account, and updates the *Worker Model.* The last two components work in sequence, given the necessity to refine the learned model. The technical challenge is to update the model while satisfying the preference the worker has provided.

The remainder of the paper focuses on a particular worker w, unless otherwise stated - i.e., each of the components of the framework is designed or invoked for her.

2.2 Data Model and Problem Definitions

Task Factors. Task characteristics are commonly defined by the platform and their values by requesters. Each task t in a set of n given tasks is characterized by a set of m factors whose values are either explicitly present or could be extracted (such as keywords, duration, pay-off). For this work, we assume that for every task, its factors are given. This gives rise to a task factor matrix $\mathcal{T}$.

Example 2.1. The matrix in Table 2.1 contains 6 tasks characterized by factors, such as *type, payoff, duration.* Example types are image annotation, ranking and sentiment analysis. Payoff determines the \$ value the workers receives as payment, whereas, duration is an indication of the maximum time a worker needs to complete that task.

task − id	*annotation*	*ranking*	*sentiment*	*payoff*	*duration*	completion time
$t1$	1	0	0	20	35	25
$t2$	1	0	0	5	5	35
$t3$	0	1	0	5	10	45
$t4$	0	1	0	5	40	5
$t5$	0	0	1	10	10	12
$t6$	0	0	1	20	30	23

Given a task t that a worker w undertakes (either via self-appointment or via an assignment algorithm), we are interested to understand and estimate task completion time, the time spent by the worker to perform the task. The last column of the task factor matrix indicates completion time of the individual tasks. When the worker is arriving in the platform for the first time, we use a budget b to initialize the *Worker Model* by asking workers b questions. Afterwards, we periodically update the model by seeking explicit feedback through **Question Selector** and update workers preference using **Preference Aggregator** in the *Worker Model.*

Worker Preferences. The preferences of a worker w are represented by a vector $\vec{w}$ of length m that takes real values and determines the preferences over the task factors. Using Example 2.1, $\vec{w}$ could be represented as a set of weights for the task factors, such as, {*duration,payment*}.

Worker Model. Central to our framework is a model that consumes task factors and given a worker's history, infers her preference vector to estimate task completion time. It is easy to notice that task completion time is continuous in nature.

Explicit Questions. An explicit question q is asked to elicit w's preference on a particular task factor, assuming there is an one to one correspondence between the questions and task factors (thus, every task factor is a potential question and total possible questions m). A set of k questions is asked to obtain a preferred order among a set of k task factors (where k is part of the input). As an example, one may ask to *"Rank task duration, annotation tasks, ranking tasks, sentiment analysis tasks, payment".* A worker may provide a full order among these 5 factors as her preference, or may provide partial order of preference. As an example of the former, she may rank *payment*, then *duration*, then *ranking tasks*, followed by *sentiment analysis*, and finally *annotation tasks.* On the contrary, her preference is partial, when the worker prefers, *payment* over *duration*, but does not explicitly say anything about the rest.

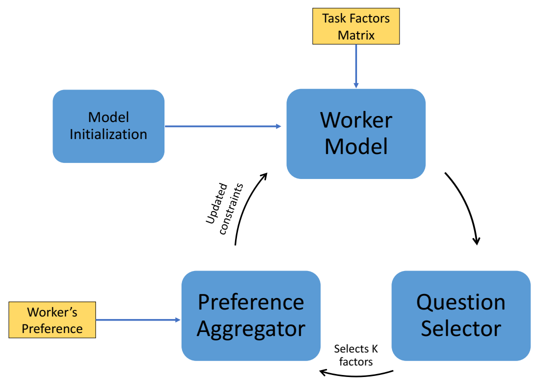


Figure 1: The ExPref Framework

2.2.1 **Problem Definition: Worker Model.** Given the task factor matrix $\mathcal{T}$ of a set of n tasks, where each task t is described by m factors and associated with a continuous variable y_t denoting the time the worker spends on t, we are interested in estimating the worker preference vector $\vec{w}$. The *Worker Model* $\mathcal{F}$ is a linear aggregate function over $\mathcal{T}$ and $\vec{w}$, denoted as $\mathcal{F} = \vec{w}^T \cdot \mathcal{T}$.

Optimization goal. Our objective is to estimate $\vec{w}$ in $\mathcal{F}$, such that it minimizes the *reconstruction error* [10], i.e.,

$$\mathcal{E} = ||\vec{w}^T \cdot \mathcal{T} - \vec{Y}||_2^2 \tag{1}$$

Once the model is built, it can estimate the completion time of a future task by the worker. Using Example 2.1, $\mathcal{F}$ can estimate the completion time of any of the 6 tasks or other future tasks, by consuming $\mathcal{T}$.

Initialization. How to initialize the *Worker Model* is a challenge. Initially when a brand new worker w joins the platform, as no past history of w is available, she is treated akin to a "cold worker". Initially we have no information of task completion, hence $\vec{w}$ can't be estimated accurately. The objective of the model initialization is to select a subset of tasks $\mathcal{B}$ ($|\mathcal{B}| = b$, given as a budget) to be completed by w and build $\mathcal{F}$ using that.

One obvious choice is to randomly select b samples (tasks) to initialize the model. However, we argue there does exist further merit in careful selection of initial set of b tasks (training inputs), as a careful selection of the training examples (inputs), will generally need far fewer examples in comparison to selecting them at random from some underlying distribution.

Our proposed formalism is inspired from *active learning* in Machine Learning [5] that is iterative in nature and is optimized to select one more input (i.e., a task) at a time that maximizes the accuracy of the underlying model. For us, from the available candidate pool of tasks that are not yet undertaken by the worker, this translates to selecting one task t (input) at a time, the worker w undertakes it, and we record its completion time y_t. [4]

Optimization goal. During the model initialization phase, in a single iteration, our objective is to select that task t that will give rise to a worker preference vector $\widehat{w}$, which is a good estimation of true worker preference $\vec{w}$ by minimizing the mean squared error of the maximum likelihood estimates between $\widehat{w}$ and $\vec{w}$.

$$E(||\widehat{w} - \vec{w}||)^2 \tag{2}$$

Given Example 2.1, if $b = 3$ and we have already selected two tasks ($t1, t2$), the objective would be to select the next best task that minimizes the mean squared error between $\hat{w}$ and $\vec{w}$.

2.2.2 **Problem Definition : Question Selector.** This module selects the best set of k questions for a worker w. The objective is to select those task factors that are responsible for the model's inaccuracy, i.e., removing them would improve the reconstruction error of $\mathcal{F}$ the most.

Optimization goal. Let $\mathcal{E}$ denote the current reconstruction error of $\mathcal{F}$ and $\hat{\mathcal{E}}$ denote it when k task factors are removed. Given Q, the k questions are selected such that the model reconstruction error improves the most, i.e., $argmax_{\{Q^k \in Q:|Q^k|=k\}}(\mathcal{E} - \hat{\mathcal{E}}_{Q-Q^k})$.

Using Example 2.1, if $k = 2$, this will select any two of the five task factors in the task factor matrix.

2.2.3 **Problem Definition : Preference Aggregator.** The preferences provided by a worker for task factors, could be expressed as a set of constraints of the form, $i > j, j > l$. Worker preference could take one of the three following forms:

(1) **Full Order:** The worker can provide a full order over the k selected factors. The full ranking will be in the form of $i > j > k > l$, if i, j, k, l are the task factors (questions). This preference could be expressed as a set of $k - 1$ pairwise linear constraints of the form, $i > j, j > k$, and etc.
(2) **Partial Order:** The worker can provide a partial order instead, especially when she can not provide full order. Given the set of k factors, a partial order takes the form of $i > j$, but no preference is elicited between for k, l.
(3) **No Preference:** The worker does not provide any preference.

Optimization goal. Worker's preferences are taken as hard constraints. Given her preference, the objective is to relearn $\mathcal{F}$ that satisfies the preferences such that its reconstruction error is minimized. The objective therefore is to minimize $\mathcal{E}$ such that the constraints are satisfied.

Using Example 2.1, if worker w explicitly states that she prefers `annotation` tasks to `ranking` tasks, this preference is translated into constraints expressed on the worker preference vector. Those are then used by the preference aggregator to update $\mathcal{F}$.

3 ALGORITHMS

We present solutions that are efficient and come with guarantees.

3.1 Worker Model

As formalized in Section 2.2.1, the *Worker Model* $\mathcal{F}$ is a linear combination of task factors and worker preference over those factors. We design algorithms for the optimization problem that is intrinsic to the model, and then a solution for model initialization.

Efficient Algorithm. We first derive an alternative form of our optimization function and then show that we can use simple matrix algebraic techniques to solve the problem.

Equation 1 could be rewritten as

$$min_{\vec{w}}(\vec{w}^T \cdot \mathcal{T})^T(\vec{w}^T \cdot \mathcal{T}) = min_{\vec{w}}(\vec{w}^T\mathcal{T}^T\mathcal{T}\vec{w} - 2\vec{w}^T\mathcal{T}^T\vec{Y} + \vec{Y}^T\vec{Y}) \tag{3}$$

[4]An offline formulation of this problem, that is selecting the entire set $\mathcal{B}$ in one shot is computationally more expensive and we defer this study to future work.

To minimize Equation 3, we compute the gradient and set it to 0. i.e.,

$$\nabla(\vec{w}) = 2\mathcal{T}^T\mathcal{T}\vec{w} - 2\mathcal{T}^T\vec{Y} = 0 \tag{4}$$

This allows us to solve $\vec{w}$ by computing the matrix inverse.

$$\mathcal{T}^T\mathcal{T}\vec{w} = \mathcal{T}^T\vec{Y} \tag{5}$$

Which finally gives,

$$\vec{w} = (\mathcal{T}^T\mathcal{T})^{-1}\mathcal{T}^T\vec{Y} \tag{6}$$

$(\mathcal{T}^T\mathcal{T})^{-1}\mathcal{T}^T$ in Equation 6 is known as the Moore-Penrose pseudo-inverse matrix of $\mathcal{T}$. This alternative representation is valid as long as the task factor matrix $\mathcal{T}$ is invertible (or could be inverted by adding an additional term, refer to [3]).

With this alternative representation in Equation 6, computing the best estimation of $\vec{w}$ is done by matrix inversion and multiplication techniques.

Running Time. The overall running time of the algorithm is dictated by matrix multiplication (multiplying $(\mathcal{T}^T\mathcal{T})$, Matrix inversion (inverting $\mathcal{T}^T\mathcal{T}^{-1}$), followed by matrix multiplication (to obtain $(\mathcal{T}^T\mathcal{T})^{-1}\mathcal{T}^T$), and a final matrix multiplication (to obtain $(\mathcal{T}^T\mathcal{T})^{-1}\mathcal{T}^T$). The asymptotic complexity is $O(m^2n + m^3)$.

3.1.1 Initializing the Worker Model.

One major challenge to develop the "supervised" `Worker Model` is how to handle "cold workers" - brand new workers. If the platform does not have any information about such workers, we initialize the `Worker Model` $\mathcal{F}$ by judiciously selecting one task at a time and repeating this process b times.

Efficient Algorithm. As described in Section 2.2.1, we present an online problem formulation, which selects one task t at a time, the worker undertakes the task, we record the completion time and update $\mathcal{F}$. We repeat this process for b iterations. As expressed in Equation 2, our objective is to estimate $\widehat{w}$ which is a good estimator of $\vec{w}$ (the true preference vector of the worker). The model initialization algorithm has to select a task whose completion time is not yet known. Therefore, the challenge is, how to select a task without knowing its completion time, so that we meet the optimization goal.

We present an alternative representation of Equation 2 that lies at the heart of our algorithm. Interestingly, this alternative representation does not involve task completion time. Hence, just by looking at the task factor matrix, we can select the next best task that optimizes Equation 2. At a given run of this algorithm, let us assume then that we already have selected a few tasks that gives rise to $\mathcal{T}'$ task factor matrix. Equation 2 can be rewritten as (we omit the details for brevity and refer to [10, 26] for details)

$$E(||\widehat{w} - \vec{w}||)^2|\mathcal{T}' = Trace[(\mathcal{T}'^T\mathcal{T}')]^{-1} \tag{7}$$

Therefore, minimizing the error is same as minimizing $Trace[(\mathcal{T}'^T\mathcal{T}')]^{-1}$, where $Trace[.]$ is the matrix trace, the sum of its diagonal components.

Now, we are ready to describe the algorithm. Consider Equation 7 and let us assume $A = (\mathcal{T}'^T\mathcal{T}')^{-1}$. (assuming it is already invertible). We are now trying to select another input task t that adds one additional row $[t]$ to $\mathcal{T}'$. Therefore, now $\mathcal{T}'$ becomes

$$\begin{bmatrix}\mathcal{T}' & t^T\end{bmatrix} \begin{bmatrix}\mathcal{T}' \\ t^T\end{bmatrix} = \mathcal{T}'^T\mathcal{T}' + \begin{bmatrix}t\end{bmatrix}\begin{bmatrix}t\end{bmatrix} \tag{8}$$

This is equal to $AA^{-1} + tt^T$. Therefore, we would like to find a t that minimizes

$$Trace[(A^{-1} + tt^T)^{-1}] \tag{9}$$

This matrix inverse could actually be carried out in closed form and it becomes

$$Trace[(A^{-1} + tt^T)^{-1}] = Trace[A] - \frac{t^TAAt}{1 + t^TAt} \tag{10}$$

Now that $Trace[A] = Trace[(\mathcal{T}'^T\mathcal{T}')]^{-1}$, to minimize the mean squared error by adding a task, we choose that task that maximizes

$$\frac{t^TAAt}{1 + t^TAt} \tag{11}$$

Therefore, at a given iteration, the algorithm selects a task that maximizes Equation 11. To select the set $\mathcal{B}$, this process is repeated b times.

Running Time. This algorithm takes $O(m^2n + m^3)$ time to compute $\frac{t^TAAt}{1+t^TAt}$ of a single candidate task. In each iteration, it has to consider the entire pool of tasks to decide the best candidate. If the number of tasks is upper-bounded by n, one iteration of this algorithm takes $O(m^2n^2 + m^3n)$

Running Example: Suppose we have selected the first two tasks in Table 2.1. We describe how to select the third task to complete the set of $b = 3$ tasks for model initialization. Suppose the following two tasks have been selected in the previous iteration,

$$\mathcal{T} = \begin{bmatrix} \textit{annotation} & \textit{ranking} & \textit{sentiment} & \textit{payoff} & \textit{duration} \\ 1 & 0 & 0 & 20 & 35 \\ 0 & 1 & 0 & 5 & 10 \end{bmatrix}$$

Assume that $A = (\mathcal{T}^T\mathcal{T})^{-1}$. After calculating the value of A, we start by examining the 4 remaining tasks one at a time and we will calculate the value of $\frac{t^TAAt}{1+t^TAt}$ for each of them. Out of the 4 remaining tasks, one can see that $\frac{t^TAAt}{1+t^TAt}$ is maximized ($\frac{t^TAAt}{1+t^TAt} = 0.512$) for the task $t4$,

$$t4 = \begin{bmatrix} \textit{annotation} & \textit{ranking} & \textit{sentiment} & \textit{payoff} & \textit{duration} \\ 0 & 1 & 0 & 5 & 40 \end{bmatrix}$$

Therefore, this task is selected and given to the worker. Once she returns it, task completion time is recorded.

3.2 Question Selector

The **Question Selector** intends to select the k-task factors (i.e., questions) whose removal maximizes the improvement of the *Worker Model* $\mathcal{F}$. The idea is to present those factors to the worker and seek her explicit preference. A careful review of the objective function (refer to Section 2.2.1) shows that since $\mathcal{E}$ is a constant at a given point - thus, maximizing $(\mathcal{E} - \hat{\mathcal{E}}_{Q-Q^k}) : \{Q^k \in Q : |Q^k| = k\}$ is same as minimizing the reconstruction error of $\hat{\mathcal{E}}_{Q-Q^k}$, i.e., retaining the best $m - k$ factors (thus eliminating the worst k factors) that has the smallest reconstruction error of $\mathcal{F}$. The problem thus becomes selecting the best $m - k$ factors that have the smallest reconstruction error. The remaining k factors would therefore be chosen as the explicit questions for preference elicitation.

Theorem 3.1. *Optimally selecting k questions for explicit worker preference is NP-hard.*

Proof. (Sketch): When a linear model such as the one in Equation 1 is assumed, the problem of identifying and removing the k worst factors, i.e., retaining the best $m - k$ factors, is akin to selecting a subset of $m - k$ columns from the task factor matrix $\mathcal{T}$ such that the pseudo-inverse of this sub-matrix has the smallest norm.

Under the ℓ_2 norm, using the rigorous NP-hardness proof described in [3], our proof follows. Given an instance of that problem [3], we set k (the k worst factors to remove) as the difference between the total number of columns and k' (k'= the best set of k' columns giving rise to the submatrix whose pseudo-inverse has the smallest norm). The rest of the proof is trivial and omitted for brevity. □

3.2.1 **Efficient Algorithm.** Under the linear model such as the one described in Equation 1 and its equivalent representation using a pseudo-inverse matrix, the objective of identifying the set Q^k of k selected questions (thereby identifying $m - k$ best factors) out of a set Q of m questions (a task factor is a question) is equivalent to retaining the task factor submatrix with $m - k$ columns that is of the following form [26]:

$$\underset{Q^k \subset Q, |Q^k|=k}{\text{argmin}} \quad Trace(\mathcal{T}^T_{Q \backslash Q^k} \mathcal{T}_{Q \backslash Q^k})^{-1} \tag{12}$$

We now describe a greedy algorithm `K-ExFactor` to identify k worst task factors (thus retaining $m - k$ best factors). Our algorithm makes use of Equation 12 and has a provable approximation guarantee. It works in a backward greedy manner and eliminates the factors iteratively. It works in k iterations, and in the i-th iteration, from the not yet selected set of factors, it selects a question q_j and eliminates it which marginally minimizes $\text{Trace}(\mathcal{T}^T_{Q \backslash q_j} \mathcal{T}_{Q \backslash q_j})^{-1}$. Once the k^{th} iteration completes the eliminated k questions are the selected k-factors for explicit elicitation. The pseudo code of the algorithm is presented in Algorithm 1.

Algorithm 1 Algorithm `k-ExFactor`: Greedy Question Selector

```
Require: Task factor matrix $\mathcal{T}$, set of questions $Q$
1: $\mathcal{T}_Q \leftarrow \mathcal{T}$
2: $Q^s \leftarrow Q$
3: for $j \leftarrow 1$ to $k$ do
4:     $q_j \leftarrow argmin_{q \in Q}$ Trace$(\mathcal{T}^T_{Q \backslash q} \mathcal{T}_{Q \backslash q})^{-1}$
5:     $\mathcal{T}_Q \leftarrow \mathcal{T}_{Q \backslash j}$
6:     $Q^s \leftarrow Q^s \setminus q_j$
7: end for
8: Return $Q - Q^s$
```

Running Time. The algorithm runs in k iterations. Line 4 in Algorithm 1 requires a $O(m^2 n)$ time for matrix multiplication and inversion for the question under consideration. Therefore, the overall complexity is $O(km^2 n^2)$. Notice that most of the complexity is actually in the process of recomputing the model error and the actual question selection is rather efficient.

Theorem 3.2. *Algorithm* `k-ExFactor` *has an approximation factor of* $\frac{m}{m-k}$.

Proof. (sketch): The proof adapts from an existing result [2, 8] that uses backward greedy algorithm for *subset selection* for matrices and retains a given smaller number of columns such that the pseudo-inverse of the smaller sub-matrix has the smallest norm possible. These results adapt, as this is akin to removing k worst task factors and retaining the best $m - k$ factors. It is also shown in recent work [3] that the objective function is not submodular, nor is it supermodular or monotone. Exploration of a better approximation factor is deferred to future work. □

Running Example: Using Example 2.1, if $k = 3$, {sentiment, Payoff, Duration} are the three task factors for which worker feedback is solicited.

3.3 Preference Aggregator

We can now describe how to aggregate worker responses and incorporate her provided preferences into the model $\mathcal{F}$. Recall Section 2 and note that the worker provides either a full order among the selected questions (task factors), a partial order, or possibly no answer. For the last scenario, since the worker does not provide any feedback, we simply update $\mathcal{F}$ implicitly. This is done by updating the model without any constraints. However, for both full and partial orders, worker preference adds a set of linear constraints in the optimization function in $\mathcal{F}$.

3.3.1 **Efficient Algorithm.** Our solution treats partial and full order in a similar fashion. In both cases, they add linear constraints to the objective function. With the linear constraints added to our objective function in Equation 1, updating the *Worker Model* under preference aggregation problem becomes a constrained least squares problem.

Specifically, our problem corresponds to a box-constrained least squares one as the solution vector must fall between known lower and upper bounds. The solution to this problem can be categorized into active-set or interior-point [23]. The active-set based methods construct a feasible region, compute the corresponding active-set, and use the variables in the active constraints to form an alternate formulation of a least squares optimization with equality constraints [30]. We use the interior-point method that is more scalable and encodes the convex set (of solutions) as a barrier function. It uses primal Newton Barrier method to ensure the KKT equality conditions to optimize the objective function [23].

Running Time. Our proposed primal Newton Barrier interior-point is iterative and the exact complexity depends on the barrier parameter and the number of iterations, but the algorithm is shown to be polynomial [30].

Running Example: Using Example 2.1 again, if the worker says that she prefers `Duration > Sentiment > Payoff`, then the new weights that the preference aggregator estimates for $\mathcal{F}$ are, tagging=0.1, ranking= 0.1, sentiment=0.12, payoff=0.11, duration=0.97. Notice that the order of the task factors provided by the worker is satisfied in the updated model.

4 EXPERIMENTAL EVALUATIONS

We describe our experimental setup, steps, and findings in this section. All algorithms are implemented in Python 3.5.1 using Intel Core i7 4GHz CPU and 16GB of memory and Linux operating system. All the numbers are presented as an average of 10 runs.

4.1 Dataset Description

We use 165, 168 micro-tasks from CrowdFlower. A task belongs to one of the 22 different categories, such as, tweet classification, searching information on the web, audio transcription, image tagging, sentiment analysis, entity resolution, etc. Each task type is assigned a set of keywords that best describe its content and a payment, ranging between \$0.01 and \$0.12. These are *micro-tasks* that take less than a minute to complete.

Initially, we group a subset of micro-tasks into 240 Human Intelligence Tasks (HITs) and publish them on Amazon Mechanical Turk. Each HIT contains 20 tasks and has a duration of 30 minutes. A worker who accepts a HIT is redirected to our platform to complete the tasks. A worker may complete several HITs in a work session and gets paid for every completed *micro-task*.

Task Factors. The task types along with other factors, such as, payment and duration, form the task factors. Our original data has 41 task factors that are continuous, categorical or binary. By involving domain experts, we binarized these them to obtain a total of 100 factors that uniquely characterize the tasks.

Worker and Keywords. Each hired worker has to previously complete at least 100 HITs that are approved, and to have an approval rate above 80%. Overall, 58 different workers complete tasks. When a worker is hired for the first time, she is asked to select a set of keywords from a given list of keywords that capture her preferences. We create a unique *Worker Model* for all the 58 different workers that participate in our experiments.

When a worker first joins, we ask her to choose the top-5 keywords of her preference. We use these chosen keywords for a case study, shown in Section 4.5.6.

Ground Truth. For each micro-task, we record the ground-truth, which is the amount of time the worker spent on it in seconds. This is encoded in the task completion time vector for the corresponding *Worker Model*.

4.2 Implemented Algorithms

4.2.1 **Worker Model.** The linear model in Section 3.1 is implemented with a regularization parameter α. When implementing statistical models, this is a standard practice to avoid overfitting. The overall objective function thus becomes,

$$\min_{\vec{w}\in\mathbb{R}^m} \left\|y - \vec{w}^T \cdot \mathcal{T}\right\|_2 + \alpha \left\|\vec{w}\right\|_2^2 \tag{13}$$

The best value of α is chosen by generalized cross validation [23].

Model Initialization. We set a fixed budget b which we use to initialize the *Worker Model* iteratively (Section 2.2.1) and implement the following algorithms:

1. Random Initialization. `RandomInit` selects a randomly selected task iteratively, presents it to the worker and records the task completion time. The algorithm stops when the budget b is exhausted.

2. Active Initialization. `ActiveInit` implements our algorithm given in Section 3.1.1.

3. **Uniform Initialization.** `UniformInit` initializes the model by assigning uniform weights to the worker preference vector.

4.2.2 **Explicit Feedback.** This has two important components - one is the **Question Selector** that selects the task factors for explicit preference elicitation, the other is **Preference Aggregator** that updates the *Worker Model* using elicited preferences.

Question Selector. We have implemented two algorithms to find the best set of questions to ask as described in Section 3.2.

1. Optimization-Aware Question Selector. `k-ExFactor` is our proposed algorithm described in Section 3.2.

2. k**-random Question Selector.** `k-Random` is a simple baseline that randomly selects k-task factors for preference elicitation.

Preference Aggregator: This is our implemented solution for preference aggregation, as described in Section 3.3.

4.2.3 **Implicit Feedback.** We also implement implicit feedback computation to be compared against explicit feedback.

Algorithm `Implicit-1` is an adaption of recent work [24] that investigates how to implicitly capture worker motivation and use that for task assignment. While we do not necessarily focus on motivation as a factor in this work, we adapt the algorithm in [24] to estimate and update the worker preference vector over time. We do that by taking the average over the worker preference vector of the worker model obtained in different iterations. Since our focus is not on task assignment, once we estimate the worker preference vector using `Implicit-1`, we use that in conjunction with our *Worker Model* to predict a task completion time.

Algorithm `Implicit-2` is a further simplification. It relearns the *Worker Model* at the end of every iteration as the worker completes tasks and does not factor in the preference of the worker.

4.3 Invocation of `ExPref`

For quality experiments, `ExPref` is invoked iteratively and in an online fashion: in the beginning, we filter out the tasks and task completion history by worker id since the framework is personalized per worker. On average, a worker undertakes 200 tasks. We randomly divide the tasks into three subsets. We use 50% of each worker's data as a holdout over which error is computed. Half of the remaining tasks are used for training/developing the *Worker Model* and the rest as the pool of available tasks.

To conduct experiments only related to *Worker Model initialization* (specifically for "cold" workers), the training set is empty in the beginning and all tasks are in the available pool. We use the budget b to find a subset $\mathcal{B}$ of tasks based on our proposed solution in Equation 11.

After *Worker Model* is trained, in every iteration, we select a set x of 20 tasks (unless otherwise stated), randomly from the pool of available tasks and present them to the worker. After recording task completion time, we add those x tasks back to the training set. Next, we invoke the **Question Selector** that seeks explicit feedback from that worker. Upon receiving worker feedback, the *Worker Model* is updated using the **Preference Aggregator** and the new training set. We calculate the error over the holdout set after this. All these steps construe a single iteration of the `ExPref`.

For scalability experiments, we are only interested to measure the running time of the algorithms in `ExPref`. Thus, the experimental set up is rather simple there and we use the entire dataset.

Error. Unless otherwise specified, we calculate the quality of the *Worker Model* as the Mean Square Error (MSE) over validation set, defined as,

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\vec{w}^T\mathcal{T} - \vec{Y}\right)^2$$

Additionally, we present R^2 (co-efficient of determination) which indicates the proportion of the variance in the task completion time that is predictable from the task factors. R^2 takes values between $[-\infty, 1]$, where higher is better.

$$R^2 = 1 - \frac{\sum_t \left(y_t - \vec{w}^T \vec{t}\right)^2}{\sum_t \left(y_t - \bar{Y}\right)^2}$$

where, $\bar{Y}$ is the average task completion time.

Iteration. We define an iteration as the completion of a HIT (Human Intelligence Task) of 20 tasks, after which we compute the MSE and R^2 of the *Worker Model*.

Preference Elicitation. As described in Section 2.2.3, workers can provide their preference either as a full order, partial order, or they may not even provide any preference.

Preference History. For every worker, we also maintain her elicited preferences in all previous iterations (full history), preferences only in the current iteration (no history), or preferences in the last few iterations (partial history). *Worker Model* is updated accordingly.

4.4 Summary of Results

1. **Our proposed explicit preference elicitation framework outperforms (with statistical significance) existing implicit ones after fewer iterations.**

• We compare our approach ExPref with two other baseline algorithms Implicit-1 [24] and Implicit-2 (Section 4.5.1). We present MSE and R^2 with statistical significance results (standard error) and show that ExPref convincingly and significantly outperforms the other baselines under varying parameters : 1) Number of iterations (Figure 2), 2) Number of task factors (Figure 3), and 3) Number of tasks worker completes in each iteration (Figure 4).

• We compare the effect of different parameters of ExPref with appropriate baselines (Section 4.5.2). We show with a small number of questions k (Figure 5), k-ExFactor outperforms the baselines. Our results demonstrate that ActiveInit is an effective model initialization algorithm (Figure 8).

• Our results also indicate that task completion time is highly correlated to task outcome/quality of the completed task. This further justifies our investigation - indeed, deeper analysis of task completion time improves the quality of the crowdsourced tasks. Our case study results show that ExPref is capable to truly capture worker preference.

2. ExPref **is scalable.**

• We compare ExPref with other baselines under varying parameters: 1) Number of tasks, 2) Number of task factors, and 3) Number of questions (k). Unsurprisingly, ExPref is slower but it still scales very well.

• We compare our model initialization method ActiveInit by varying the budget b. ActiveInit is slower than the two other baselines. Despite that, it scales reasonably well. These results demonstrate the effectiveness of eliciting explicit preferences making ExPref usable in practice.

4.5 Quality Experiments

The objective of these experiments is to capture the effectiveness of our explicit feedback elicitation framework and compare it with appropriate baselines. Specifically, we are interested in answering the following questions :

1. How ExPref performs compared to implicit ones (Section 4.5.1).
2. Effect of different parameters in ExPref (Section 4.5.2).
3. Relationship between task completion time and task outcome (Section 4.5.4).
3. A case study on worker's explicit feedback (Section 4.5.5).
4. A case study on worker preferences (Section 4.5.6).

Table 2: Parameter Settings

Parameters	Range	Default
# tasks in each iteration (x)	5, 10, 15, 20, 25	20
# task factors (m)	5, 10, 25, 50, 80	80
# questions to ask (k)	3, 5, 7, 9	3
# iterations	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	7
initialization budget (b)	5, 15, 30, 50, 75	50

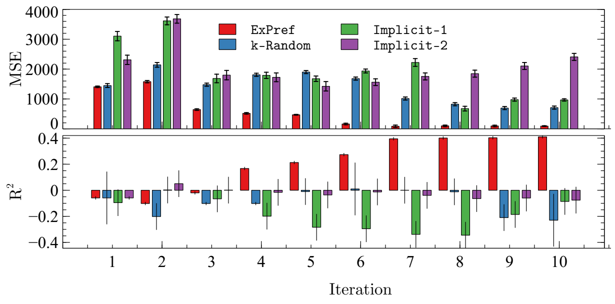


Figure 2: Comparison between the error of the 4 models after 10 iterations

Parameter Setting. For a given worker, there are four parameters to vary: 1) Number of tasks in each iteration (x), 2) Number of task factors (m), 3) Number of questions asked (k), and 4) Number of iterations. For initializing the model, we additionally vary the budget b. Table 2 presents the default values alongside the experimental settings for each parameter. To select a different number of task factors, the best m features are retained by finding factors that are highly correlated to the target and discarding the rest. By default, we always maintain the full history of worker's preference while updating the *Worker Model* under varying iterations.

4.5.1 ExPref **vs. Baselines**.

We compare two explicit solutions with two implicit ones. We vary # iterations, # task factors, and x (# tasks assigned to a worker after which the framework is invoked).

Varying the number of iterations.
Figure 2 presents the error of the 4 *Worker Models* in the course of 10 iterations. We notice that after the 7th iteration, all the four models become stable and their corresponding errors vary only by a very small margin. This means that ExPref can achieve significantly better results than the other three baselines after few iterations. Additionally, we observed that ExPref achieve stability in far fewer iterations than the baselines. This confirms the fact that worker's preference will be helpful to the model.

Varying the number of task factors.
In Figure 3, we observed that by adding more task factors, all the models perform better but ExPref performs significantly better. We also see that the number of task factors extracted from the tasks on our *Worker Model* is minimal compared to the other three baselines.

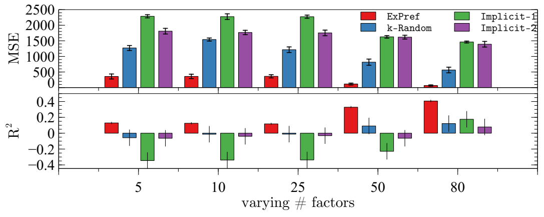


Figure 3: Error varying number of task factors

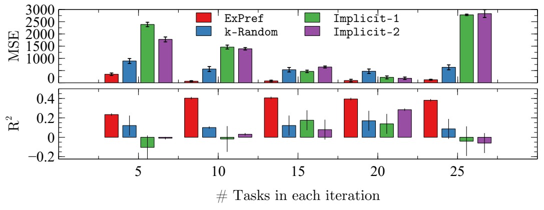


Figure 4: Error varying number of tasks in each iteration

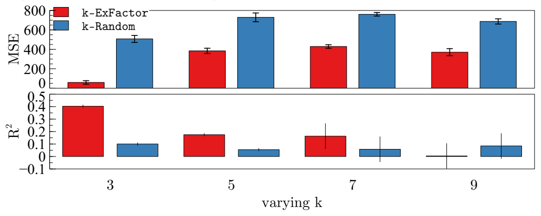


Figure 5: Error varying number of questions

Varying the number of tasks in each iteration.
Figure 4 presents the results for varying the number of tasks a worker receives in each iteration (denoted by x). ExPref outperforms the other three baseline by a large margin in terms of achieving smaller error. Notice that for $x = 20$ all the algorithms perform well but ExPref is better that the other three. For this reason, we set the default value of x to be 20.

Varying the number of questions.
Since the results of Implicit-1 and Implicit-2 do not change with k, we present the results in the next section (Section 4.5.2).

4.5.2 Effect of Different Parameters.

We do a comparative study on the effect of different parameters, namely, how we track worker's history (recent history vs full history vs partial history), the number of questions we ask a worker (k), and the budget we use for model initialization (b).

Number of explicit questions to ask.
As the number of questions increases, the quality of the *Worker Model* decreases (Figure 5). This happens for two reasons. First, when the number of questions is higher than 5, worker responses are inconsistent. Second, since we keep the full history of responses for the worker, the number of constraints imposed in our optimization problem grows significantly which in turn affects performance.

Similarly, as we ask more questions from workers, most of the answers provided are in the form of partial ranking rather than full ranking. It's likely that the workers simply picked the most important factors and ignored the rest.

Full, partial, no history.
We present a comparative study between fully or partially capturing the history of the worker's preferences versus no history, i.e., using the most recent preferences only. To better understand the difference between the three, as an example, consider we ask 4 explicit questions to a worker in each iteration, after the third

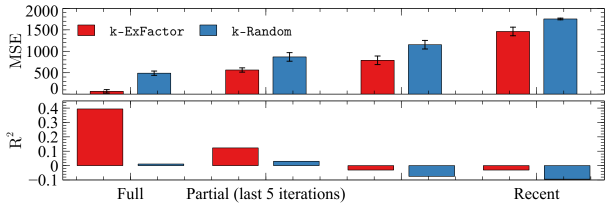


Figure 6: Recent history vs partial history vs full history

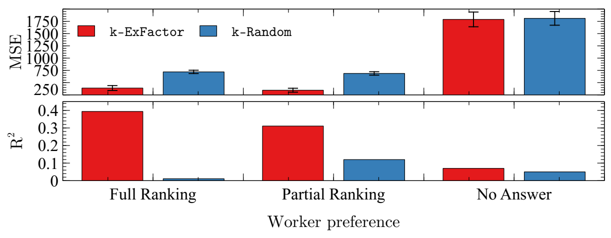


Figure 7: Error varying worker preference type

iteration, her full history size is 12, whereas, her most recent history size is 4; i.e., the recent history represents the number of feedback in the current iteration. Assuming an expiration time of 2, the partial history is defined as the preference of the worker based on the last 2 iterations which is of size 8. Figure 6 presents the results of k-ExFactor against k-Random in the last iteration for four scenarios. We omit the results for the other two implicit models since their results is not affected by the worker's history. Clearly, maintaining a full history for the worker performs significantly better. Intuitively, this shows that the model can understand the worker better when all the information about her is maintained. This means that the *Worker Model* can handle variations in worker's behavior significantly better compared to using recent information only.

Full, partial, no order of preference.
Figure 7 presents the results for different types of feedback a worker can provide. We set the number of questions that we ask in each iteration to $k = 4$. Notice that when the worker does not provide any answer, the results are very similar to Implicit-2. This means that if the worker does not provide any answer, we fall back to the implicit model. Similarly, note that the performance of the *Worker Model* is not affected by the type of answer a worker provides. This means that as long as the worker provides some feedback, albeit partially, our **Preference Aggregator** can help the *Worker Model* better estimate the task completion time.

4.5.3 Model Initialization.

Figure 8 presents the difference in the reconstruction error between the three initialization methods. ActiveInit performs better overall and for $b = 50$ it has the lowest reconstruction error between the three methods. Notice that as the size of initialization set grows, the reconstruction error drops. This is attributed to the fact that the *Worker Model* needs a reasonable amount of training data to be able to predict the task completion duration. Another important phenomena is that increasing b beyond 50 results in an increase in the reconstruction error. This is because the *Worker Model* will overfit the training data and lose its predictive power.

4.5.4 Task Completion Time vs. Task Outcome.

We notice that completed tasks that have correct answers take more time

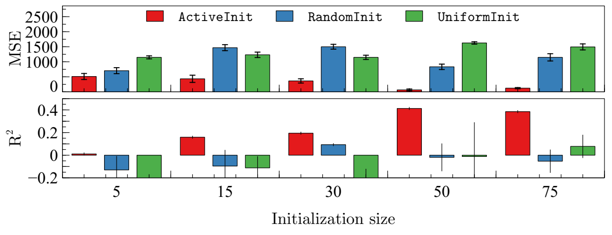


Figure 8: Evaluation of model initialization algorithms

Worker no	Worker Keywords	Top-2 preference
1	dress,google street view, airlines, classification, scene	dress, scene
2	business, body parts, google street view, health, classification	classification, google street view
3	image, south Asia, disease, animals, text	image, text

Table 3: Worker keywords and preference questions

on average to complete than the tasks that are not. Using *Chi-squared test*, we observe a high positive correlation between task completion time and its outcome/quality (with $\chi^2 = 3796.99$ and $p - value = 0.00001$). This in fact is one of the motivations behind our study, as the correct estimation of task completion time will help us better understand task outcome.

4.5.5 **Worker's Feedback.** We profile all 58 workers over the course of their participation and notice that they always provide some feedback (partial/full). We notice that if the number of questions is less than 4, the workers are more likely to provide full feedback. As we increase the number of questions, worker's tend to give partial feedback.

4.5.6 **A Case Study.** We profile three workers randomly from our database and analyze their models in conjunction with the keywords they have initially chosen. Table 3 presents the 5 keywords chosen by the workers and the top-2 worker preferences. It is easy to notice that they are highly correlated, which shows that our proposed model successfully captures worker preference.

4.6 Scalability Experiments

We are interested in answering the following questions :
1. How `ExPref` scales compared to implicit preference computation (Section 4.6.1).
2. Effect of different parameters in `ExPref` (Section 4.6.2).
3. Time Profile of each component of `ExPref` (Section 4.6.3).

Unless otherwise stated, we report running times in seconds.

Parameter Setting. Our dataset contains 165, 168 tasks and 80 task factors obtained from 58 workers. In these experiments, we vary the following parameters: # tasks, # task factors, k, and the initialization budget b. Unless otherwise stated, all the numbers present the average running time of a single iteration over all the 58 workers. The default values are set as # tasks = 30, 000, # task factors = 50, $k = 3$, and $b = 20$. By default, we consider full order of worker preference since that adds more constraints to the problem. For the *Worker Model* initialization comparison, only the appropriate three methods are compared.

4.6.1 **Efficiency of `ExPref` vs. Baselines**. Figure 9(a) presents the running times of the four algorithms with varying number of tasks. Of course, our proposed solution `k-ExFactor` makes a lot more computation to ensure optimization and hence has the highest running time. However, it is easy to notice that with an increasing number of tasks, it scales well and the running time is comparable to the other competing algorithms. A similar observation holds when we vary the number of task factors, as shown in Figure 9(b). `k-ExFactor` scales well and never takes more than 80 seconds.

4.6.2 **Effect of Parameters on Efficiency.** Figure 9(c) represents the running times by varying k, the number of task factors chosen for preference elicitation. Here only `k-ExFactor` is compared with `k-Random`, as the other two algorithms do not rely on explicit preference elicitation. Unsurprisingly, `k-Random` is faster, but our proposed solution `k-ExFactor` scales well and has a comparable running time. Finally, in Figure 9(d), we vary the initialization sample size and present the running time of `ActiveInit`. Our initialization model scales well and does not take much time as the size of initialization set grows. The other two baselines do not perform any computation and take negligible time to terminate.

4.6.3 **Profiling `ExPref` for Efficiency**. We further profile the individual running time of `ExPref` with the default settings; i.e., # tasks = 30, 000, # task factors= 50, $k = 3$. It takes 8 seconds to train the *Worker Model*, 6.15 seconds to solve **Question Selector** that finds the best k factors, and 9.1 seconds to run **Preference Aggregation** that updates the *Worker Model* with the added constraints. These results demonstrate that the individual components of the framework take comparable time.

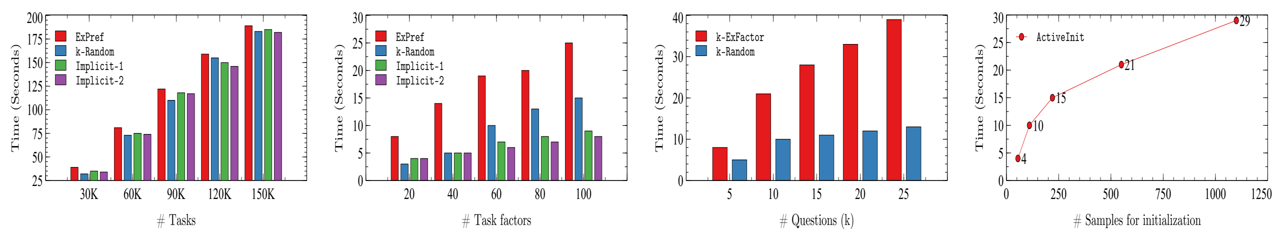
5 RELATED WORK

The related work can be classified into three categories: preference elicitation from the crowd, leveraging worker preferences in crowdsourcing processes, and worker models.

Preference Elicitation. In [7, 15, 25], the crowd was solicited to perform max/top-k and clustering operations with the assumption that workers may make errors. These papers study the relationship between the number of comparisons needed and error. Efficient algorithms are proposed with a guarantee to achieve correct results with high probability. A similar problem was addressed in [14] in the case of a skyline evaluation. In that setting, it is assumed that items can only be compared through noisy comparisons provided by the crowd and the goal is to minimize the number of comparisons. A recent work studies the problem of computing the all pair distance graph [22] by relying on noisy human workers. The authors addressed the challenge of how to aggregate those feedback and what additional feedback to solicit from the crowd to improve other estimated distances.

While we also rely on inputs from the crowd, the elicited input represents each worker's preference for different factors (as opposed to completing actual tasks), and is hence not assumed to be noisy or erroneous. However, as worker preferences evolve over time, we propose an iterative approach with the goal of improving task completion time overall.

Leveraging Preferences. Worker preferences for task factors are heavily leveraged in all crowdsourcing processes. Very few of these efforts focused on leveraging them in *task completion* [4, 6, 29]. Authors of [20] investigated 13 worker *motivation* factors and found



(a) varying # tasks (b) varying # task factors (c) varying k (d) varying b

Figure 9: Scalability study

that workers were interested in *skill variety* or *task autonomy* as much as *task reward*. Chandler and Kapelner [4] empirically showed that workers *perceived meaningfulness* of a task improved throughput without degrading quality. Shaw et al. [29] assessed 14 incentives schemes and found that incentives based on *worker-to-worker comparisons* yield better crowd work quality. Hata et al. [16] studied worker *fatigue* and it affects how work quality over extended periods of time. Other efforts focused on gradually increasing pay during task completion to improve worker retention [13]. Lately, adaptive task assignment were studied with a particular focus on maximizing the quality of crowdwork [11, 17, 18, 24] but primary for improved task assignment.

Existing work showed the importance of leveraging implicit worker preferences for task assignment. In contrast, we show explicit elicitation of worker preferences results in a more accurate model that leads to better estimation of task completion time.

6 CONCLUSION

We present a framework ExPref for eliciting explicit worker’s preference for task completion time in crowdsourcing platforms by developing and maintaining a personalized *Worker Model*. Around this model, we define two core optimization problems; **Question Selector** that selects the best set of questions to obtain a worker’s preference in the form of a full/partial ranking of task factors, and **Preference Aggregator** that updates the worker model with provided preferences. We present theoretical results showing the hardness of our problems and algorithms with theoretical guarantees. We conduct large-scale experiments with 165, 168 tasks from CrowdFlower involving 58 workers hired from Amazon Mechanical Turk. Our quality experiments corroborate the necessity of explicit preference elicitation by comparing that with state of the art implicit preference computation. Our scalability results demonstrate that our framework is practical and could be used in real crowdsourcing platforms. As an ongoing work, we are investigating how to adapt this problem, when the explicit feedback is erroneous or has bias due to malicious user behavior.

ACKNOWLEDGMENTS

The work of Mohammadreza Esfandiari and Senjuti Basu Roy are supported by the National Science Foundation under Grant No.: 1814595.

REFERENCES

[1] Sihem Amer-Yahia and Senjuti Basu Roy. 2016. Human factors in crowdsourcing. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1615–1618.
[2] Haim Avron and Christos Boutsidis. 2013. Faster subset selection for matrices and applications. *SIAM J. Matrix Anal. Appl.* 34, 4 (2013), 1464–1499.
[3] Sampoorna Biswas et al. 2017. Combating the Cold Start User Problem in Model Based Collaborative Filtering. *CoRR* abs/1703.00397 (2017). http://arxiv.org/abs/1703.00397
[4] Dana Chandler and Adam Kapelner. 2012. Breaking Monotony with Meaning: Motivation in Crowdsourcing Markets. *CoRR* abs/1210.0962 (2012).
[5] David A Cohn et al. 1996. Active learning with statistical models. *Journal of artificial intelligence research* (1996).
[6] Peng Dai et al. 2015. And Now for Something Completely Different: Improving Crowdsourcing Workflows with Micro-Diversions. In *ACM CSCW*.
[7] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. 2014. Top-k and Clustering with Noisy Comparisons. *ACM TODS* 39, 4 (2014), 35:1–35:39.
[8] FR De Hoog and RMM Mattheij. 2007. Subset selection for matrices. *Linear Algebra Appl.* 422, 2-3 (2007), 349–359.
[9] Mohammadreza Esfandiari, Kavan Bharat Patel, Sihem Amer-Yahia, and Senjuti Basu Roy. 2018. Crowdsourcing Analytics With CrowdCur. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 1701–1704.
[10] Ludwig Fahrmeir and Gerhard Tutz. 2013. *Multivariate statistical modelling based on generalized linear models*. Springer Science & Business Media.
[11] Ju Fan et al. 2015. iCrowd: An Adaptive Crowdsourcing Framework. In *SIGMOD*.
[12] Siamak Faradani et al. 2011. What’s the Right Price? Pricing Tasks for Finishing on Time. In *AAAI*.
[13] Yihan Gao et al. 2014. Finish Them!: Pricing Algorithms for Human Computation. *PVLDB* (2014).
[14] Benoît Groz and Tova Milo. 2015. Skyline Queries with Noisy Comparisons. In *PODS*. 185–198.
[15] Stephen Guo, Aditya G. Parameswaran, and Hector Garcia-Molina. 2012. So who won?: dynamic max discovery with the crowd. In *SIGMOD 2012*. 385–396.
[16] Kenji Hata et al. 2017. A Glimpse Far into the Future: Understanding Long-term Crowd Worker Quality. In *CSCW*.
[17] Chien-Ju Ho et al. 2013. Adaptive Task Assignment for Crowdsourced Classification. In *ICML*.
[18] Chien-Ju Ho and Jennifer Wortman Vaughan. 2012. Online Task Assignment in Crowdsourcing Markets. In *AAAI*.
[19] Ece Kamar et al. 2016. Intervention strategies for increasing engagement in crowdsourcing: Platform, predictions, and experiments. (2016).
[20] Nicolas Kaufmann, Thimo Schulze, and Daniel Veit. 2011. More than fun and money. Worker Motivation in Crowdsourcing-A Study on Mechanical Turk.. In *AMCIS*, Vol. 11. 1–11.
[21] Andrew Mao et al. 2013. Why stop now? predicting worker engagement in online crowdsourcing. In *HCOMP*.
[22] Volker Markl, Salvatore Orlando, Bernhard Mitschang, Periklis Andritsos, Kai-Uwe Sattler, and Sebastian Breß (Eds.). 2017. *EDBT*.
[23] Jodi L Mead and Rosemary A Renaut. 2010. Least squares problems with inequality constraints as quadratic constraints. *Linear Algebra Appl.* (2010).
[24] Julien Pilourdault et al. 2017. Motivation-Aware Task Assignment in Crowdsourcing. In *EDBT*.
[25] Vassilis Polychronopoulos, Luca de Alfaro, James Davis, Hector Garcia-Molina, and Neoklis Polyzotis. 2013. Human-Powered Top-k Lists. In *WebDB*. 25–30.
[26] Friedrich Pukelsheim. 2006. *Optimal design of experiments*. SIAM.
[27] Senjuti Basu Roy et al. 2013. Crowds, not Drones: Modeling Human Factors in Interactive Crowdsourcing. In *DBCrowd*. 39–42.
[28] Senjuti Basu Roy et al. 2015. Task assignment optimization in knowledge-intensive crowdsourcing. *VLDB J.* (2015).
[29] Aaron D. Shaw et al. 2011. Designing incentives for inexpert human raters. In *CSCW*.
[30] Philip B Stark and Robert L Parker. 1995. Bounded-variable least-squares: an algorithm and applications. *Computational Statistics* 10 (1995), 129–129.