Filtering With the Crowd: CrowdScreen Revisited

Benoît Groz[1], Ezra Levin[2], Isaac Meilijson[3], and Tova Milo[4]

1 **Université Paris Sud 11, Orsay, France**
`benoit.groz@lri.fr`
2 **Tel Aviv University, Tel Aviv, Israel**
`ezralevin@gmail.com`
3 **Tel Aviv University, Tel Aviv, Israel**
`isaco@post.tau.ac.il`
4 **Tel Aviv University, Tel Aviv, Israel**
`milo@cs.tau.ac.il`

Abstract

Filtering a set of items, based on a set of properties that can be verified by humans, is a common application of CrowdSourcing. When the workers are error-prone, each item is presented to multiple users, to limit the probability of misclassification. Since the Crowd is a relatively expensive resource, minimizing the number of questions per item may naturally result in big savings. Several algorithms to address this minimization problem have been presented in the CrowdScreen framework by Parameswaran et al. However, those algorithms do not scale well and therefore cannot be used in scenarios where high accuracy is required in spite of high user error rates. The goal of this paper is thus to devise algorithms that can cope with such situations. To achieve this, we provide new theoretical insights to the problem, then use them to develop a new efficient algorithm. We also propose novel optimizations for the algorithms of CrowdScreen that improve their scalability. We complement our theoretical study by an experimental evaluation of the algorithms on a large set of synthetic parameters as well as real-life crowdsourcing scenarios, demonstrating the advantages of our solution.

1998 ACM Subject Classification H.3.3 Information filtering

Keywords and phrases CrowdSourcing, filtering, algorithms, sprt, hypothesis testing.

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.12

1 Introduction

CrowdSourcing for Filtering

Building upon a flourishing ecosystem of CrowdSourcing platforms, a new kind of database systems such as CrowdDB and Qurk endeavors to exploit human inputs to extract or process information [20, 8]. Queries in these systems rely on a small set of basic operators to elicit missing information from the crowd. This triggered a new line of research devoted to the optimization of such basic operations as Joins, Ordering, Aggregates, Selection, etc., in a CrowdSourcing environment [19, 18]. In this paper we focus on the Selection operation, i.e., using the crowd to filter the items satisfying some specific property.

As an example, assume we are sensitive to gluten and would like to know which food items, out of a given list or a menu, may be problematic for us. Scanning food recipes and labels could give information on each individual item, but this is a time consuming job, and the results may be incorrect, e.g. due to some ignored factors such as cross-contamination issues. Asking the Crowd about their knowledge/experience with the product may provide an alternative solution to the problem. However, contributors will sometimes provide erroneous answers, so that multiple answers must be gathered in order to ascertain that a product is gluten-free. But how many people need to be asked? Let us assume that (1) the probability that each category of food contains the ingredients, and (2) the error rates among the answers (false positives and false negatives rates) are prior knowledge – we will briefly discuss this assumption later. For example, suppose that some category of dishes, e.g. cereal, contains gluten with probability 0.5, and suppose the probability of false positives and false negatives are both 0.4. How can we decide with an average precision of 90% whether or not a given cereal contains some gluten, and how many answers will be required for that?

A simple solution is to fix in advance some budget m for answers, and then decide that our dish contains (resp. does not contain) gluten as soon as we get more than $m/2$ positive (negative) answers. For our parameters, one can easily check that a budget of $m = 41$ answers is required to obtain 90% precision with this strategy. Naturally, we do not always have to use the full budget – as soon as 21 positive (or negative) answers are obtained we can stop and make a decision with the required precision. We will thus ask between 21 and 41 questions and, on average, about 34 questions (we omit the exact computation). Note however that a smaller average number of questions can be used if we employ a more efficient strategy, known in the literature as a *sequential test* [23], and adapt dynamically the budget as answers are received. We can for instance show that on average only 23 answers are sufficient to reach a decision if we use the following strategy which also guarantees an average precision of 90%:

- claim there is gluten as soon as the number of positive answers exceeds the number of negative answers by 6
- claim there is none when negative answers exceed positive answers by 6
- use majority vote in the absence of conclusion after 51 questions

More generally, the challenge that we try to address in this paper is devising tests that minimize the expected number of answers required from the crowd for deciding whether a given object satisfies a selection criteria, while guaranteeing that the average error stays below the required threshold.

The CrowdScreen Framework

The problem of minimizing the number of questions needed to classify items accurately clearly predates CrowdSourcing and we discuss related work in the conclusion. Yet, CrowdSourcing scenarios may be particular in the sense that errors on specific items are typically tolerated as long as a good accuracy is guaranteed on average over the whole set of items [22].

To study the optimization of filtering, we adopt in this paper a simple and general model by Parameswaran et al. [22], whose purpose is to compute optimal querying strategies. They define a (deterministic) strategy as a function mapping the number of positive and negative answers received from the users to a decision in $\{\texttt{Pass}, \texttt{Fail}, \texttt{Cont}\}$. A `Pass` (resp. `Fail`) decision signifies we stop asking questions and accept the object in question as satisfying the filter (resp. reject the object), and `Cont` stands for asking additional questions. They also consider probabilistic strategies that map each point to both a probability to stop asking questions and the decision (`Pass` or `Fail`) in case the strategy terminates at this point. Questions to the Crowd are considered expensive and therefore the maximal number of questions allotted to the strategy is bounded by some fixed budget. The selectivity of the filter and the error rates of the answers, as previously mentioned, are considered prior knowledge in the model. A problem instance thus consists of a budget bound, a maximal bound on the expected error authorized for the strategy, and those prior probabilities.

Several algorithms and heuristics have been introduced in [22] to compute deterministic and probabilistic strategies. While these algorithms are efficient for very small budgets (up to 14 questions per item), larger sample sizes were hardly considered. In fact, the presented algorithms are not a good fit for larger budget as they either have high complexity (exponential, or polynomial but with high degree), or suffer from numeric instability, hence do not always return a strategy meeting the error constraint.

The restriction to small budgets may be justified by the assumption that CrowdSourcing applications typically use little redundancy. Yet 14 questions are not sufficient to filter items with a high precision when the error rates are high: our motivating example for instance requires more than 40 questions, and the original works about sequential tests in statistical testing [23, 2] generally consider budgets featuring hundreds or thousands of answers. *The goal of this paper is thus to devise algorithms that scale well for large budgets.*

Contributions

Our contributions are three-fold. First, we provide new theoretical insights to the problem. We then devise efficient algorithms based on these insights. We also propose optimizations of algorithms in [22] to improve their scalability.

Specifically, we first show, in Section 2, that key properties of the problem derive from well-known results on the *likelihood ratio* (to be formally defined). We exploit these in Section 3 to devise a scalable algorithm: `AdaptSprt` inspired from the popular SPRT [23]. We then revisit, in Section 4, the heuristics from [22]. In particular, we show that their method that enumerates all (ladder-shaped) strategies has complexity $O(2^{2m})$, and we present optimizations extending the range of budgets for which this enumeration is tractable by a factor $\approxeq 1.5$. We similarly show that the `shrink` heuristic from [22], which computes slightly suboptimal deterministic strategies, can be optimized to run in $O(m^4)$ instead of $O(m^5)$, and further establish, in Section 5, connections between deterministic and probabilistic strategies. In particular we show that an optimal probabilistic strategy can be computed through a minor modification to the `shrink` strategy, as an alternative to the linear programming approach that was considered there (and which we show to suffer from numeric instability). For space constraints we defer some of the proofs to the technical report [11]. Finally, to complement our theoretical study we briefly illustrate (with more details in the technical report) the practical advantages and limitations of our solutions by a set of experiments on (1) a large set of synthetic parameters and (2) a small real-life scenario.

2 Preliminaries

We first list definitions and notations, as well as general properties we use to devise efficient strategies. The formal introduction below follows [22] and we diverge afterwards.

2.1 Definitions

We wish to harness the wisdom of the crowd to determine, for each object O of a large dataset D, whether the object has property V ($V = 1$) or not ($V = 0$). We thus ask users in the crowd if they believe the object has the property. To compensate for possible mistakes, we query multiple users until we have gathered enough evidence to reach a pass/fail decision about O. The selectivity ratio s (percentage of objects in D having property V) and the users' error rates are assumed prior knowledge. We thus define the error rates e_0 and $e_1 < 0.5$ as the probability of a user error, given that $V = 0$ and $V = 1$ respectively. We briefly discuss in the conclusion how these values can be estimated. Finally, we consider that each question has a unit cost, and specify a *budget constraint* m; the maximal number of questions we are allowed to ask before reaching a decision on item O.

Strategies

The sequence of answers received when classifying an item can be visualized as a walk on a discrete 2-dimensional grid where the x and y axes represent the number of negative and positive answers received. The current state of the sequence is the point (x, y) matching the number of positive and negative answers received. The transitions between states match the answers provided by the users: if a negative answer is received in state (x, y), the state moves to $(x + 1, y)$. If a positive answer is received instead, the state moves to $(x, y + 1)$. For each point on the grid we define $P_{stop}(x, y)$ as the probability that the walk terminates upon reaching point (x, y). When terminating, a claim on the value of V is returned: `Pass` $(V = 1)$ or `Fail` $(V = 0)$.

A strategy is defined by the function $P_{stop}(x, y)$ mapping each point to the probability of terminating when reaching point (x, y). In a probabilistic strategy $P_{stop}(x, y)$ is taken over the interval $[0, 1]$, whereas in a deterministic strategy $P_{stop}(x, y)$ must be either 0 or 1. Point (x, y) is a *continuing point* if $P_{stop}(x, y) = 0$, a *terminating point* if $P_{stop}(x, y) = 1$, and a *probabilistic* point otherwise. An optimal choice betweeb `Pass` or `Fail` in case we stop can easily be computed from x, y and the input parameters, using a well-known property of the likelihood ratio recalled in Section 2.2 (the choice does not depend on the strategy). A point in which the decision is `Pass` is an *accepting point* and a point in which the decision is `Fail` is a *rejecting point.* The *cost* of a given strategy is the expected number of answers needed in order to reach a decision, while the *error* of the strategy is the probability that the strategy reaches a wrong decision. We formalize this next.

Strategy and Grid characteristics

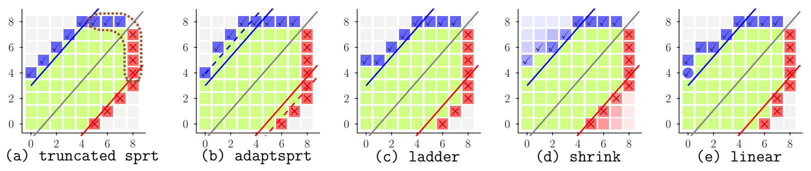
We compute the cost and error of a strategy as mentioned in [22]. Intuitively, our equations first count paths leading to (x, y) according to the strategy, then multiply the result (*Path*) by the probability (S_0, S_1) that answers follow any single such path. Let $S_i(x, y)$ be the probability that by the time we have asked $x + y$ queries we receive (in any specific order) x negative answers and y positive answers and have $V = i$. We then have:

$$S_0(x, y) = (1 - s) \times (1 - e_0)^x \times e_0^y \tag{1}$$

$$S_1(x, y) = s \times e_1^x \times (1 - e_1)^y \tag{2}$$

Let $Path(x, y)$ denote the weighted number of paths (i.e., sequences of answers) that consist of y positive and x negative answers, each path being weighted by $(1-p)$ where p is the probability (depending on the path and the strategy) to stop along the path before reaching point (x, y). We partition these paths into two groups: $Path(x, y) = tPath(x, y) + cPath(x, y)$ with $tPath(x, y) = P_{stop}(x, y) \times Path(x, y)$. Thus, $tPath(x, y)$ and $cPath(x, y)$ respectively count the paths that terminate and continue after reaching point (x, y). We observe that the strategy P_{stop} uniquely determines the values of *tPath* and *cPath*, and reciprocally. A point (x, y) is *reachable* if $Path(x, y) > 0$.

▶ **Example 1.** The running example illustrates the definitions along the paper with error rates $e_0 = .25$ and $e_1 = .2$, threshold $\tau = .0075$, budget $m = 15$, and selectivity $s = .8$. Figure 1 pictures the strategies returned for those parameters by the algorithms investigated



algorithm	cost	error
SPRT	6.94	0.008
`AdaptSprt`	7.748	0.00741
`ladder`	7.59	0.00749
`shrink`	7.73	0.00748
`linear`	7.56	0.0075

Figure 1 Strategies returned for $e_0 = .25$, $e_1 = .2$, $\tau = .0075$, $m = 15$, and $s = .8$.

in this paper: unreachable, accepting, rejecting, and continuing points are represented as white, blue (with a checkmark), red (with cross), and green squares respectively. Probabilistic points are circles, with similar colors (and marks). Other signs in the figure will be discussed later on.

We further define $g_i(x, y)$ as the probability that $V = i$ and the point (x, y) is ever reached for $i = 0, 1$. This value is computed as: $g_i(x, y) = Path(x, y) \times S_i(x, y)$. Let $\mathrm{Err}(x, y)$ denote the probability of error when making a decision at point (x, y) (we detail in the next section how to calculate $\mathrm{Err}(x, y)$). The probability that we reach (x, y) and stop there is $\sum_{(x,y)}(g_0(x, y) + g_1(x, y)) \times P_{stop}(x, y)$. The cost of a strategy is therefore:

$$C = \sum_{(x,y)} (g_0(x, y) + g_1(x, y)) \times P_{stop}(x, y) \times (x + y)$$

and the error of the strategy is:

$$E = \sum_{(x,y)} (g_0(x, y) + g_1(x, y)) \times P_{stop}(x, y) \times \mathrm{Err}(x, y)$$

The Problem Definition

The error threshold τ fixes the maximal error a strategy is allowed. A strategy is *feasible* if it satisfies the budget and error constraints m and τ, and *optimal* if it has minimal cost among feasible strategies. Our objective is to find the optimal strategy, given the priors e_0, e_1, s and the constraints m and τ.

Optimal Stopping Problem
Input: selectivity s, error threshold τ, error rates e_0, e_1 and budget m
Question: find a feasible strategy that minimizes C

The strategies (and grids) that we consider satisfy certain constraints, enumerated below. The objective is to minimize the cost C under the following constraints:

1. There is exactly one path going through the origin : $cPath(0, 0) + tPath(0, 0) = 1$
2. Conservation of paths: the weighted number of paths reaching point (x, y) is equal to the number of paths that continue through its predecessors $(x - 1, y)$ and $(x, y - 1)$: $Path(x, y) = cPath(x - 1, y) + cPath(x, y - 1)$
3. All strategies are limited to m queries: $\forall(x, y), x + y = m \implies cPath(x, y) = 0$
4. The error rate of the strategy is at most τ:

$$E = \sum_{(x,y):x+y\leq m} tPath(x, y) \times \min(S_0(x, y), S_1(x, y)) \leq \tau$$

Up till now, the introduction followed the definitions and equations of [22], but the remainder of this section presents useful properties of strategies from a different perspective.

2.2 General Framework

The probability that $V = i$ given that (x, y) has been reached is given by: $g_i(x,y)/(g_0(x,y)+g_1(x,y)) = 1/(1+(g_{(1-i)}(x,y)/g_i(x,y)))$, where $g_i(x,y)$, as previously defined, is the probability that $V = i$ and point (x,y) is reached for $i = 0, 1$. The error committed when making a decision at (x, y) is therefore:

$$\text{Err}(x,y) = \begin{cases} \frac{1}{1+g_1(x,y)/g_0(x,y)} & \text{if the decision at } (x,y) \text{ is } \texttt{Pass} \\ \frac{1}{1+g_0(x,y)/g_1(x,y)} & \text{if the decision at } (x,y) \text{ is } \texttt{Fail} \end{cases} \tag{3}$$

To minimize error, a strategy should therefore opt for `Pass` if $g_1(x,y)/g_0(x,y) > 1$, and `Fail` if $g_1(x,y)/g_0(x,y) < 1$. The decision has no impact on error when $g_1(x,y) = g_0(x,y)$. We henceforth assume that all strategies adopt this decision rule since it minimizes error and has no impact on the cost. The decision to accept or reject thus only depends on the value of the *likelihood ratio* $g_1(x,y)/g_0(x,y)$, which can be computed from x, y, and the parameters independently from the strategy. The following equation further details the location of accepting and rejecting points, and as such refines the property presented as the *path principle* in [22]:

$$\begin{aligned} \log \frac{g_1(x,y)}{g_0(x,y)} &= \log\left(\frac{s}{1-s} \times \left(\frac{e_1}{1-e_0}\right)^x \times \left(\frac{1-e_1}{e_0}\right)^y\right) \\ &= \log\frac{s}{1-s} + x\log\left(\frac{e_1}{1-e_0}\right) + y\log\left(\frac{1-e_1}{e_0}\right) \end{aligned}$$

▶ Remark. The contour lines for the likelihood ratio (i.e., the set of points with likelihood ratio $g_1(x,y)/g_0(x,y) = c$ for some constant c) form a straight line on the grid, and all contour lines are parallel. Furthermore $e_0, e_1 < 1/2$ so the ratio increases strictly with y and decreases with x.

We call the line $(g_1(x,y)/g_0(x,y)) = 1$ the *decision line*. Points above this line satisfy $1 < g_1(x,y)/g_0(x,y)$ and are therefore accepting, while points below the line are rejecting.

▶ **Example 2.** For the running example with $e_0 = .25$, $e_1 = .2$, $\tau = .0075$, $m = 15$ and $s = .8$, the decision line has equation: $y = -\frac{\log(.2/.75)}{\log(.8/.25)}x - \frac{\log(4)}{\log(.8/.25)} \approxeq 1.11 \times x - 2$. This line is depicted in grey on all the grids in Figure 1.

2.3 Simple Optimizations

Before presenting the algorithms we describe three basic optimizations that they all employ. The first is borrowed from [22] and the other two are new.

Ladder Strategies

Parameswaran et al. [22] prove that under reasonable assumptions, all optimal strategies have a particular shape. They define a *ladder strategy* as a strategy whose terminating points can be partitioned into two converging sequences: the *upper ladder* and the *lower ladder*. The points of an upper ladder are given by a non-decreasing mapping from x to y whereas the lower ladder is a non-decreasing mapping from y to x. Furthermore, the points of the upper ladder stay above the decision line, whereas those of the lower ladder stay below. For example, all deterministic strategies represented in Figure 1 (i.e., a,b,c, and d) are ladder strategies. It has been conjectured in [22] that any optimal strategy is a ladder strategy. We adopt this conjecture and focus in this paper on ladder strategies.

Pruning the Grid

Let $(x_{\mathtt{dec}}, y_{\mathtt{dec}})$ denote the point at which the decision line and $x + y = m + 1$ intersect, i.e., the unique point such that $x + y = m + 1$, $(x - 1, y)$ is accepting and $(x, y - 1)$ rejecting. All points with $x = x_{\mathtt{dec}}$ or $y = y_{\mathtt{dec}}$ are terminating in any optimal strategy, since all points reachable from (x, y) return the same decision (e.g., `Pass`) so that continuing asking questions from (x, y) is pointless.

▶ **Example 3.** In the running example, the budget bounds $x + y$ by 15, so that $(x_{\mathtt{dec}}, y_{\mathtt{dec}}) = (8, 8)$. All strategies presented in Figure 1 are therefore restricted to $x, y \leq 8$.

Deciding Feasibility

A problem instance admits a feasible strategy (strategy meeting the error and budget constraints) if and only if the rectangular strategy $\sigma_{\mathtt{rect}}(x_{\mathtt{dec}}, y_{\mathtt{dec}})$ with terminating points only along $x = x_{\mathtt{dec}}$ and $y = y_{\mathtt{dec}}$ is feasible. Point $(x_{\mathtt{dec}}, y_{\mathtt{dec}})$ is obtained in constant time as the intersection of two lines: the decision line and the line $x + y = m$. The error of $\sigma_{\mathtt{rect}}(x_{\mathtt{dec}}, y_{\mathtt{dec}})$ can thus be computed as $B(e_0; y_{\mathtt{dec}}, x_{\mathtt{dec}}) + B(e_1; x_{\mathtt{dec}}, y_{\mathtt{dec}})$, where B denotes the incomplete beta function [7], incorporated in standard numeric libraries. One can thus decide feasibility in constant time for all practical purposes, and we therefore only consider feasible problems from now on.

3 Likelihood Ratio Test

The first solution we introduce is based on the Sequential Probability Ratio Test (SPRT), defined by Wald [23] in the context of quality control. As it may return strategies with unbounded budgets, we also consider its truncated variant which limits the budget but may exceed the error constraint. We finally propose an adapted version of SPRT to accommodate both budget and error constraints.

3.1 SPRT: Definition and Boundaries

General SPRT: Infinite Budget

The SPRT strategy defined by Wald [23] is the strategy that continues asking questions until the likelihood ratio (defined in Section 2.2) leaves interval $]\alpha, \beta[$, where α and β depend on the error we are willing to tolerate under $V = 0$ and $V = 1$. To continue asking questions until reaching a point with $\mathrm{Err}(x, y) \leq \tau$, we thus set $\alpha = \frac{\tau}{1-\tau}$, $\beta = \frac{1-\tau}{\tau}$. The error in each decision point (hence the overall error of the strategy) is bounded by τ. As a corollary of Remark 2.2, grid points where $\mathrm{Err}(x, y) > \tau$ are bound by two parallel lines, so the continuing points of the SPRT strategy are the points located between these SPRT lines, characterized in the following Proposition:

▶ **Proposition 4.** *A point (x, y) satisfies $Err(x, y) \leq \tau$ if and only if $g_1(x, y)/g_0(x, y) \notin]\frac{1-\tau}{\tau}, \frac{\tau}{1-\tau}[$. Furthermore, the points with $Err(x, y) \leq \tau$ are either the points above the line: $\log\left(\frac{1-\tau}{\tau} \times \frac{1-s}{s}\right) \leq x \log\left(\frac{e_1}{1-e_0}\right) + y \log\left(\frac{1-e_1}{e_0}\right)$ (accepting points) or the points below the line: $\log\left(\frac{\tau}{1-\tau} \times \frac{1-s}{s}\right) \geq x \log\left(\frac{e_1}{1-e_0}\right) + y \log\left(\frac{1-e_1}{e_0}\right)$ (rejecting points). We note that both lines have identical slopes and are therefore parallel.*

▶ **Example 5.** In the running example, the equations of the SPRT lines are approximately $1.11 \times x - 2 \pm 4.2$. To facilitate the comparison of strategies, the SPRT lines are represented in blue and red on every plot of Figure 1.

As shown by Wald [23], this property allows to approximate in constant time the expected cost of the SPRT. Even though an arbitrary number of questions may be needed to reach a decision, the expected cost is typically small [23].

Limitations

Although SPRT is optimal when the budget for questions is unlimited [23], it yields unbounded strategies which may issue an arbitrary (possibly infinite) number of questions, and is thereby not suitable for our limited budget.

Truncated SPRT

To limit the maximum number of questions, Wald also introduces the truncated SPRT, similar to SPRT, except that all points with $x + y = m$ are terminating to guarantee a decision is reached after at most m questions. Obviously, we then prune the strategy along the lines $y = y_{\texttt{dec}}$ and $x = x_{\texttt{dec}}$ as detailed in Section 2.3.

▶ **Example 6.** Figure 1(a) represents the truncated SPRT strategy for the running example with brown dots around the truncation points. Its error; 0.008, exceeds τ, because the truncation includes some decision points with $\mathrm{Err}(x, y) > \tau$. To compensate for this additional error, any feasible strategy must therefore include points further from the SPRT lines.

The truncated SPRT provides a strategy within constant time, since one only needs to compute the likelihood ratio r of the current point (x, y) to decide whether to continue $r \in]\frac{\tau}{1-\tau}, \frac{1-\tau}{\tau}[$, accept ($r \geq \frac{1-\tau}{\tau}$) or reject ($r \leq \frac{\tau}{1-\tau}$]).

But the error of the strategy may be larger than τ since the truncation points have error larger than τ. In some instances, the truncated SPRT still returns a feasible strategy, e.g. when some decision points along the SPRT lines have an error slightly less than τ, thus compensating for the additional error caused by the truncation. But feasibility is not always guaranteed, and therefore the truncated SPRT cannot be trusted to solve our problem.

3.2 Adapting the SPRT Threshold

As SPRT cannot be trusted to provide feasible strategies, we propose a new adaptation of the SPRT strategy, called `AdaptSprt`, which preserves the simplicity of the SPRT approach but always returns a feasible solution.

Intuitively, the `AdaptSprt` algorithm computes the best strategy whose terminating points form two lines, parallel and equidistant to the decision line, plus truncation points along $x = x_{\texttt{dec}}$ and $y = y_{\texttt{dec}}$. In other words, `AdaptSprt` starts from initial strategy $\sigma_{\texttt{rect}}(x_{\texttt{dec}}, y_{\texttt{dec}})$ and turns the points further from the decision line into terminating points, as long as the error of the strategy remains below the authorized threshold. This guarantees that a feasible strategy will always be returned when there is one. For efficiency, we use binary search to determine which points can be turned into terminating points.

Algorithm `AdaptSprt` can be defined more formally in terms of the likelihood ratio. For all $\eta > 0$ let σ_η be the strategy that continues asking questions on all points (x, y), $x \leq x_{\texttt{dec}}, y \leq y_{\texttt{dec}}$ where the likelihood ratio belongs to $]1/\eta, \eta[$. `AdaptSprt` computes the maximal threshold η for which σ_η is feasible. Algorithm 1 details the steps in `AdaptSprt`. We first build in $O(m^2 \log m)$ a list of all points (x, y) with $x < x_{\texttt{dec}}$ and $y < y_{\texttt{dec}}$, ordered by increasing likelihood ratio r (lines 1,2 of Algorithm 1). The continuing points of the `AdaptSprt` strategy will be the first i points from the list, for some index i. As the error (resp. the cost) increases

Algorithm 1: `AdaptSprt` (e_0, e_1, τ, m, s)

```
1  for all $x, y$, compute $r(x, y) = g_1(x, y)/g_0(x, y)$
2  $L \leftarrow$ points $(x, y)$ ordered by increasing $r(x, y)$
3  Compute $i_0 = \min\{i \mid \texttt{EvalErr}(i, L) < \tau\}$
4  return $r(L(i_0))$

   procedure EvalErr($i$ : int, $L$ : point list)
5      for $j$ in $\{0, \ldots, i-1\}$
6          $P_{stop}(L(j)) \leftarrow 0$
7      for $j$ in $\{i+1, \ldots\}$
8          $P_{stop}(L(j)) \leftarrow 1$
9      Compute and return the error of strategy $P_{stop}$
```

(resp. decreases) with i, the optimal strategy of this form is obtained by computing the minimal i that gives a feasible strategy. The strategy corresponding to index i is evaluated by procedure `EvalErr` in $O(m^2)$, and we can use binary search to compute the minimal index within $\log(m^2)$ iterations. Hence an overall complexity of $O(m^2 \log(m))$.

To represent the `AdaptSprt` strategy, the value r of the likelihood ratio of the i^{th} point is sufficient: when asking queries we can calculate in constant time whether a point has likelihood ratio between $1/r$ and r, and thus reconstruct the grid on the fly. We can also compute the strategy P_{stop} from the list and the index i, as shown in procedure `EvalErr` from Algorithm 1 if a grid representation is preferred.

▶ **Proposition 7.** *The (time) complexity of* `AdaptSprt` *is* $O(m^2 \log(m))$.

▶ **Example 8.** Figure 1(b) presents the `AdaptSprt` strategy for the running example, with termination lines represented as dashed lines. The truncation of the SPRT raises the error substantially above τ, so that `AdaptSprt` must adopt a likelihood ratio threshold η much larger than $(1-\tau)/\tau$ to compensate for the truncation. The dashed lines are thus almost one question beyond those of SPRT.

4 Deterministic Algorithms

We next investigate the scalability of algorithms proposed by Parameswaran et al. [22] for computing strategies. Specifically, we analyze the complexity of these algorithms and present optimizations that drastically reduce the running time of the algorithms compared to the more naïve versions presented in [22], thereby allowing to support larger budgets.

4.1 Enumeration of Ladder Strategies

The most naïve approach to compute an optimal strategy is to enumerate and evaluate all possible strategies. This naïve approach has complexity $O(m^2 \times 2^{m^2/2})$ and is thus intractable.[1] Parameswaran et al. [22] therefore proposed the `ladder` algorithm which limits the search to ladder strategies, as explained in Section 2.3. They report running times that are reasonable for small values of m ($m \leq 14$), but grow exponentially and become unfeasible

[1] A bound of $m^2 \times 2^m$ was improperly claimed in [22] for the naïve enumeration of grids but it is clear from their proof that the actual bound is $O(m^2 \times 2^{m^2/2})$[21]

as m gets larger. We first establish a tight exponential bound for the complexity of `ladder`, and then introduce optimizations offering much shorter running time in practice, in spite of a similar worse case exponential complexity.

Asymptotic Analysis

We prove that the complexity of ladder is essentially $O(2^{2m})$. Our exponential bound bears witness to the efficiency of the `ladder` algorithm relative to the enumeration of all possible (not necessarily ladder-shaped) strategies. For this we can easily show the following lower bound:

▶ **Lemma 9.** *The number of possible upper and lower ladders can be roughly bounded by $O(2^m/\sqrt{m})$. Hence, there are $O(2^{2m}/m)$ deterministic ladder strategies.*

This is an overapproximation, yet a fairly accurate one: we show in the technical report [11] that for $s = 0.5, e_0 = e_1$, the number of ladder strategies is $\Omega(2^{2m}/m^3)$.

Enumeration with Incremental Evaluation

We detail in Algorithm 2 an optimized implementation that computes the cost and error of every ladder strategy incrementally, in overall $O(2^{2m})$. We first discuss our representation of a ladder strategy and then explain our optimizations.

As mentioned in Section 2.3, a ladder strategy consists of two distinct sequences of points: the upper ladder and the lower ladder. Each ladder is represented as an array with size $x_{\texttt{dec}}$ storing integers from -1 up to $y_{\texttt{dec}}$. Array `up` and `down` represent respectively the upper and lower ladder: `down`(i) and `up`(i) record respectively the lowest and highest reachable points on column i according to the strategy.

▶ **Example 10.** Figure 1(c) represents the optimal ladder strategy for our input parameters: `up` $= [5, 5, 6, 7, 8, 8, 8, 8]$ and `down` $= [-1, \ldots, -1, 0, 1]$. None of the other algorithms depicted returns the optimal strategy on that instance, although the performances are quite similar.

We adapt an old technique (see [14, Algorithm P]) to iterate over all upper ladders in increasing lexicographic order, and enumerate for each one the lower ladders in decreasing order. As a result, arrays representing successive strategies generally differ on the last few columns only, which reduces the amount of work required to evaluate a strategy.

Two simple optimizations allow us to speedup the enumeration: (1) we evaluate incrementally the cost and error of strategies, and (2) we skip some strategies that cannot contribute an optimal solution. For this, we store two arrays `errorTill` and `costTill`, where `errorTill`(i) records the partial sum of E restricted to the points with $x \leq i$, and similarly with `costTill` for C. We update `errorTill`, `costTill`, and *Path* from one strategy to the next (line 7 of Algorithm 2). The iterator `down`.*next*() returns $(-1, [])$ if `down` is already the minimal ladder, and otherwise returns the greatest possible ladder `down`$'$ smaller than `down`, together with the smallest index i in which `down` and `down`$'$ differ. To skip hopeless candidates, we set `down`(j) to `down`(i) for all $j > i$ when the error up to column i exceeds the threshold, or the cost up to column i exceeds the cost of the best strategy encountered so far (line 13 in Algorithm 2).

▶ **Example 11.** When experimenting on the running example, more than half the strategies were skipped in line 13, and the average index i was 5.5. Some 16 points were visited per strategy, on average, when updating arrays and matrix in line 7, instead of $\approxeq 56$ without incremental evaluation.

Algorithm 2: `ladder` (e_0, e_1, τ, m, s)

```
errorTill, costTill ← $[0, 0, \ldots, 0]$
BestCost ← $m + 1$
BestStrategy ← $Null$
for up in upperladders
    down ← maximal lowerladder; $i \leftarrow 0$
    while $i \geq 0$
        Update errorTill, costTill, $Path$
        if (errorTill$[m] < \tau$ and
                costTill$[m]$ < BestCost$[m]$ )
            BestCost ← costTill$[m]$
            BestStrategy ← (up, down)
        if (errorTill$[i] > \tau$)
            skip ladders until down$(i)$ is modified
        else $(i,$ down$)$ ← down.$next()$
return BestStrategy
```

We show in the technical report [11] that the average number of cells updated on line 7 is m. As a consequence, Algorithm 2 has complexity $O(2^{2m}/m) \times O(m)$.

▶ **Proposition 12.** *Algorithm 2 runs in $O(2^{2m})$.*

We have thus proved that an optimal ladder can be obtained in $O(2^{2m})$, and the number of possible ladders strategies is exponential. This does not preclude the existence of faster algorithms, and we leave lower bounds on the complexity of the problem for future research.

4.2 Shrink

Another interesting heuristic-based algorithm introduced by Parameswaran et al. [22] is `shrink`. The strategies returned by this heuristic are not necessarily optimal, but are hardly worse than the optimal ladder strategy in practice, while the running time is much improved. A naïve implementation following [22] has complexity $O(m^5)$ and therefore, does not scale well for large values of m. We next show how `shrink` can be run in $O(m^4)$.

We recall the `shrink` heuristic from [22] in Algorithm 3. This algorithm starts with the initial strategy $\sigma_{\texttt{triangle}}(m)$ having terminating points along the line $x + y = m$. At each iteration, for each terminating point (x, y) on the grid, we check if the solution would remain feasible if we were to turn one of the neighboring points $(x-1, y)$, $(x, y-1)$ into a terminating point. For all feasible point we calculate the change in cost ΔC and error ΔE that would result from shrinking the point. We then shrink the point with the largest ratio $-\frac{\Delta C}{\Delta E}$ and repeat this step until no more points can be shrunk. We thus use ratio $-\frac{\Delta C}{\Delta E}$ in order to maximize the cost removed from the strategy while minimizing the additional error.

▶ **Example 13.** In Figure 1(d), we shade points that were turned into terminating points along the successive iterations of `shrink`, with darker points corresponding to later iterations[2]. The first point is thus $(0, 7)$, followed by $(1, 7), (0, 6), \ldots, (6, 1)$, and $(5, 0)$.

[2] Terminating points with $x = 8$ or $y = 8$ are particular in that they were not shrinked but were terminating from the beginning. We color them in dark red and blue.

Algorithm 3: `shrink` (e_0, e_1, τ, m, s)

```
1  Compute $S_0, S_1, x_{\mathtt{dec}}, y_{\mathtt{dec}}$
2  for all $x, y$:
       $P_{stop}(x,y) \leftarrow \begin{cases} 1 \text{ if } x+y=m, \\ 0 \text{ otherwise} \end{cases}$
3  Compute $Path$, $\Delta_{\text{Cost}}$ and $\Delta_{\text{Err}}$
4  Error $\leftarrow \Delta_{\text{Err}}(0,0)$
5  $S \leftarrow \{$(x,y) along the boundary $|$
           Error$+Path(x,y) \times \Delta_{\text{Err}}(x,y) < \tau\}$
6  while $S \neq \emptyset$
7      $(x_0, y_0) \leftarrow$ point of $S$ maximizing $-\frac{\Delta_{\text{Cost}}(x,y)}{\Delta_{\text{Err}}(x,y)}$
8      $P_{stop}(x_0, y_0) \leftarrow 1$
9      for all $x, y$: update $Path$, $\Delta_{\text{Cost}}$, $\Delta_{\text{Err}}$
10     update $S$
11 return $P_{stop}$
```

Algorithm `shrink` from [22] is polynomial, but still pretty slow. We next present new equations for the ratios together with a pruning optimization, that make it run faster.

4.2.1 Computing Cost/Error Ratios Efficiently

A major source of inefficiency in the above `shrink` implementation is the calculation of the Cost/Error ratio in each iteration. The naïve implementation of `shrink` computes the Cost/Error ratio separately for each terminating point on the grid, by evaluating the cost and error of the shrunken strategy. As there are $\Omega(m)$ terminating points this requires $\Omega(m^3)$ operations per iteration. We introduce new equations that help compute $\Delta_{\text{Cost}}(x,y)$ and $\Delta_{\text{Err}}(x,y)$ for all points (x,y), with overall complexity $O(m^2)$. Algorithm `shrink` was initially designed to compute deterministic strategies, but in Section 5 we extend it to probabilistic strategies, so we present all equations in a general probabilistic setting.

Impact of Modifying the Probability to Stop

Let us denote by $\mathtt{CostImpact}(x,y)$ and $\mathtt{ErrorImpact}(x,y)$ the average contributions to cost and error of one single path through (x,y) (and possibly stopping at (x,y)). We then have:

$$\mathtt{CostImpact}(x,y) = P_{stop}(x,y) * X + (1 - P_{stop}(x,y)) * Y$$
$$\mathtt{ErrorImpact}(x,y) = P_{stop}(x,y) * Z + (1 - P_{stop}(x,y)) * T$$

where X, Y, Z and T are defined as

$$X = (S_0(x,y) + S_1(x,y)) * (x+y) \qquad Y = \mathtt{CostImpact}(x+1,y) + \mathtt{CostImpact}(x,y+1)$$
$$Z = \min(S_0(x,y), S_1(x,y)) \qquad T = \mathtt{ErrorImpact}(x+1,y) + \mathtt{ErrorImpact}(x,y+1)$$

Intuitively, X and Z are the contribution to the overall cost and error from any sequence of x negative answers and y positive answers, whereas Y and T are inductively defined as the contribution to cost and error of a path traversing the node. To compute the impact of modifying the strategy at (x,y) in terms of these expressions, let E, E' and C, C' denote the cost and error of the strategy before and after adding $\delta \in [-1,1]$ to $P_{stop}(x,y)$. Then

$E' - E = \delta \times \mathit{Path}(x, y) \times \Delta_{\mathrm{Err}}$ and $C' - C = \delta \times \mathit{Path}(x, y) \times \Delta_{\mathrm{Cost}}$ where

$$\Delta_{\mathrm{Cost}} = X - Y \qquad \text{and} \qquad \Delta_{\mathrm{Err}} = Z - T \tag{4}$$

We observe in these equations that the Cost/Error ratio is independent of δ and $\mathit{Path}(x, y)$, and is given by $\gamma(x, y) = (T - Z)/(X - Y)$. `CostImpact` and `ErrorImpact` can be computed recursively in $O(m^2)$ over the whole grid, starting from point $(x_{\mathtt{dec}}, y_{\mathtt{dec}})$. We have thus proved that Δ_{Err} and Δ_{Cost} can be computed at all points in overall $O(m^2)$ according to Equations 4. Furthermore, *Path* can also be computed in $O(m^2)$ according to the preliminaries, so that each iteration of `shrink` takes time $O(m^2)$. In addition, there are at most $O(m^2)$ such iterations, since the number of iterations is at most the number of squares on the grid. Therefore, our implementation of shrink runs in $O(m^4)$.

▶ **Proposition 14.** *With our optimizations, the* `shrink` *algorithm runs in* $O(m^4)$.

Note however that the actual number of iterations is proportional to the number of points removed from the grid so the running time is quadratic when few points are removed.

4.2.2 Minimizing Shrink Iterations

To further speed up the computation we show how the pruning optimization described in subsection 2.3 can spare about half the iterations. Specifically, we prune the initial strategy $\sigma_{\mathtt{triangle}}(m)$ into $\sigma_{\mathtt{rect}}(x_{\mathtt{dec}}, y_{\mathtt{dec}})$. To justify this move, we show in the technical report [11] that the points that are pruned are anyway the first points eliminated by `shrink`.

▶ **Proposition 15.** *The first iterations of* `shrink` *from the initial strategy* $\sigma_{\mathtt{triangle}}(m)$ *eliminate the points with* $x > x_{\mathtt{dec}}$ *or* $y > y_{\mathtt{dec}}$ *until the strategy* $\sigma_{\mathtt{rect}}(x_{\mathtt{dec}}, y_{\mathtt{dec}})$ *is considered. Consequently the solutions returned by* `shrink` *from initial strategy* $\sigma_{\mathtt{triangle}}(m)$ *and from* $\sigma_{\mathtt{rect}}(x_{\mathtt{dec}}, y_{\mathtt{dec}})$ *are identical.*

▶ Remark. Another heuristic, symmetric to `shrink`, was introduced in [22]. This `growth` heuristic starts with the initial strategy asking 0 questions: all points are initially terminating, and then iteratively turns terminating points into continuing points. Heuristic `growth` did not always return a feasible strategy, but we show in the technical report [11] that adopting a better initial strategy wipes off the problem. The performances of `growth` and `shrink` are fairly similar, so we do not detail the heuristic further in this paper.

5 Randomized strategies

Previous sections focus on deterministic strategies, for which we have no optimal scalable algorithm. But if we search instead for probabilistic strategies, our optimization problem becomes continuous, and the constraints presented in Section 2.1 are all linear. We can thus use linear programming to compute an optimal solution in PTIME [22]. How are the probabilistic and deterministic strategies related? In particular, can we compute reasonably good deterministic strategies from probabilistic ones?

In this section we first prove that an optimal probabilistic strategy has essentially a *single* probabilistic point (point where P_{stop} differs from 0 and 1). Continuing at this point thus provides a deterministic strategy. Conversely, we show that a minor modification to the `shrink` algorithm allows to compute an optimal probabilistic strategy.

5.1 Randomization is Limited

We prove in the technical report [11] that in any optimal strategy the Cost/Error ratio is the same in all probabilistic points, and this ratio is not greater than the ratio of any terminating point nor smaller than the ratio of any continuing point. We also prove that the probability of terminating can be transferred from a point to any point with higher ratio without increasing error and cost, and exploit this property to prove the following result:

▶ **Proposition 16.** *There exists an optimal probabilistic strategy with a single probabilistic point. Furthermore, in any optimal strategy the probabilistic points maximize the ratio γ among non-terminating points.*

By turning the unique probabilistic point of such a strategy into a continuing point, one thus obtains a deterministic strategy with error less than τ and with slightly larger cost.

▶ **Example 17.** Figure 1(e) represents an optimal probabilistic strategy, with a single probabilistic point, at $(0, 4)$, where the probability of terminating is $P_{stop}(0, 4) \approxeq 0.623$. If we set $P_{stop}(0, 4)$ to 0, the cost rises to $\approxeq 7.789$.

The linear programming techniques mentioned above are very efficient for small values of m, and have polynomial complexity in theory. In practice, however, our experiments with common linear solvers show that they may be rather slow or inaccurate, returning poor strategies even for moderately large budgets. We therefore propose an alternative efficient algorithm based on `shrink` to compute optimal probabilistic strategies.

5.2 Shrink for Randomized Strategies

Algorithm `shrink` as defined in [22] returns a deterministic, not necessarily optimal, strategy, but it can easily be adapted to compute an optimal randomized strategy by replacing lines 5, 6, and 8 with respectively:

- line 5: $S \leftarrow \{(x, y) \mid \text{(x,y) is reachable}\}$
- line 6: **while** $S \neq \emptyset$ and $\texttt{Error} < \tau$
- line 8: $P_{stop}(x_0, y_0) \leftarrow \min(1, \frac{\tau - \texttt{Error}}{Path(x,y) \times \Delta_{\text{Err}}(x,y)})$

This new algorithm `shrinkp` still computes the point with the maximal ratio, but adapts the probability of terminating at this point so as not to exceed error τ, instead of restricting the maximum to points on which one can terminate without exceeding error τ. It turns out that `shrinkp` returns an optimal strategy (we leave the proof for the technical report [11]):

▶ **Proposition 18.** *The probabilistic strategy returned by* `shrinkp` *is optimal, and has a single probabilistic point.*

This result sheds a new light on the `shrink` algorithm, but it can also be used to leverage the running time of `shrink` and the linear program, since `shrink` and `shrinkp` coincide at any step until the last iteration of `shrink` as we discuss in the technical report [11].

6 Experimental evaluation

Synthetic and real-crowd experiments: quality of the strategies

To complement the theoretical study we conducted experiments on a large set of synthetic parameters. Due to space constraints we only present a small sample of our experiments here and leave details for the technical report [11]. Those experiments show that some linear

	question	s	e_0	e_1
Q1	photos from Australia	.18	.25	.36
Q2	photos from Greece or Cyprus	.26	.27	.32
Q3	dishes containing dairy	.17	.11	.27
Q4	dishes containing onions	.54	.38	.27
~~$Q5$~~	~~dishes containing garlic~~	~~.62~~	~~.44~~	~~.48~~
~~$Q6$~~	~~dishes containing eggs~~	~~.19~~	~~.22~~	~~.57~~

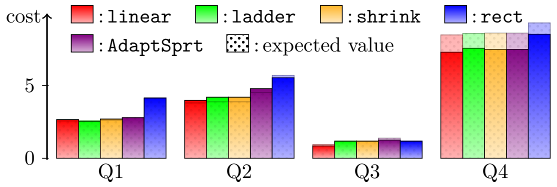


Figure 2 Question parameters(left) and average cost per item (right, with $m = 12$, $\tau = .1$).

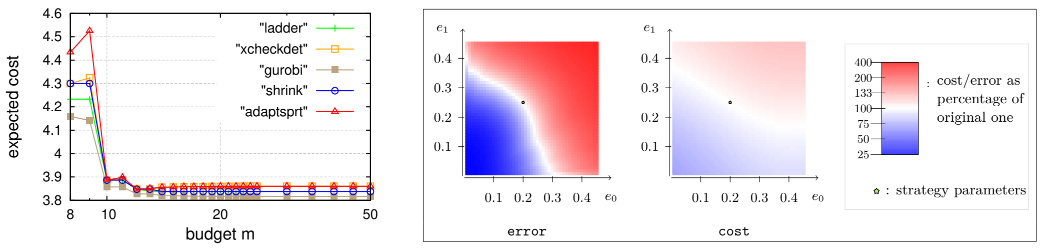


Figure 3 For $e_0 = .2, e_1 = .25, \tau = .05, s = .6$: cost, and sensitivity (only for `shrink` with $m = 15$).

program solvers become unreliable for budgets beyond $m = 30$ questions, while `ladder` times out around $m = 20$ and `shrink` and `AdaptSprt` manage hundreds of questions. The expected cost of strategies matches theoretical expectations with `AdaptSprt` slightly worse than `shrink` and `ladder`, themselves a bit more expensive than the optimal probabilistic strategies. The experiments on a real crowd with budgets up to $m = 40$ exhibit similar patterns. Figure 2 depicts the quality of strategies obtained when asking the crowd to detect (a) the presence of an ingredient in some recipe or (b) the location of a photograph. Experiments were run with a pool of 100 workers on the AskIt [4] crowdsourcing game platform, developed in our lab.

The error rates, summarized in Figure 2, are relatively high because answers were rarely obvious. For question 6 in particular, e_1 was above .5 which means the users more often than not missed the presence of eggs in the dishes. We focus our analysis on questions with reasonable error rates ($Q1$ to $Q4$).

Sensitivity of the model

Applying our algorithms on a real crowd raised new issues such as the adequacy of the model considered. Our algorithms indeed assume the crowd behaves as a random oracle according to error parameters known beforehand. Our synthetic experiment in Figure 3 measures the sensitivity of a strategy computed by `shrink` to input parameters: it shows the expected error and cost when the strategy is executed on an oracle with error parameters diverging from their assumed values. A related issue is the relevance of approximating workers as a random oracle with uniform error over tasks: a threshold effect appears when we try to request arbitrarily high accuracy: when τ is set to a very small values, adding workers did not always provide in practice additional information to complete the most difficult tasks with enough accuracy.

7 Conclusion and Related work

This paper investigates the optimization of queries that filter data using humans. We provided new theoretical insights into the problem, and so designed two novel algorithms – `AdaptSprt` and `shrinkp` – that overcome the scalability issues of previous proposals. We also optimized algorithms `ladder` and `shrink` from [22], and evaluated thoroughly all algorithms. Our results show that `AdaptSprt` is the only algorithm which performs well for all budgets, while `ladder` performs marginally better for small budgets, but is still extremely slow with larger ones (even when optimized), whereas for moderate budgets our optimized `shrink` works well. With regard to probabilistic strategies, the results show `shrinkp` to have superior reliability, compared to the previous proposals that rely on linear solvers. In summary, our results show that `AdaptSprt` and `shrinkp` both scale well for large budgets. Although cost wise `shrinkp` is optimal, the actual difference of cost is negligible while the running time of `AdaptSprt` is superior.

We already discussed extensively the CrowdScreen framework [22] revisited in this paper. Parameswaran et al. have reviewed in [22] the connections with the related fields in machine learning and statistics, and we thus do not repeat this here and only briefly survey two directions of related work: sequential testing, and classifying with the Crowd.

Sequential tests have been used in numerous fields since their introduction by Wald [23]: quality control, clinical research, acoustic detection, econometrics, etc. Numerous variants have been considered for computing efficient tests, depending on the number of categories tested to which an object may belong; the cost function to be optimized; the form of the strategy boundary [2] and budget constraint [9]; or on whether questions are issued one at a time or in batches [15]. To the best of our knowledge, however, the problem of efficiently computing the optimal test, in the sense studied here, has not yet been addressed. Closest to our work is the system of [10] considering the profit/penalty of correct/wrong answers in a multi-question scenario. Extending our work into such settings is left for future work.

The optimal strategy depends on the query selectivity and the estimated users error. Experiments in [17] stress that classifier performance improves a lot with a proper choice of prior error rates. In practice, the nature of error can be estimated by asking questions to the crowd on a small test set for which the correct answer is already known. Online methods to calculate error rates are discussed in [16], [5], [25] where the error rates are tuned based on comparison of the strategy's decision and the users' answers. One goal of our framework is to avoid any kind of computation online by fixing the filtering strategies beforehand. Adapting strategies according to online error computation is left for further research.

Classification problems with heterogeneous workers and data have been considered in particular in the machine learning literature, exploiting a wide range of techniques from multi-armed bandit problems [1] to singular value decomposition [13], Bayesian learning [24] and variational inference in graphical models [17]. Users and tasks with diverging characteristics raise the challenge of selecting tasks and users to make the most of the budget. For example, Karger et al. [13] propose an algorithm to assign questions to heterogeneous workers with optimal tradeoff between redundancy and accuracy. Empirical models have also been proposed to improve the accuracy of classification by identifying annotation patterns (inherent difficulty of images, groups of users with similar behaviors) [24, 17]. Incorporating some of these ideas in our work is a challenging future work.

Our results focus on binary filters that classify items in two disjoint sets, but can easily be adapted to classify items among n classes, though complexity increases exponentially with n. Devising optimizations to improve performance in this setting is thus a future challenge. Furthermore, processing several filters simultaneously may allow to exploit correlations between filters, or to select dynamically the questions that would be most informative [4, 6, 12].

Finally, empirical studies show that batching tasks may have positive impact on Crowd-Sourcing efficiency [19]. Similarly, pre-recruiting schemes [3], that allow to obtain answers from the workers within seconds, may help to exploit the full benefit of sequential testing without increasing latency. Devising optimization strategies with batches is challenging.

Acknowledgements. The authors are very thankful to A. Parameswaran for helpful discussions. This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, and by the Israel Ministry of Science.

References

1 Ittai Abraham, Omar Alonso, Vasilis Kandylas, and Aleksandrs Slivkins. Adaptive crowdsourcing algorithms for the bandit survey problem. *To appear in JMLR W&CP*, 30, 2013.

2 T. W. Anderson. A modification of the sequential probability ratio test to reduce the sample size. *The Annals of Math. Stat.*, 31(1):pp. 165–197, 1960. URL: `http://www.jstor.org/stable/2237502`.

3 Michael S. Bernstein, David R. Karger, Robert C. Miller, and Joel Brandt. Analytic methods for optimizing realtime crowdsourcing. In *Collective Intelligence*, 2012.

4 Rubi Boim, Ohad Greenshpan, Tova Milo, Slava Novgorodov, Neoklis Polyzotis, and Wang Chiew Tan. Asking the right questions in crowd data sourcing. In *ICDE*, pages 1261–1264, 2012. `doi:10.1109/ICDE.2012.122`.

5 Peng Dai, Mausam, and Daniel S. Weld. Decision-theoretic control of crowd-sourced workflows. In *AAAI*, 2010.

6 Nilesh N. Dalvi, Aditya G. Parameswaran, and Vibhor Rastogi. Minimizing uncertainty in pipelines. In *NIPS*, pages 2951–2959, 2012.

7 Jacques Dutka. The incomplete beta function – a historical profile. *Archive for History of Exact Sciences*, 24(1):11–29, 1981. `doi:10.1007/BF00327713`.

8 Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011. `doi:10.1145/1989323.1989331`.

9 Peter Frazier and Angela J. Yu. Sequential hypothesis testing under stochastic deadlines. In *NIPS*, 2007.

10 Jinyang Gao, Xuan Liu, Beng Chin Ooi, Haixun Wang, and Gang Chen. An online cost sensitive decision-making method in crowdsourcing systems. In *SIGMOD*, pages 217–228, 2013. `doi:10.1145/2463676.2465307`.

11 Benoit Groz, Ezra Levin, Isaco Meilijson, and Tova Milo. Filtering with the crowd: Crowdscreen revisited. https://hal.archives-ouvertes.fr/view/index/docid/1239458.

12 Haim Kaplan, Ilia Lotosh, Tova Milo, and Slava Novgorodov. Answering planning queries with the crowd. *PVLDB*, 6(9):697–708, 2013.

13 David R. Karger, Sewoong Oh, and Devavrat Shah. Efficient crowdsourcing for multi-class labeling. In *SIGMETRICS*, pages 81–92, 2013. `doi:10.1145/2465529.2465761`.

14 Donald E. Knuth. *The Art of Computer Programming, Volume IV, draft of 7.2.1.6.* Addison-Wesley, 2004.

15 Walter Lehmacher and Gernot Wassmer. Adaptive sample size calculations in group sequential trials. *Biometrics*, 55(4):1286–1290, 1999. `doi:10.1111/j.0006-341X.1999.01286.x`.

16 Christopher H. Lin, Mausam, and Daniel S. Weld. Dynamically switching between synergistic workflows for crowdsourcing. In *AAAI*, 2012.

17 Qiang Liu, Jian Peng, and Alexander T. Ihler. Variational inference for crowdsourcing. In *NIPS*, pages 701–709, 2012.

18 Adam Marcus, David R. Karger, Samuel Madden, Rob Miller, and Sewoong Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.

19 Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

20 Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.

21 Aditya G. Parameswaran. Personal Communication.

22 Aditya G. Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012. `doi:10.1145/2213836.2213878`.

23 A. Wald. Sequential tests of statistical hypotheses. *The Annals of Math. Stat.*, 16(2):pp. 117–186, 1945. URL: `http://www.jstor.org/stable/2235829`.

24 Peter Welinder, Steve Branson, Serge Belongie, and Pietro Perona. The multidimensional wisdom of crowds. In *NIPS*, pages 2424–2432, 2010.

25 Jacob Whitehill, Paul Ruvolo, Tingfan Wu, Jacob Bergsma, and Javier R. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*, pages 2035–2043, 2009.