Filtering with the Crowd: CrowdScreen revisited (technical report)

Benoît Groz, Ezra Levin, Isaac Meilijson, Tova Milo

▶ To cite this version:

Benoît Groz, Ezra Levin, Isaac Meilijson, Tova Milo. Filtering with the Crowd: CrowdScreen revisited (technical report). 19th International Conference on Database Theory (ICDT 2016), Mar 2016, Bordeaux, France. pp.12:1–12:18, 10.4230/LIPIcs.ICDT.2016.12 . hal-01239458

HAL Id: hal-01239458
https://hal.science/hal-01239458

Submitted on 7 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Filtering with the Crowd: CrowdScreen revisited (technical report).

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

Tel-Aviv, Israel
[1] benoit.groz@lri.fr
[2] ezralevin@gmail.com
[3] isaco@post.tau.ac.il
[4] 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.

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 [21, 10]. 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 [20, 19]. 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

© Benoît Groz and Ezra Levin and I. Meilijson and T.Milo;
licensed under Creative Commons License CC-BY

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* [24], 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 [23].

To study the optimization of filtering, we adopt in this paper a simple and general model by Parameswaran et al. [23], 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 [23] 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 [24, 4] 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 [23] 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 [24]. We then revisit, in Section 4, the heuristics from [23]. 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 [23], 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). We deferred some of the proofs to Appendix A. Finally, to complement our theoretical study we briefly illustrate (with more details in Appendix A) 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 [23] 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 [23]. 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 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 $\text{Err}(x,y)$ denote the probability of error when making a decision at point (x,y) (we detail in the next section how to calculate $\text{Err}(x,y)$). 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:

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

The Problem Definition

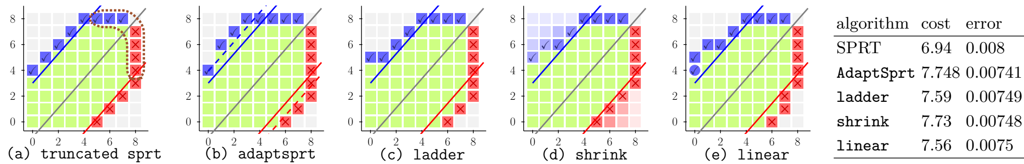
The error threshold τ fixes the maximal error a strategy is allowed. A strategy is *feasible* if it satisfies the budget and error constraints m and τ, and *optimal* if it has minimal cost among feasible strategies. Our objective is to find the optimal strategy, given the priors e_0, e_1, s and the constraints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

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



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 $s = .8$.

Up till now, the introduction followed the definitions and equations of [23], 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:

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

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 [23]:

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

▶ 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 [23] and the other two are new.

Ladder Strategies

Parameswaran et al. [23] 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 [23] that any optimal strategy is a ladder strategy. We adopt this conjecture and focus in this paper on ladder strategies.

Pruning the Grid

Let $(x_{\texttt{dec}}, y_{\texttt{dec}})$ 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 [9], 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 [24] 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 [24] 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 3.1. A point (x, y) satisfies $\mathrm{Err}(x, y) \leq \tau$ 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 $\mathrm{Err}(x, y) \leq \tau$ 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 4.** 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 [24], 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 [24].

Limitations

Although SPRT is optimal when the budget for questions is unlimited [24], 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 5.** 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 (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 ith 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 3.2. The (time) complexity of `AdaptSprt` is $O(m^2 \log(m))$.

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

```
1  for all x, y, compute r(x, y) = g1(x, y)/g0(x, y)
2  L ← points (x, y) ordered by increasing r(x, y)
3  Compute i0 = min{i | EvalErr(i, L) < τ}
4  return r(L(i0))
   procedure EvalErr(i : int, L : point list)
5      for j in {0, ..., i − 1}
6          P_stop(L(j)) ← 0
7      for j in {i + 1, ...}
8          P_stop(L(j)) ← 1
9      Compute and return the error of strategy P_stop
```

▶ **Example 6.** 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. [23] 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 [23], 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. [23] 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 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.

[1] A bound of $m^2 \times 2^m$ was improperly claimed in [23] for the naïve enumeration of grids but it is clear from their proof that the actual bound is $O(m^2 \times 2^{m^2/2})$[22]

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 7.** *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 Appendix A 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: $\texttt{down}(i)$ and $\texttt{up}(i)$ record respectively the lowest and highest reachable points on column i according to the strategy.

▶ **Example 8.** Figure 1(c) represents the optimal ladder strategy for our input parameters: $\texttt{up} = [5, 5, 6, 7, 8, 8, 8, 8]$ and $\texttt{down} = [-1, \ldots, -1, 0, 1]$. 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 [15, 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 $\texttt{errorTill}(i)$ 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 $\texttt{down}.next()$ returns $(-1, [])$ if `down` is already the minimal ladder, and otherwise returns the greatest possible ladder $\texttt{down}'$ smaller than `down`, together with the smallest index i in which `down` and $\texttt{down}'$ differ. To skip hopeless candidates, we set $\texttt{down}(j)$ to $\texttt{down}(i)$ 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 9.** 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.

We show in Appendix A 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 4.1. Algorithm 2 runs in $O(2^{2m})$.

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

```
1  errorTill, costTill ← [0, 0, ..., 0]
2  BestCost ← m + 1
3  BestStrategy ← Null
4  for up in upperladders
5      down ← maximal lowerladder; i ← 0
6      while i ≥ 0
7          Update errorTill, costTill, Path
8          if (errorTill[m] < τ and
9                  costTill[m] < BestCost[m] )
10             BestCost ← costTill[m]
11             BestStrategy ← (up, down)
12         if (errorTill[i] > τ)
13             skip ladders until down(i) is modified
14         else (i, down) ← down.next()
15 return BestStrategy
```

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. [23] 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 [23] 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 [23] 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 10.** 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 $(5, 0)$.

Algorithm `shrink` from [23] is polynomial, but still pretty slow. We next present new equations for the ratios together with a pruning optimization, that make it run faster.

[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 $S_0, S_1, x_{\texttt{dec}}, y_{\texttt{dec}}$
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*, Δ_{Cost} and Δ_{Err}
4 `Error` $\leftarrow \Delta_{\text{Err}}(0,0)$
5 $S \leftarrow \{$(x,y) along the boundary $|$
  `Error`$+ \mathit{Path}(x,y) \times \Delta_{\text{Err}}(x,y) < \tau\}$
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*, Δ_{Cost}, Δ_{Err}
10   update S
11 **return** P_{stop}

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

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

where X, Y, Z and T are defined as

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

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

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

We observe in these equations that the Cost/Error ratio is independent of δ and $Path(x, y)$, 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 $(x_{\texttt{dec}}, y_{\texttt{dec}})$. 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 4.2. 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 $\sigma_{\texttt{triangle}}(m)$ into $\sigma_{\texttt{rect}}(x_{\texttt{dec}}, y_{\texttt{dec}})$. To justify this move, we show in Appendix A that the points that are pruned are anyway the first points eliminated by `shrink`.

▶ Proposition 4.3. The first iterations of `shrink` from the initial strategy $\sigma_{\texttt{triangle}}(m)$ eliminate the points with $x > x_{\texttt{dec}}$ or $y > y_{\texttt{dec}}$ until the strategy $\sigma_{\texttt{rect}}(x_{\texttt{dec}}, y_{\texttt{dec}})$ is considered. Consequently the solutions returned by `shrink` from initial strategy $\sigma_{\texttt{triangle}}(m)$ and from $\sigma_{\texttt{rect}}(x_{\texttt{dec}}, y_{\texttt{dec}})$ are identical.

▶ Remark. Another heuristic, symmetric to `shrink`, was introduced in [23]. 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 Appendix A 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 [23]. 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 Appendix A 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 5.1. 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 11.** 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 [23] 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: $S \leftarrow \{(x, y) \mid$ (x,y) is reachable$\}$
- line 6: **while** $S \neq \emptyset$ and `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 Appendix A):

▶ Proposition 5.2. The probabilistic strategy returned by `shrinkp` is optimal, and has a single probabilistic point.

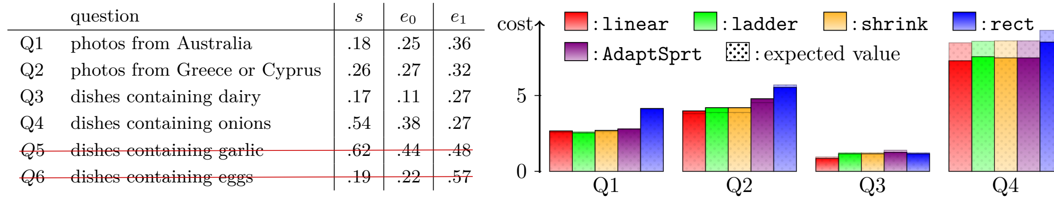
This result sheds a new light on the `shrink` algorithm, but it can also be used to leverage the running time of `shrink` and the linear program, since `shrink` and `shrinkp` coincide at any step until the last iteration of `shrink` as we discuss in Appendix A.

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. We only present a small sample of our experiments here and leave details for Appendix B. Those experiments show that some linear 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 6 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 [6] crowdsourcing game platform, developed in our lab.

The error rates, summarized in Figure 6, 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 $Q4$).



	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 $m = 12$, $\tau = .1$)

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.

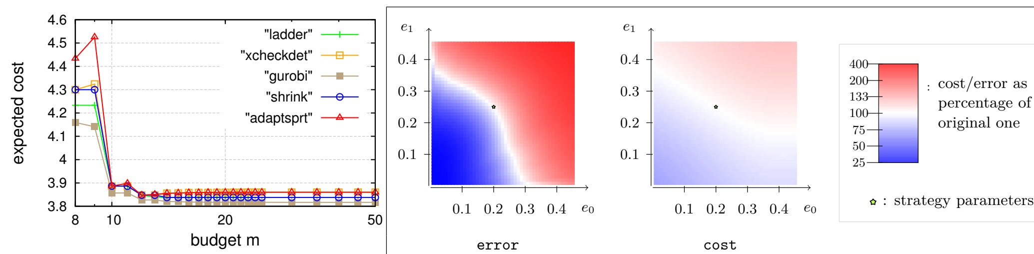


Figure 3 For $e_0 = .2, e_1 = .25, \tau = .05, s = .6$: cost, and sensitivity (only for `shrink` with $m = 15$).

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 [23], 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 [23] revisited in this paper. Parameswaran et al. have reviewed in [23] 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 [24]: 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 [4] and budget constraint [11]; or on whether questions are issued one at a time or in batches [16]. 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 [12] 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 [18] 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 [17], [7], [26] 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 [3] to singular value decomposition [14], Bayesian learning [25] and variational inference in graphical models [18]. 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. [14] 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) [25, 18]. 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 [6, 8, 13].

Finally, empirical studies show that batching tasks may have positive impact on CrowdSourcing efficiency [20]. Similarly, pre-recruiting schemes [5], 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 http://www.joyofkosher.com/meal-course/course-main/?sbv=2, retrieved 09/13.
2 Mechanical Turk. http://mturk.com.
3 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.
4 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.
5 Michael S. Bernstein, David R. Karger, Robert C. Miller, and Joel Brandt. Analytic methods for optimizing realtime crowdsourcing. In *Collective Intelligence*, 2012.
6 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.
7 Peng Dai, Mausam, and Daniel S. Weld. Decision-theoretic control of crowd-sourced workflows. In *AAAI*, 2010.
8 Nilesh N. Dalvi, Aditya G. Parameswaran, and Vibhor Rastogi. Minimizing uncertainty in pipelines. In *NIPS*, pages 2951–2959, 2012.
9 Jacques Dutka. The incomplete beta function — a historical profile. *Archive for History of Exact Sciences*, 24(1):11–29, 1981.
10 Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.
11 Peter Frazier and Angela J. Yu. Sequential hypothesis testing under stochastic deadlines. In *NIPS*, 2007.
12 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.
13 Haim Kaplan, Ilia Lotosh, Tova Milo, and Slava Novgorodov. Answering planning queries with the crowd. *PVLDB*, 6(9):697–708, 2013.
14 David R. Karger, Sewoong Oh, and Devavrat Shah. Efficient crowdsourcing for multi-class labeling. In *SIGMETRICS*, pages 81–92, 2013.
15 Donald E. Knuth. *The Art of Computer Programming, Volume IV, draft of 7.2.1.6.* Addison-Wesley, 2004.
16 Walter Lehmacher and Gernot Wassmer. Adaptive sample size calculations in group sequential trials. *Biometrics*, 55(4):1286–1290, 1999.
17 Christopher H. Lin, Mausam, and Daniel S. Weld. Dynamically switching between synergistic workflows for crowdsourcing. In *AAAI*, 2012.
18 Qiang Liu, Jian Peng, and Alexander T. Ihler. Variational inference for crowdsourcing. In *NIPS*, pages 701–709, 2012.
19 Adam Marcus, David R. Karger, Samuel Madden, Rob Miller, and Sewoong Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.
20 Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.
21 Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.
22 Aditya G. Parameswaran. personal communication.
23 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.
24 A. Wald. Sequential tests of statistical hypotheses. *The Annals of Math. Stat.*, 16(2):pp. 117–186, 1945.

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

26 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.

8 Appendix A

Number of ladder strategies

Lemma 7: *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.*

Proof. Every ladder strategy can be viewed as a pair of paths from $(-1,-1)$ to $(x_{\texttt{dec}}, y_{\texttt{dec}})$: one for the upper ladder, and one for the lower one. The number of upper paths can be bounded by the number of possible paths with $x_{\texttt{dec}}$ rightward and $y_{\texttt{dec}}$ upward moves:

$$\binom{x_{\texttt{dec}} + y_{\texttt{dec}}}{x_{\texttt{dec}}} \leq \binom{m}{m/2} \sim 2^m \sqrt{\frac{2}{m\pi}} \quad (\text{as } m \to +\infty)$$

The number of ladder strategies is therefore bounded by

$$2^{2m} * 2/(m\pi) \in O(2^{2m}/m)$$

◀

This is an overapproximation, yet a fairly accurate one:

▶ **Lemma 12.** *For* $s = 0.5, e_0 = e_1$*, the number of ladder strategies is* $\Omega(2^{2m}/m^3)$.

Proof. This is because in that case $x_{\texttt{dec}} = y_{\texttt{dec}} = m/2$ and so the number of strategies is at least the number of ways to combine a ladder from $(-1, 1)$ to $(m/2-2, m/2)$ staying above $y = x + 2$ with a ladder from $(1, -1)$ to $(m/2, m/2 - 2)$ staying below $y = x - 2$, where $N = m/2$. These combinations do not take into account all possible strategies, so we only get a lower bound on the number of strategies. Nevertheless these combinations guarantee that the upper ladder and lower ladder do not meet, which simplifies the computation since we then simply multiply the number of upper and lower ladders. The number of such ladders is given by the Catalan numbers, namely, if we set $N = m/2 - 1$:

$$\frac{1}{N+1}\binom{2N}{N} \sim \frac{4^N}{N^{3/2}\sqrt{\pi}}$$

hence a bound of $\Omega(2^{2m}/m^3)$ strategies to consider. ◀

Complexity of ladder enumeration

We prove in this section that our implementation of `ladder` meets the complexity bound claimed in Proposition 8:

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

Implementation details

For optimization purposes, we adopt a different order depending on whether the slope of the decision line is greater than 1 ($e_0 \geq e_1$) or not. In this proof we only consider the case $e_0 \geq e_1$ since a symmetric algorithm covers the case $e_1 > e_0$. We consider strategy $(\mathtt{up}, \mathtt{down})$ to be smaller than $(\mathtt{up}', \mathtt{down}')$ if $\mathtt{up} <_{lex} \mathtt{up}'$, or $\mathtt{up} = \mathtt{up}'$ and $-\mathtt{down} <_{lex} -\mathtt{down}'$. In other words, we follow a lexicographic order on $(\mathtt{up}, -\mathtt{down})$, where $-\mathtt{down}$ is the array obtained from `down` when multiplying every value by -1. In this case, the initial ladder strategy has both ladders tightly close to the decision line, and the final strategy is the rectangle $\sigma_{\mathtt{rect}}(x_{\mathtt{dec}}, y_{\mathtt{dec}})$. We say that `down` is modified at column i if $\mathtt{down}(i)$ is decremented and all $\mathtt{down}(j)$ $(j > i)$ are reset to their maximal possible value below the decision line, with $\mathtt{down}(j)$ staying the same for all $j < i$. At each enumeration step, only two cases may occurr: either the upper ladder is modified and the lower ladder is “reset” to the minimal one, or the lower ladder only is modified at some column i.

We compute once and for all the values of all $S_0(x, y)$, $S_1(x, y)$, and store them in a matrix. We also precompute a function returning the lowest point above the decision line, *decLine*, such that $decLine(x) = \min\{y \mid S_1(x, y) \geq S_0(x, y)\}$. This value can anyway be computed in constant time based on section 2.2 since $S_1 > S_0$ iff $g_1/g_0 > 1$. We then maintain the value of $Path(x, y)$ in a matrix and represent the upper and lower ladders as a pair $(\mathtt{up}, \mathtt{down})$ of arrays with size $x_{\mathtt{dec}} + 1$, where $\mathtt{up}(i)$ records the lowest terminating point (x, y) of the upper ladder ladder with $x = i$, and $\mathtt{down}(i)$ records the corresponding highest terminating point of the lower ladder.

In order to compute incrementally the cost and error of strategies when the ladders are modified we also store two arrays `errorTill` and `costTill`, where $\mathtt{errorTill}(i)$ records the partial sum corresponding to E restricted to the points with $x \leq i$, and symmetrically with `costTill` for C. When the strategy is modified only from column i, i.e., only on points with $x > i$ for some $i \geq 1$, a careful analysis of the equations shows that we can use the previous value of $Path(i, y)$, $\mathtt{errorTill}[i]$ and $\mathtt{costTill}[i]$ to compute in $O((m - i) \times m)$ the error, cost, and corresponding new matrices and arrays for the new strategy.

Complexity Analysis

We show that ladder strategies can be enumerated efficiently and evaluated in amortized cost $O(m)$ thanks to incremental computation. Let us fix a given upper ladder and consider the enumeration of all corresponding lower ladders. We first show that on average only the k last columns are modified. The intuition is quite simple: `down` is most frequently modified at column $y_{\mathtt{dec}}$, less frequently at $y_{\mathtt{dec}} - 1$, etc.

Every time `down` is modified at some column $j \leq x_{\mathtt{dec}} - i$, $\mathtt{down}(i)$ is then reset to its maximal value $decLine(i)$ in the resulting ladder (and similarly for all $i' \geq i$. Consequently, the number of times `down` is modified at some column $j \leq x_{\mathtt{dec}} - i$ is exactly the number $L(i)$ of possible lower ladders up to $(i, decLine(i))$. On average, `down` is thus modified at column $x_{\mathtt{dec}} - 1 - (L(x_0) + L(2) + \cdots + L(x_{\mathtt{dec}} - 1))/L(x_{\mathtt{dec}})$ where x_0 is the smallest i such that $decLine(i) > 0$. As we assumed the slope of the decision line to be greater than 1, $decLine(i+1) \geq decLine(i)+1$, and therefore we claim that $L(i+1) \geq 2 \times L(i)$ for all $i \geq x_0$. To prove the claim, let us denote by $L(x, y)$ the number of possible lower ladders up to (x, y). Then $L(i) = L(i, decLine(i) - 1)$ and $L(i + 1) = L(i + 1, decLine(i + 1) - 1) = \cdots = L(i + 1, decLine(i))$. Furthermore, $L(i+1, decLine(i)) = L(i, decLine(i)) + L(i+1, decLine(i) - 1)$ and $L(i + 1, decLine(i) - 1) \geq L(i, decLine(i) - 1)$. Combining those equations, we get $L(i+1) \geq 2 \times L(i)$. We thus get $(L(x_0) + L(x_0+1) + \cdots + L(x_{\mathtt{dec}} - 1)) \leq (\sum_k 1/2^k) \times L(x_{\mathtt{dec}})$

so that, on average, `down` is modified in the last two columns only. The cost of evaluating the corresponding strategy is thus $O(2 * m)$.

Overall, there are $O(2^m/\sqrt{m})$ upper ladders to consider, according to Lemma 7. For each such ladder, the initial evaluation has cost m^2, and the next $O(2^m/\sqrt{m})$ lower ladders are each evaluated in linear time. The corresponding complexity is therefore $O(2^m/\sqrt{m} \times (m^2 + 2 \times m \times 2^m/\sqrt{m})) = O(2^{2m})$.

Algorithm `shrink`

Proposition 4.3: *The first iterations of* `shrink` *from the initial strategy* $\sigma_{triangle}(m)$ *eliminate the points with* $x > x_{dec}$ *or* $y > y_{dec}$ *until the strategy* $\sigma_{rect}(x_{dec}, y_{dec})$ *is considered. Consequently the solutions returned by* `shrink` *from initial strategy* $\sigma_{triangle}(m)$ *and from* $\sigma_{rect}(x_{dec}, y_{dec})$ *are identical.*

Proof. Every point (x, y) with $x > x_{\texttt{dec}}$ has an infinite Cost/Error ratio since the ultimate decision is "accept" so that deciding immediately has no impact on error while decreasing cost. Symmetrically, every point with $y > y_{\texttt{dec}}$ also has an infinite Cost/Error ratio. Contrariwise, points with $x < x_{\texttt{dec}}$ and $y < y_{\texttt{dec}}$ have a finite ratio Cost/Error during those first iterations since the ultimate decision may still change after that point, so that advancing the decision at (x, y) strictly increases the error. Points with $x > x_{\texttt{dec}}$ or $y > y_{\texttt{dec}}$ are thus shrinked first. ◀

Growth

We next show how to remedy a major shortcoming of the `growth` algorithm from [23]. The `growth` heuristic proposed in [23] is essentially symmetric to `shrink`. This algorithm starts with the initial strategy σ_{null} asking 0 questions: all points are initially terminating. For each terminating point, we calculate the change in cost ΔC and error ΔE obtained when turning that point into a continuing point. We then turn the point with the smallest ratio $-\frac{\Delta C}{\Delta E}$ into a continuing point, and repeat this step until the error drops below the threshold. This heuristic may also terminate when all terminating points have $\Delta E = 0$, however, which justifies why infeasible strategies are sometimes returned by the `growth` algorithm from [23]. We next show that a more careful initialization of the strategy staves off the problem. Specifically, we show that one should use $\sigma_{\texttt{rect}}(x_0, y_0)$ as the initial strategy, where (x_0, y_0) intuitively denotes the closest point to the origin on the decision line.

More formally, we define (x_0, y_0) as $(0, 0)$ if strategy σ_{null} is feasible, and otherwise let $x_0 \geq 1, y_0 \geq_1$ be such that $(x_0 - 1, y_0)$ is accepting, $(x_0, y_0 - 1)$ is rejecting, and $x_0 + y_0$ is minimal. Remark 2.2 guarantees that (x_0, y_0) is unique, and even that $x_0 = 1$ or $y_0 = 1$. The following proposition shows that $\sigma_{\texttt{rect}}(x_0, y_0)$ is a better initial strategy for `growth` than σ_{null}.

▶ Proposition 8.1. When the `growth` heuristic returns a feasible strategy from the initial strategy σ_{null}, it returns the same strategy from $\sigma_{\texttt{rect}}(x_0, y_0)$. Furthermore, the `growth` heuristic always returns a feasible strategy (on feasible instances) from the initial strategy $\sigma_{\texttt{rect}}(x_0, y_0)$.

Proof. If `growth` returns a feasible strategy, then $\Delta E < 0$ for point $(0, 0)$ in σ_{null}, hence $(0, 1)$ is accepting and $(1, 0)$ is rejecting. Consequently, $x_0 = y_0 = 1$ so that $\sigma_{\texttt{rect}}(x_0, y_0)$ is the strategy obtained after one iteration of `growth`: the final strategies returned in both cases are therefore the same. With our initialization, one can easily prove that at any stage of the algorithm the strategy contains a terminating point (x, y) such that $\Delta_E > 0$, until

the strategy equals $\sigma_{\texttt{rect}}(x_{\texttt{dec}}, y_{\texttt{dec}})$, so that the `growth` heuristic always returns a feasible strategy if there is one. ◀

Randomization is limited

We show here the existance of an optimal probabilistic strategy with a single probabilistic point. For this, we show in the next two lemmas that in any probabilistic strategy, we can transfer probabilities between specific probabilistic points without increasing the expected error and cost of the strategy.

For any reachable point (x, y) of a given strategy, let us denote by $\gamma(x, y)$ the ratio $-\Delta_{\text{Cost}}(x, y)/\Delta_{\text{Err}}(x, y)$ between cost and error modification when the probability to continue at (x, y) is modified. We first present a simple lemma.

▶ **Lemma 13.** *Let (x, y) and (x', y') be two points such that (x, y) is not reachable from (x', y'). Modifying the strategy in (x, y) has no impact on $\Delta_{Err}(x', y')$ and $\Delta_{Cost}(x', y')$.*

Proof. Equations 4 only involve points that are reachable from (x, y). ◀

We exploit this technical lemma to establish the following tranfer property:

▶ **Lemma 14.** *Let (x_1, y_1), (x_2, y_2) denote reachable points with Cost/Error ratios $\gamma_1 \geq \gamma_2$ in some strategy σ. There exist $\delta_1, \delta_2 \geq 0$ such that increasing $P_{stop}(x_1, y_1)$ by δ_1 and reducing $P_{stop}(x_2, y_2)$ by δ_2 yields a strategy σ' satisfying both (1) $P_{stop}(x_1, y_1) = 1$ or $P_{stop}(x_2, y_2) = 0$, and (2) the cost and error of σ' are smaller or equal to those of σ. If $\gamma_1 < \gamma_2$, we can even obtain a strictly smaller cost.*

Proof. If $\Delta_{\text{Err}}(x_1, y_1)$, the result is obvious. Otherwise, let us first assume that (x_1, y_1) is not reachable from (x_2, y_2). Let us add some $\delta > 0$ to $P_{stop}(x_1, y_1)$, and let Δ'_{Cost}, Δ'_{Err} and $Path'$ denote the functions mapping each point to their new variation rates and number of paths after this operation. According to Lemma 13, $\Delta'_{\text{Err}}(x_2, y_2) = \Delta_{\text{Err}}(x_2, y_2)$. Assume that $Path'(x_2, y_2) \times \Delta'_{\text{Err}}(x_2, y_2) \neq 0$, and remove $\delta_2 = \delta \times \frac{Path(x_1,y_1)\times\Delta_{\text{Err}}(x_1,y_1)}{Path'(x_2,y_2)\times\Delta'_{\text{Err}}(x_2,y_2)}$ from $P_{stop}(x_2, y_2)$. The admissible values of δ are those for which the modified values of P_{stop} remain in $[0, 1]$ at both points. Over the course of the process above, whatever error is added through point (x_1, y_1) is removed in point (x_2, y_2), so the the process preserves the error rate of the strategy. In addition, the expected cost of the strategy does not increase as it varies by $\delta \times Path(x_1, y_1) \times \Delta_{\text{Err}}(x_1, y_1) \times (\gamma_2 - \gamma_1) \leq 0$. The cost even strictly decreases when $\gamma_2 - \gamma_1 > 0$. The case $Path'(x_2, y_2) \times \Delta'_{\text{Err}}(x_2, y_2) = 0$ can be handled by continuity. When (x_1, y_1) is reachable from (x_2, y_2), (x_2, y_2) is not reachable from (x_1, y_1) so we obtain a symmetric proof by first decreasing $P_{stop}(x_2, y_2)$ and then increasing $P_{stop}(x_1, y_1)$ by the appropriate amount. ◀

Using Lemma 14, we next prove that in any strategy with several probabilistic points, the number of probabilistic points can be reduced without increasing the cost and error.

Proposition 5.1: *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.*

Proof. Using the transfer property from Lemma 14, we can repeatedly reduce the number of probabilistic points of a strategy having several such points until at most one remains. We prove the second part of the claim by contradiction. Assume that some probabilistic point of an optimal strategy has ratio $\gamma(x_1, y_1)$ strictly smaller than another point (x_2, y_2) satisfying $cPath(x_2, y_2) > 0$. Then we can obtain a strategy with smaller error and strictly

smaller cost, according to the transfer property above. This contradicts the optimality of the strategy. ◀

Optimality of shrinkp

Proposition 5.2: *The probabilistic strategy returned by algorithm* `shrinkp` *is optimal.*

sketch. We fix some parameters e_0, e_1, τ, m, s, and deduce the corresponding values of $x_{\texttt{dec}}, y_{\texttt{dec}}$ as in Section 2.3. If the trivial strategy is feasible, the result is obvious. Otherwise, we first observe that in `shrinkp` and in any optimal probabilistic strategy, the error is exactly τ. Let $C_{\texttt{shrink}}$ be the cost of the `shrinkp` strategy. Let σ denote a feasible strategy with error τ. Let k the number of points on which σ differs from the `shrinkp` strategy. We show by induction on k that the cost C_σ of σ is at least $C_{\texttt{shrink}}$

Let (x_0, y_0) denote the first point, in the order of "elimination" by `shrinkp`, on which σ and `shrinkp` disagree. Let also S_0 denote all the points visited before (x_0, y_0) by `shrinkp`, and S_1 the remaining points. We have thus partitioned the set of all points into $S_0 \uplus S_1 \uplus \{(x_0, y_0)\}$, and σ and `shrinkp` agree on all points of S_0. By definition of `shrinkp`, (x_0, y_0) is the maximal ratio in the strategy σ_0 that terminates on all points of S_0 and continues on others (except those with $x = x_{\texttt{dec}}$ or $y = y_{\texttt{dec}}$). Let Strat denote the set of all strategies σ' that satisfy the following three conditions: (1) σ' agrees with σ on S_0 (2) $\sigma'(x_0, y_0) = \texttt{shrinkp}(x_0, y_0)$, and (3) $\sigma'(x, y) \leq \sigma(x, y)$ for every $(x, y) \in S_1$. We wish to prove that there exists a strategy in Strat with error exactly τ and cost no higher than C_σ.

We first study the maximal and minimal error achieved by strategies in Strat. By definition of `shrinkp`, the probability of terminating at (x_0, y_0) is higher in `shrinkp` than in σ. Consequently, the strategy in Strat that agrees with σ on S_1 has error higher than σ, hence than τ. On the other hand, the strategy in Strat that continues with probability 1 on all points of S_1 except those with $x = x_{\texttt{dec}}$ or $y = y_{\texttt{dec}}$ has error lower than `shrinkp`, hence than τ. What is more, the error is a continuous function of the strategy, and Strat is a connex set. Therefore, there is a strategy σ' in Strat with error exactly τ. We claim that this strategy is at least as efficient as σ.

Claim: the expected cost of σ' is at most that of σ.

According to this claim, there is a feasible strategy, σ' with cost at most C_σ that differs with `shrinkp` on at most $k - 1$ points, which concludes our proof by induction.

Let us prove the claim. By construction of Strat, both strategies σ and σ' can be obtained from σ_0 by increasing the probability of terminating at (x_0, y_0) and some points of S_1. The increase is larger at (x_0, y_0) and smaller at all other points for σ' w.r.t. σ. Furthermore, the ratio of (x_0, y_0) in σ_0 is higher than the ratio of any point in S_1. As a consequence, the ratio between cost and error for strategy σ is no smaller than this ratio for σ'. We conclude that the cost of σ is at least the cost of σ'. We next provide a more formal proof of this: let $N = |S_1| + 1$, and let $r_1, r_2, \ldots, r_N$ denote the Cost/Error ratios in σ_0 of the points from $S_1 \uplus \{(x_0, y_0)\}$, by increasing order of $x + y$ (with ties ordered by increasing x, for instance). We denote by j the index of (x_0, y_0) in this ordered sequence. For each $0 \leq i \leq N$, we also denote by E_i (resp E'_i) the error of the strategy which agrees with σ (resp σ') on the i first points in the above order, and agrees with σ_0 on all other points. For each $1 \leq i \leq N$, we then define $\delta_i = E_i - E_{i-1}$ and $\delta'_i = E'_i - E'_{i-1}$. According to Lemma 13, the cost of σ differs from the cost of σ_0 by $\sum_i r_i \delta_i$, whereas the cost of σ' differs from the cost of σ_0 by $\sum_i r_i \delta'_i$. What is more, $\sum_i \delta_i = \sum_i \delta'_i$, $\delta_j \leq \delta'_j$, and for all $i \neq j$ we have $\delta_i \geq \delta'_i$ while $r_j > r_i$. The result follows. ◀

▶ Remark. We observe that in spite of its similarities with `shrink`, Algorithm `growth` does not extend so easily into an optimal probabilistic algorithm. One may indeed wish to consider the algorithm that starts with the initial strategy $\sigma_{\texttt{rect}}(x_0, y_0)$ presented above Proposition 8.1 and then in each iteration maximizes the probability to continue at the point minimizing the Cost/Error ratio while preserving the error above τ. But this algorithm does not guarantee an optimal probailistic solution.

Deterministic vs Probabilistic Strategies

One shows easily that an optimal deterministic solution will be an optimal probabilistic solution only if it has error exactly τ, which has probability 0 when parameters are random floats, or if the trivial strategy is feasible. In our experiments, the cost was indeed strictly lower for the probabilistic strategy whenever the trivial strategy was not feasible ($s \in]\tau, 1-\tau[$). We actually observed in our experiments that the trivial strategy is feasible in $\approxeq 10\%$ of all instances, and show in the remark below that this proportion is coherent with theory.

The small sample size in [23] could explain slight differences in figures, but does not explain why the proportion of strictly better instances does not steadily increase with budget in [23, Figure 5a]. We assume the discrepancy might be due to rounding approximations in [23, Figure 5a], since the authors do not detail how they round parameters and compare floats.

▶ Remark. For our choice of random parameters ($s \in [0, 1]$ and $\tau \in [0.005, 0.1]$), the probability of getting a trivial instance is 0.105.

Proof. We need to compute the probability p of having $s \leq \tau$ or $1 - s \leq \tau$. This is clearly twice the probability that $s \leq \tau$, hence:

$$p = 2 \int_{0.005}^{0.1} \Pr(\tau = t) \times \Pr(s \leq t)\, \mathrm{d}t$$

and so for our uniform sampling we get

$$p = 2 \int_{0.005}^{0.1} (1/.095) t\, \mathrm{d}t = [t^2/.095]_{0.005}^{0.1}$$

We conclude that $p = 0.105$. ◀

9 Appendix B

We complement our theoretical study with an experimental evaluation of the algorithms. We contrast the running time and robustness of our algorithms, and compare the expected cost of the strategies produced. Sections 9.2,9.3 and 9.4 average measures over numerous synthetic random instances, Section 9.5 investigates particular parameters, and finally Section 9.7 reports on some experiment with real crowd answers. Figure 4 (a), (b) and (d) and Figure 5 use logarithmic scales to help distinguish the curves and emphasise small budgets.

9.1 Experimental Settings

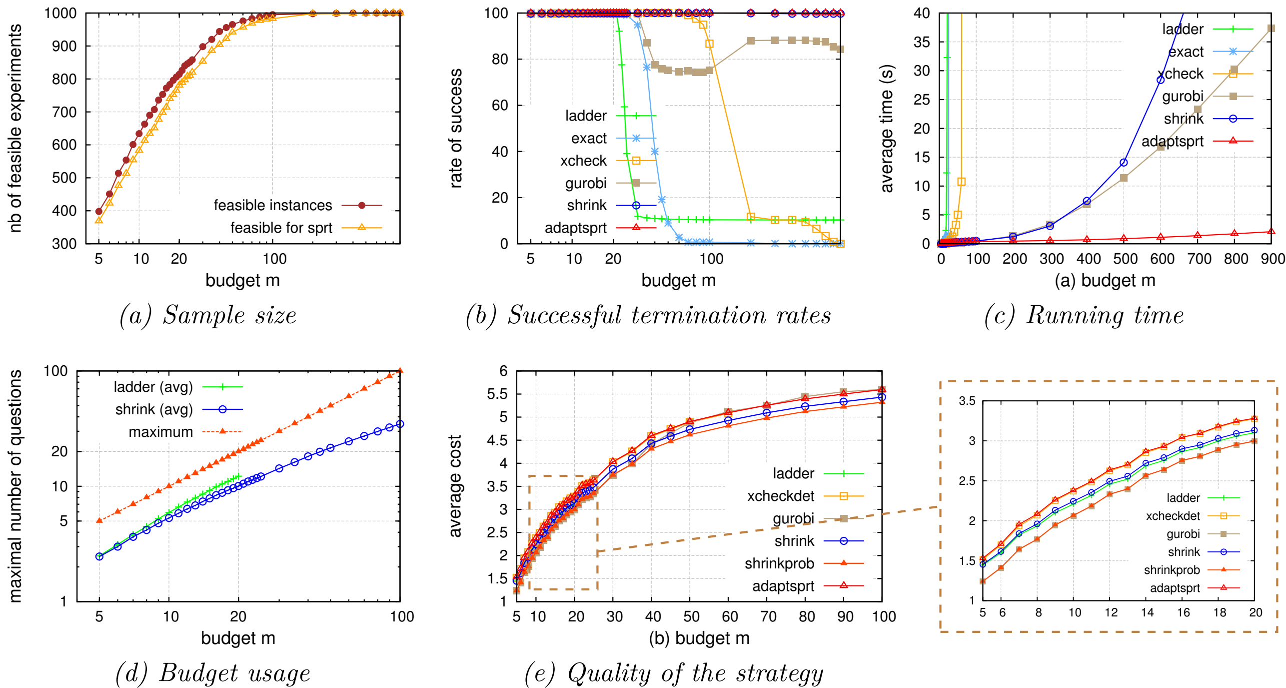
Algorithms

Experiments in this section evaluate all algorithms described in this paper: `AdaptSprt` from Section 3, `ladder` and `shrink` from Section 4, and `shrinkp`, as well as the linear

programming approach from Section 5. The performances of `growth` and `shrink` are quite similar, so we do not detail the heuristic further in this paper. The performance of the linear programming approach is sensitive to the solver and so we compared three solvers, denoted by `xcheck`, `exact` and `gurobi`, and we denote by `linear` the linear program algorithms when there is no reason to distinguish solvers. Our `gurobi` program uses the commercial Gurobi optimizer, reported to be one of the fastest, with default parameters. Our `xcheck` and `exact` programs use the GLPSol solver from the GNU Linear Programming kit, with options “xcheck” and “exact” respectively, and with other parameters kept to their default values. The “exact” option implements the simplex algorithm with exact (rational) arithmetic, whereas “xcheck” first uses the simplex with floating point arithmetic, but checks the final basis using exact arithmetic and performs a few more simplex iterations if the solution is not optimal. As a consequence, the Gurobi solver is faster but less accurate than GLPSol with “exact” or “xcheck” options.

Environment

We implemented all algorithms in Python, using the default options in PyPy 1.8.0 and CPython 2.7.3. The linear solvers Gurobi and GLPSol are written in C, but the program itself was written in Python and issued calls to the solvers using the default options of the PuLP interface. The experiments were run on an Intel i7-2600 CPU using a 32bit Ubuntu 12.04 system. All experiments were planned such that each CPU would run a single experiment at a time.



(a) Sample size *(b) Successful termination rates* *(c) Running time*

(d) Budget usage *(e) Quality of the strategy*

Figure 4 Evaluating algorithms on random instances

Parameters

We sample uniformly the parameters, represented as floats (not-rounded). In order to compare with previous work [23], we sample

- the error rates e_0, e_1 between 0.05 and 0.45.
- the average error tolerated τ, between 0.005 and 0.1
- the selectivity s in $[0, 1]$.

For each value of the parameters, we run all algorithms on increasing values of m: $5, 6, \ldots, 25$, then $30, 35, \ldots, 50$, 60, $70, \ldots, 100$, and finally $200, 300 \ldots, 1000$. We recall that we only consider feasible instances. We point out like [23] that in general a set of parameters e_0, e_1, τ, s will not be feasible for a small value of m, but will become so beyond a certain value m_0 of m, as evidenced in Figure 4(a). The corresponding instances with $m < m_0$ are dropped and those with $m \geq m_0$ are preserved into our aggregated results. As a result, the average expected cost of the strategies will generally increase with m since harder strategies become feasible.

Timeout

Some algorithms did not return any solution after several days for some values of the parameters with large budgets, so we decided to interrupt experiments after a timeout of 5 min. When an algorithm fails to return a solution for some value of the parameters on $m = m_0$, we skip the evaluation of the algorithm for those parameters on all $m > m_0$.

9.2 Success Rates

In this section, we first discuss why an algorithm might fail to return a reasonably good strategy and then proceed to investigate the limitations of specific algorithms.

Limiting Factors

The algorithms may fall short of their formal specification on several grounds. First, floating point arithmetics may produce inaccurate results, as many numeric approximations are combined along the calculation of error and cost, which may affect the choice of strategy by our algorithms. This problem may often be staved off by using high-precision numeric libraries, such as `mpmath` for python, but this quick fix increases the running time by a large factor. Another issue is the running time, as we showed that many of the algorithms have a high complexity. Due to limited hardware resources, large budgets can cause these algorithms to run out of time or memory.

Failure Rates

Figure 4(b) measures how often our algorithms fail to return correct strategies. We consider the output as incorrect when either the output strategy is infeasible or the algorithm does not terminate within 5 minutes. In addition, for all optimal probabilistic algorithms, we also consider any solution whose cost was at least 0.01 larger than the cost of the optimal solution to be a deviation from the expected result. The Figure attests that `AdaptSprt` was successful on all instances. The same holds for the optimized `shrink` algorithm, except on 3 instances where it failed due to floating point approximations. Our Gurobi program suffers even more from numerical instability as some output strategies were infeasible while others were suboptimal from $m = 30$. Gurobi also ran out of memory for $m > 900$. The `ladder`,

exact and xcheck programs systematically timed out for moderate budgets, around $m = 20$, 40, and 80 respectively, but otherwise returned optimal strategies.

9.3 Time and Cost Performance

We next evaluate the running time of each algorithm, and the expected cost of the strategies produced.

Running Time

Figure 4(c) measures the average running time of each algorithm as a function of the budget. The running times of deterministic algorithms are coherent with their theoretical complexity, and in particular AdaptSprt is clearly the fastest. The running times of probabilistic algorithms depend heavily on the choice of solvers. Unlike exact and xcheck, the gurobi solver is generally quite efficient and returns the solution within half a minute for $m \leq 900$. Finally we obtained a similar graph when considering the worst instances instead of averaging the running time.

Quality of the Strategy

The expected cost of the strategies are reproduced on Figure 4(e). All algorithms yield similar costs, with a 8% gap between the optimal and worst one. As expected the optimal probabilistic solutions shrinkp, exact, and xcheck have lowest cost. Next comes the optimal deterministic solution ladder with the shrink heuristic close behind, while AdaptSprt is slightly worse. The gurobi solver, however, often provides grossly suboptimal solutions, due to numerical instability. Contrary to what was observed in [23, Figure 5a], the optimal probabilistic solution always improved upon its deterministic counterpart except for trivial instances (strategies terminating at $(0, 0)$). We elaborate in the section “Deterministic vs Probabilistic Strategies” why our numbers make more sense.

We next compare our sequential strategies with naïve strategies that fix the sample-size, i.e., with the optimal rectangle strategy rect. The cost of the optimal rectangle strategy is up to 5 times, and on average 1.5 larger than the cost of shrink, and the gap increases with the budget from 1.2 when $m = 5$ to 1.8 when $m > 300$.

9.4 Shape of the Strategies

Some properties about the decision point (e.g., that it lies closer to the y-axis when s is small [23]) derive from its location across the decision line. But it is more of a surprise that optimal strategies generally do not use the full budget available. By definition, AdaptSprt strategies use the full budget, but Figure 4(d) shows that most ladder and shrink strategies do not. In this Figure, we plot for all m the maximal number of questions issued by an *average* ladder (resp. shrink) strategy with budget m. The maximal number of questions issued by *the most demanding* shrink strategy with budget m from our sample coincides with the budget (red curve; the curve for ladder turned out to coincide, so was omitted). Hence some ladder and shrink strategies use the maximal budget m, but on the other hand an average strategy only uses half. What is more, shrink uses a slightly smaller part of the budget on average than ladder. Further experiments show that shrink does not shrink many more points than shrinkp: on average, for a budget m, the shrink and shrinkp strategies only differ on $m/2$ points.

9.5 Fixed Parameters

We also monitored the running time and cost for specific parameters, when only m varies. The figures were similar for the running time, but not for cost, which tends to decrease with the budget when other parameters are fixed.

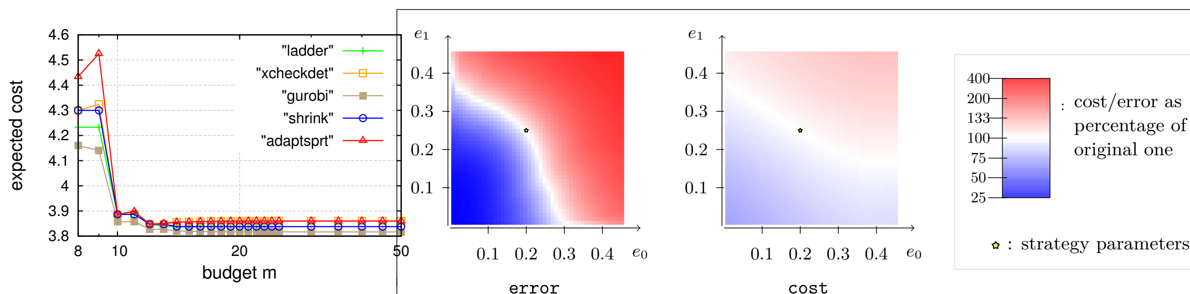


Figure 5 Cost of strategies with $e_0 = .2, e_1 = .25, \tau = .05, s = .6$, and sensitivity (for `shrink` only, with $m = 15$).

Quality of the Strategy

Figure 5 presents the expected cost of the strategies returned by each algorithm for varying m while all other parameters are fixed to $e_0 = 0.2, e_1 = 0.25, \tau = 0.05, s = 0.6$, which allows to compare with the similar experiment in [23]. We observe that the cost decreases sharply during the first few steps but stabilizes beyond $m = 14$. The relative order of the strategies is similar to their order in the aggregated results. Surprisingly, the costs for `shrink` do not match the ones reported in [23, Figure 4a]: our `shrink` strategy is more efficient than theirs, and in our experiment the cost of `shrink` decreases with m. Actually the cost of our `shrink` strategy matches the cost they report for `growth`. We cannot explain why because they only provide a brief explanation of `shrink`'s behaviour.

Sensitivity analysis

The performance of the algorithms varies with all parameters s, e_0, e_1, τ, m. The variations of cost with s, e_0 and e_1 were surveyed in [23] and our experiments only confirmed theirs. We therefore focus in this paper on the variations with m. However, the exact value of s, e_0, e_1 may not be available in real scenarios, so we also render in Figure 5 the performance of a strategy when the parameters of the data and questions are slightly different from the parameters used to build the strategy. In that picture the color at (x, y) measures the expected cost and error that would be obtained if the strategy computed by `shrink` for $e_0 = .2, e_1 = .25, \tau = .05, m = 15, s = .6$ is used while the real users have error rates $e_0 = x, e_1 = y$ $(x, y \leq .45)$. The values are measured as a percentage of the cost (3.8) and error (.049) obtained when the error rates coincide with the strategy. We observe that error varies much more than cost, and deviations from the expected error rates have limited impact as long as they are small enough. We observed similar trends for other values of the parameters [**?**].

Optimizations

We measured the contribution of our optimizations to the `ladder` and `shrink` heuristics on selected instances. The PyPy compiler is generally above 10 times faster than

cPython. Algorithmic optimizations reduce drastically the running time, especially for `shrink`, and when m gets large. For instance, with the parameters of the running example ($e_0 = .25, e_1 = .2, \tau = .0075, s = .8$) and a budget of $m = 200$, the naïve `shrink` implementation requires over $5h$ with CPython, and 12min with PyPy whereas the optimized implementation completes in 1.5s (with PyPy).

9.6 Classification with multiple choices

This paper is in general focused on binary classification. But most analyses and algorithms can be generalized to classify items in presence of multiple choices. In the general setting we ask the crowd to determine to which class the item belongs, out of n candidate classes. The priors are then given by an n-dimensional vector $\vec{s}$ for selectivity, and an $n \times n$ matrix e for error probabilites, where $e_{i,j}$ denotes the probability that a user answers $V = i$ when the item belongs to class j. We again assume that the correct answer is always the most likely, i.e., $e_{i,i} > e_{j,i}$ when $i \neq j$. The sequence of answers received can then be represented as a walk over the n-dimensional cubic lattice, and most of our analyses remain valid; `shrinkp` still returns the same strategy as the linear program. . . Algorithms `shrink`, `AdaptSprt` generalize naturally to this setting, as well as the linear program of [23].

Due to the dispersion of answers between multiple options, a large budget will generally be necessary to classify items accurately. However, the algorithms scale poorly even with the budgets we considered for binary classification (e.g., $m = 50$); in the worst case, our optimized version of `shrink` indeed has exponential complexity $O(n \times m^{2n})$, whereas `AdaptSprt` runs in $O(n \times m^n)$. We observe that for $m = 4$, our implementation of `shrink` fails to scale beyond $m = 40$, whereas `AdaptSprt` manages budgets up to $m = 50$.

9.7 Experiments with the Crowd

Our synthetic experiments survey the theoretical performance of our algorithms when the users behave as a random oracle specified by e_0 and e_1. We next examine the performance in a real-life crowdsourcing scenario.

Queries and dataset

We experimented with two classes of filtering queries. The first scenario is similar to our motivating example in the Introduction, trying to identify which dishes contain a specific ingredient. For this dataset we randomly selected about 50 dishes from a recipes website [1]. Users were then presented with the picture and name of the dish, and were asked to determine whether it contains the given ingredient (e.g., onions, garlic, etc.).

The second scenario uses the crowd to identify which pictures, out of a given set, are from a given country. The dataset was built from holiday pictures contributed by members of the lab. For privacy, we removed those that contain individuals, which left us with about 100 pictures over which the experiments were run.

To build the ground truth for the first dataset we manually extracted the ingredients of the recipes from the cooking instructions at the Web site, and for the second we asked the contributors to provide the location of their photographs.

Experiment Settings

To avoid spam issues that typically arise in payment-based crowdsourcing platforms like Amazon Mechanical Turk [2], we run the experiments on the AskIt [6] crowdsourcing game

platform, developed in our lab, which queries the crowd using a trivia-like game and can engage users through social network. We modified AskIt's engine so that each question instance is submitted to different players until sufficient answers are gathered. We built the strategies for several accuracy and budget constraints, and then distributed the stream of user answers to all, executing all strategies in parallel so that their behavior can be compared on identical inputs. The crowd of 100 players provided about 10000 answers overall, which provided us with around 35 (resp. 25) answers per question on a given dish-ingredient pair (picture-country pair). The bulk of the answers were gathered withing 48 hours.

Table 1 illustrates the type of questions issued to users and the corresponding parameters computed from the data for each of the filtering tasks. All parameters were computed over the whole dataset, though in a full-fledge system, they would be estimated first on a small sample, but parameter estimation lies outside the scope of this paper, and we refer the reader to the previous section for sensitivity analysis. The error rates 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 therefore discarded the question from our analysis.

	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~~

Table 1 Questions and parameters

To meet a (rather lenient) error threshold of 10% ($\tau = 0.1$) given these parameters, the minimal budget that should be allotted to questions 1,2,3,4 is respectively $7, 8, 2$ and 12. For 5% ($\tau = .05$), the minimal budgets for the same questions are $13, 13, 5, 21$. Q5 has higher error rates and correspondingly requires a budget of almost 300 for $\tau = 0.1$ and 40 for $\tau = 0.3$. With $m = 40$ we indeed obtain for this query an error between 29% and 35% for our algorithms, with average cost around 18 (while `rect` has cost above 26). However, compared to the other questions, these results have lower statistical significance since the error rates for $Q5$ are close to 50%. We therefore focus in the analysis below on the other queries.

We first discuss the strategies created by our algorithms for $\tau = .1$ and $m = 12$, the minimal budget for which Q1-Q4 admit a feasible strategy according to the parameters in Table 1, then examine how larger budgets affect the behavior.

Cost

Figure 6 represents the average number of answers (cost) before the strategies reach a decision on the pictures. On this actual cost, we superimposed the theoretical values (as a pattern in background). The costs roughly match their expected value, yet with some variations. In particular, `ladder` or `linear` did not always outperform the other algorithms, though they generally compare favorably. The figure also shows that the strategies computed by our algorithms outperform the optimal rectangle strategy `rect`. We observed similar

trends for other values of m and τ. In most cases, increasing budget m improves the performance of our strategies. For $m = 20$, for instance, the cost of Q4 drops from 7.5 to ~ 5.7 for all of our algorithms, which justifies using strategies with larger budget bound m, to save on average cost. Here again, as seen in the synthetic experiments, `ladder` runs out of time for $m > 20$.

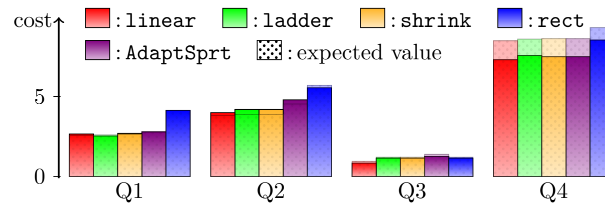


Figure 6 Average cost per item ($m = 12$, $\tau = .1$)

Error Model and Accuracy

As we could expect, real users do not behave as random oracles, which has an impact on the accuracy of our strategies. In our experiment, error rates are not uniform over items, and are not uniform among users either. The standard deviation of the error rates between different pictures is roughly .2, and the standard deviation of error rates between different users is slightly above 0.1.

As a consequence, the average error of the filtering process is not always close to τ. Actually, accuracy varies substantially with the question: instead of .9, algorithm `ladder` yields an accuracy around .87, .8, .9 and .79 respectively for Q1, Q2, Q3 and Q4 when $\tau = .1$ and $m = 12$, and the results are pretty similar for the other algorithms. More surprisingly maybe, the accuracy did not significantly improve when the expected error τ of the strategy was lowered to .05: only for question Q3 was the accuracy improved to .96, whereas the accuracy of other questions culminated below .85. This discrepancy is due to the limitations of the model for this scenario: error rates are not uniform over items, because some pictures contained only little or no clue for the question. Once “easy” items were correctly classified, additional user answers did not help much to classify the harder ones. This issue affects equally any strategy that is applied to all pictures indiscriminately, including the naïve rectangular strategy, and accuracy varies little with the choice of algorithm and budget, provided the budget is large enough to allow feasibility. The error rates also fluctuate between users, but the standard deviation is smaller and has lower impact since additional annotations still help to classify each picture. To sum up, even when parameters allow to build feasible strategies for high precision levels, there is no guarantee that such high precision can indeed be obtained in practice. Nonetheless, using our strategies with large budgets provides substantial savings in the cost of the filtering, which justifies the need for scalability.