Cost and Quality Assurance in Crowdsourcing Workflows
Loïc Hélouët, Zoltan Miklos, Rituraj Singh

To cite this version:
Loïc Hélouët, Zoltan Miklos, Rituraj Singh. Cost and Quality Assurance in Crowdsourcing Workflows. 2020. hal-02964736

HAL Id: hal-02964736
https://inria.hal.science/hal-02964736
Preprint submitted on 12 Oct 2020

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.
Cost and Quality Assurance in Crowdsourcing Workflows

Loïc Hélouët
Inria Rennes
loic.helouet@inria.fr

Zoltan Miklos
Univ. Rennes 1
zoltan.miklos@irisa.fr

Rituraj Singh
Univ. Rennes 1
rituraj.singh@irisa.fr

ABSTRACT
Crowdsourcing platforms provide tools to replicate and distribute micro tasks (simple, independent work units) to the crowd and assemble results. However, real-life problems are often complex: they require to collect, organize, process or transform data, guarantee the quality of results, and meet budget constraints. Further, specifications and realization of intricate jobs on crowdsourcing platforms are still in their infancy.

In this work, we combine workflows with crowdsourcing to enable efficient execution of complex tasks. Workflows provide ways to organize a complex task in phases and guide the overall realization. The challenge is to interface workflows and crowdsourcing systems efficiently to achieve good accuracy of results at a reasonable cost. Standard “static” allocation of work in crowdsourcing affects a fixed number of workers per micro-task to realize and aggregates the results. We propose synchronous and asynchronous dynamic worker allocation techniques on top of workflows, where decisions to replicate tasks, worker allocation and progress of the workflow execution depend on inferred tasks difficulty, worker expertise, confidence in answers and remaining budget. We evaluate the performance of this framework on a benchmark and show that the proposed approaches outperform static allocation in terms of cost and accuracy.

KEYWORDS
Crowdsourcing; Data-centric workflows

1 INTRODUCTION
Despite recent advances in artificial intelligence and machine learning, many tasks still require human contributions. With the growing availability of internet, it is now possible to hire workers all around the world on crowdsourcing marketplaces. Many crowdsourcing platforms have emerged in the last decade: Amazon Mechanical Turk\(^1\), Figure Eight\(^2\), Wirk\(^3\), etc. They hire workers from a crowd to solve problems \([25]\). A platform allows employers to post tasks, that are then realized by workers in exchange for some incentives \([6]\). Common tasks include image annotation, surveys, classification, recommendation, sentiment analysis, etc. \([12]\). The existing platforms support simple, repetitive and independent micro-tasks which require few minutes to an hour to complete.

However, many real-world problems are not simple micro tasks, but rather complex orchestrations of dependent tasks, which aims are to process input data and workers answers. The existing crowdsourcing platforms provide interfaces to execute micro tasks and access crowd, but lack ways to specify and execute complex tasks. The next stage of crowdsourcing is to design systems to realize more complex and involved tasks over existing crowdsourcing platforms. A natural solution is to define complex tasks as workflows, i.e., orchestrations of jobs that exchange data to achieve a final objective \([30]\). The output of individual jobs (documents, annotations, data) is passed to the next tasks according to the workflow rules.

Complex workflows appear in many contexts. Consider for instance online request services frequently proposed by smart cities (Figure 1). These systems receive a huge number of requests that must be filtered, classified and then sent to the appropriate bureau of the city. This requires a lot of work power, but it follows a workflow made of several simple stages. A first step \(p_0\) sorts a load of demands \(D\) according to the address of the client (demands from residents of the city will not be processed like external demands). The demands from city inhabitants are validated and sent to the next phase \(p_1\). The goal of \(p_1\) is to decide in which category (Health, roads, ...) the request falls. After classification, phase \(p_2\) decides whether the demand is Urgent or not if some demands from that category call for rapid response. For categories that are not urgent, demands are forwarded directly to \(p_F\). The final phase \(p_F\) gathers all annotated demands in an annotated dataset \(D_{\text{Annotated}}\).

Workflows alone are not sufficient to handle complex tasks with crowdsourcing. Many data-centric applications come with budget and quality constraints: As human workers are prone to errors, one has to hire several workers to aggregate a final answer with sufficient confidence. An unlimited budget would allow hiring large pools of human workers to assemble reliable answers for each micro-task, but in general, a client for a complex task imposes a limited budget \(B_0\) that must not be exceeded. A limited budget forces to replicate micro-tasks in an optimal way to achieve the best possible quality. The objective is then to obtain a reliable result, forged through a complex orchestration, and at a reasonable cost.

In this paper, we propose a solution for the efficient realization of complex tasks. We define a workflow model, which orchestrates tasks and work distribution according to a dynamic policy that considers confidence in aggregated data and the cost to increase this confidence. A workflow can be seen as an orchestration of phases, where the goal of each phase is to tag records from the dataset input to the phase. A complex task terminates when the last of its phases has completed its tagging. The output of one phase acts as an input to the next ones in the workflow. For simplicity, we consider simple Boolean tagging tasks that associate a tag in \([0,1]\) to every record in a dataset. We also assume that workers are uniformly paid. As humans are prone to errors, tagging must be performed by several workers, and the answers assembled with aggregation technique. For each record, one of the possible answer (called the ground truth) is correct, and an aggregated answer is considered as reliable if its probability to
be the ground truth is high. Hiring more workers to tag records increases the reliability of the aggregated answer. The overall challenge is hence to realize a workflow within budget $B_0$, while guaranteeing that the final dataset forged during the last phase of the workflow has a high probability to be the ground truth.

This general orchestration schema leaves several design choices open. First, the aggregation technique used influences the quality of the final results. Furthermore, the mechanisms used to decide if more workers must be hired to improve quality have an impact on costs and on the overall accuracy of answers. The simplest way to replicate micro tasks is static execution, i.e. affect an identical fixed number of workers to each micro-task in the orchestration without exceeding budget $B_0$. On the other hand, one can allocate workers to tasks dynamically. One can wait in each phase to achieve a sufficient reliability of answers for each record of the input before forwarding data. This is called a synchronous execution of a workflow. Last, one can eagerly forward records with reliable tags to the next phases without waiting for the total completion of a phase. This is called asynchronous execution.

In this work, we study execution strategies for complex workflows with different design choices. We consider several types of workflow, of aggregation (namely Majority Voting (MV) and Expectation Maximization (EM) based techniques [13]) several distributions of data, difficulty of tasks and workers expertise, and study the cost and accuracy of workflows execution under static, synchronous and asynchronous assignment of workers to tasks. Unsurprisingly, dynamic distribution of work allows to save costs in all cases. A more surprising result is that synchronous realization of complex tasks is in general more efficient than asynchronous realization.

Related Work: Several works have considered complex workflows, interactions with crowdsourcing systems, or advanced aggregation techniques to improve data quality. We do not claim exhaustiveness but list below some papers which are the closest to our complex crowdsourcing solution. Coordination of tasks has been considered in many languages such as BPMN [24], ORC [16, 21], BPEL [2, 23], or workflow nets [31], a variant of Petri nets dedicated to business processes. They allow parallel or sequential execution of tasks, fork and join operations to create or merge a finite number of parallel threads. Tasks are represented by transitions. Workflow nets mainly deal with the control part of business processes, and data is not central for this model: Tasks are not replicated, and tokens do not carry data. The more advanced versions of the model [10] handle data through global variables. Similarly, BPMN, ORC and BPEL are centered on the notion of transaction, and are not tailored for datasets manipulations, replication of tasks, nor aggregation of results. To overcome the limitations of orchestration models, several data-centric models have emerged. Guarded Active XML [1] is a specification model where references to external services are introduced in structured data. Services can modify data when their guard is satisfied, and replace a part of the data by some computed value that may contain new references to service calls. Business artifacts were originally developed by IBM [22]. Artifacts and their many variants are data-centric, i.e. they focus on possible modifications of datasets within a system. 14 considers Data-Centric Dynamic Systems (DCDS), i.e. relational databases equipped with guarded actions that can modify their contents, and call external services. Most of these data-centric models are Turing powerful, and hence in their full generality can be seen as programs coordinating data transformations (or calls to a crowdsourcing platform). Their strength lies in their capacity to describe formally how data is transformed, an for some models in the existence of subsets of the language that can be formally verified [1, 5]. However, they are tailored to manipulate data within a single transaction, and do not consider replication of tasks nor aggregation of results, which are essential in crowdsourcing.

Some works propose empirical solutions for complex data acquisition, mainly at the level of micro-tasks [12, 19]. Crowdforge uses Map-Reduce techniques along with a graphical interface to solve complex tasks [17]. Turkit [20] is a crash and rerun programming model. It builds on an imperative language, that allows for repeated calls to services provided by a crowdsourcing platform. Turkmatic [18] is a tool that recruits crowd workers to help clients planning and solving complex jobs. It implements a Price, Divide and Solve (PDS) loop, that asks crowd workers to divide a task into orchestrations of subtasks, and repeats this operation up to the level of micro-tasks. A PDS scheme is also used by [33] in a model based on hierarchical state machines that orchestrates sub-tasks.

In this work, we assemble answers returned by workers using aggregation techniques. Basic aggregation uses majority voting (MV), i.e. takes as final result for a tagging task the most returned answer. Several approaches have improved MV by using Expectation Maximization (EM) or by considering workers competences, expressed in terms of accuracy (ratio of correct answers) or in terms of recall and specificity (that considers correct classification for each possible type of answer). It is usually admitted [34] that recall and specificity give a finer picture of worker’s competence than accuracy. We only highlight works that focus on EM or MV to aggregate data, and refer interested readers to [34] for a more complete survey of the domain. Zencrowd [8] considers workers competences in terms of accuracy and aggregates answers using EM. Workers accuracy and ground truth are the hidden variables that must be discovered in order to minimize the deviations between workers answers and aggregated conclusion. D&S [7] uses EM to synthesize answers that minimize error rates from a set of patient records. It considers recall and specificity, but not the difficulty of tasks. [15] proposes an algorithm to assign tasks to workers, synthesize answers, and reduce the cost of crowdsourcing. It assumes that all tasks have the same difficulty, and that workers reliability is a static probability to return the correct value that applies to all types of tasks. EM is used by [27] to discover recall and specificity of workers and propose maximum-likelihood estimator that jointly learns a classifier, discovers the best experts, and estimates ground truth. Most of the works cited above consider expertise of workers but do not address tasks difficulty. Approaches such as GLAD [32] or [4] also estimate tasks difficulty to improve quality of answers aggregation on a single dataset.

Generally the database and machine learning communities focus on data aggregation techniques and leave budget optimization apart. CrowdBudget [29] is an approach that divides a budget $B$ among $K$ existing tasks to replicate them and then aggregate answers with MV. Raykar et.al [26] use Markov Decision Processes to decide whether answers to a database tagging scheme are sufficiently accurate or new worker should be hired. Crowdinc [28] is an EM-based aggregation technique that considers task difficulty, recall and specificity of workers. It computes accuracy of an aggregation, and launches new tasks dynamically to achieve a given threshold. The model proposed in this paper is a workflow that orchestrates tasks, replicates them, distributes them and aggregates the returned results before passing the forged dataset to the next tasks. It is a variant of the complex workflow
model proposed in [3], and it uses the aggregation technique of Crowdinc [28] to forge reliable answers.

The rest of the paper is organized as follows. Section 2 describes the complex workflow model, and Section 3 the aggregation technique used to forge answers from crowdworkers returns. In Section 4, we propose algorithms to realize complex workflows while maintaining a trade-off between cost and quality. We report the experimental evaluation and result in Section 5 before conclusion. Due to lack of space, some technical details are provided in a separate appendix at the end of the paper.

2 COMPLEX WORKFLOW WITH AGGREGATION

In this section, we formalize the complex workflow model. The model is inspired by data centric workflows [3], but adds tasks replication to this model, and further considers aggregation and budget management in its semantics. The context of the workflow is the following: A client wants to realize a complex task that needs the knowledge and skills of human workers. Complex tasks are divided into several dependent phases. Each phase processes records from an input dataset or merges different inputs to a single one, and forwards the result to its successor. Datasets are collections of records, i.e. relations of the form \( (x_1, \ldots, x_k) \) where \( x_1, \ldots, x_k \) are values for the fields of the record. One can use First Order statements to address properties of a record (e.g. write \( v_i = \text{true} \)), or of a set of records in a dataset (e.g. \( \exists r(x_1, \ldots, x_k) \in D, v_i = \text{true} \)). We will denote by \( FO^R \) the FO formulas for records, and by \( FO^P \) the FO formulas for datasets. For simplicity, we assume that processing a record is a micro-task that simply consists in adding a new Boolean field (called a tag) to this record. Hence a micro-task can be seen as an operation that transforms a record \( r(x_1, \ldots, x_k) \) into a new record \( r'(x_1, \ldots, x_k, v_{k+1}) \) where \( v_{k+1} \) is a Boolean value. This setting can be easily adapted to let \( v_{k+1} \) take values from a discrete domain.

As humans are prone to errors, micro-tasks are replicated, and allocated to several workers. Their answers are then aggregated before proceeding to the next tasks. Hence, an aggregation mechanism is required to combine the answers and forward the results to the next phases. When a phase has several successors, the contents of records is used to decide to which successor(s) results to the next phases. When a phase has several successors, tagging of records, any record processed on a predecessor of \( p_x \) can be processed individually without waiting for other results to be available. This allows asynchronous executions in which two phases \( p_x, p_y \) can be concurrently active (i.e. have started processing records), even if \( p_x \) precedes \( p_y \). On the contrary, if the execution of a phase \( p_x \) is synchronous, and \( p_y \) is a successor of \( p_x \) all records output to \( p_y \) must be tagged before starting phase \( p_y \).

The semantics of a complex workflow is defined in terms of moves from a configuration to the next one, organized in rounds. Configurations memorize the data received by phases, a remaining budget, the answers of workers, and quality measures on answers and workers competences.

Definition 2.2. A configuration of a complex workflow is a tuple \( C = (D_\text{init}, W_\text{init}, W_\text{out}, \text{conf, exp, B}) \) where

- \( D_\text{init} : P \to \text{Dsets} \) Associates a dataset to every phase \( p_x \in P \) (this dataset can be empty).
- \( W_\text{init} : P \times [0,1] \to 2^W \) associates a set of workers to each record in \( D_\text{init}(p_x) \).
- \( W_\text{out} : P \times 2^W \times W \to \{0,1\} \cup \emptyset \) associates a tag or the empty set to a worker, a phase and a record number that was formerly affected to the worker. We will write \( W_\text{out}(p_x, w, r, k) = \emptyset \) to denote the answer returned by worker \( w \) when tagging record \( r \) in phase \( p_x \).
- \( \text{conf} : P \times 2^W \to [0,1] \) is a map that associates to each record in \( D_\text{init}(p_x) \) a confidence score in \( [0,1] \) computed from \( W_\text{out} \).
- \( \text{exp} : W \times P \to [0,1] \) is a confidence score that associates to each worker a score between 0 and 1.
- \( B \in [0,1] \) is the remaining budget.

\( W_\text{out}(p_x, k, w) = \emptyset \) indicates that a worker \( w \) in a phase \( p_x \) has not yet processed record \( r_k \). We say that phase \( p_x \) is completed for a record \( r_k \) if \( W_\text{out}(p_x, r, k) \neq \emptyset \). As soon as phase \( p_x \) is completed for \( r_k \), we can derive an aggregated answer \( \text{agg}(p_x, w, r_k) \) for each record \( r_k, (v_1, \ldots, v_n) \) from the set of all answers returned by the workers in \( W_\text{init}(p_x, k) \). Similarly, we can compute a confidence score
conf(\(p_x, k\)) on the value \(y^*_k\) and the expertise of each worker
(we will see how these values are evaluated in section 4). We say
that a record \(r_i\) in a phase \(p_x\) is inactive if no more workers are
assigned to it. It is active otherwise. Given a threshold value \(T_h\),
will say that \(p_x\) is finished for a record \(r_i\) from \(D_{in}(p_x)\) if \(p_x\) is
completed and \(conf(p_x, k) > T_h\). Record \(r'_k(w_1, \ldots, w_n, y^*_k)\) will
then be part of the input of phase \(p_y\) if \((p_x, g_{x,y}, p_y) \leftarrow \rightarrow\) and \(r'_k(v_1, \ldots, v_n, y^*_k)\) satisfies guard \(g_{x,y}\).
We can now detail how rounds change the configuration of a
workflow. The key idea is that each round aggregates available
answers, and then decides whether the confidence in aggregated
results is sufficient. If confidence in a record is high enough, this
record is forwarded to the successor phases, if not new work-
ers are hired for the next round, which decreases the remaining
budget. The threshold for the confidence decreases accordingly.
Then new workers are hired for freshly forwarded data, leav-
ing the system ready for the next round. From a configuration
\(C = (D_{in}, W_{in}, W_{out}, conf, exp, B)\), a round produces a new con-
figuration \(C' = (D_{in}, W_{in}, W_{out}, conf, exp, B)\) as follows:

- **Answers**: Workers that were hired in preceding round
produce new data. For every phase \(p_x\) every record \(r_i\) in
\(D_{in}(p_x)\) and every worker \(w_i\) such that \(w_i \in W_{in}(p_x, n)\)
and \(W_{out}(p_x, w_i, n) = \emptyset\), we produce a new output
\(l^*_i \in \{0, 1\}\) and set \(W_{out}(p_x, w_i, n) = l^*_i\).

- **Aggregation**: The system aggregates overall answers
in every active phase \(p_x\). For every record \(r_i\) in
\(D_{in}(p_x)\), we compute an aggregated answer \(y^*_k\) from the set of
answers \(A_n = \{l^*_i \mid w_i \in W_{in}(p_x, n)\}\). We also compute a new
confidence score \(conf'(p_x, n)\) for the aggregated
answer (this confidence depends on the aggregation techni-
que), and evaluate tasks difficulty and workers expertise
(with the algorithm shown in Section 3).

- **Data forwarding**: We distinguish asynchronous and syn-
chronous phases. Let \(p_y\) be an asynchronous phase (\(\bigotimes_y\))
that is a union of records. Then \(p_y\) accepts every new
record \(r'(v_1, \ldots, v_n, y^*_k)\) that was not yet among its inputs
from a predecessor \(p_x\) provided \(r'\) satisfies guard \(g_{x,y}\), and the
confidence in the aggregated answer \(y^*_k\) is high enough.
Formally, \(D_{in}(p_y) = D_{in}(p_x) \cup \{r'(v_1, \ldots, v_n, y^*_k)\}\) if
\((p_x, g_{x,y}, p_y) \leftarrow \rightarrow\) and \(conf'(p_x, n) \geq T_h\) and \(r'(v_1, \ldots, v_n, y^*_k)\) = \(g_{x,y}\).
Let \(p_y\) be a phase such that \(\bigotimes_y\) is synchronous. We
will say that a phase is closed if all its predecessors are
closed, and for every \(n, r_i \in D_{in}(p_x), conf(p_x, n) \geq T_h\).
If there exists a predecessor \(p_x\) of \(p_y\) that is not closed,
then \(D_{in}(p_y) = \emptyset\). Otherwise we can compute an input for
phase \(p_y\) as a join over datasets computed by all preceding
phases. Formally,
\[
D_{in}(p_y) = \bigotimes_{p_x \leftarrow \rightarrow p_y} D_x
\]
where \(p_x\) ranges over the set of predecessors of \(p_y\), and
\(D_x = \{ r(v_1, \ldots, v_n, y^*_k) \mid r(v_1, \ldots, v_n) \in D_{in}(p_x)
\land (v_1, \ldots, v_n, y^*_k) = g_{x,y}\}\).

Hence, for synchronous a phase \(p_y\), the input dataset is a
join operation computed over datasets filtered by guards,
and realized only once predecessor tasks have produced all
their results. Both in synchronous and asynchronous set-
tings, phase \(p_y\) becomes active if \(D_{in}'(p_y) = G(p_y)\), and we
set \(conf'(p_y, n) = 0\) for every new record in \(D_{in}'(p_y)\)

- **Worker allocation**: For every phase \(p_x\) that is active and every
record \(r_i\) = \((v_1, \ldots, v_n, y^*_k) \in D_{in}'(p_x)\) such that \(conf'(p_x, n) < T_h\),
we allocate \(k\) new workers \(w_1, \ldots, w_k\) to record \(r_i\) for
phase \(p_x\), i.e. \(W'_{in}(p_x, n) = W_{in}(p_x, n) \cup \{w_1, \ldots, w_k\}\).
The number of workers depend on the chosen policy (see de-
ctails in Section 4). Accordingly, for every new worker \(w_i\)
affected to a tagging task for a record \(r_n\) in phase \(p_x\), we set
\(W_{out}(p_x, w_i, n) = \emptyset\).

- **Budget update**: We then update the budget. The overall
number of workers hired is
\[
NW = \sum_{p_x \in P} \sum_{r_n \in D_{in}(p_x)} |W'_{in}(p_x, n) \setminus W_{in}(p_x, n)|
\]
We consider, for simplicity, that all workers and tasks have
identical costs, we hence set \(B' = B - NW\).

An execution begins from an initial configuration \(C_0\) in which
only \(p_0\) is active, with an input dataset affected to \(p_0\), and starts
with workers allocation. Executions end in a configuration \(C_f\)
where all records in \(D_{in}(p_f)\) are tagged with a sufficient thresh-
old. Notice that several factors influence the overall execution of
a workflow. First of all, the way workers answers are aggregated
influence the number of workers that must be hired to achieve
a decent confidence in the synthesized answer. We propose to
consider two main aggregation policies. The first one is majority
voting (MV), where a fixed static number of workers is hired for
each record in each phase. A second policy is the expectation
maximization (EM) based technique proposed in [28], in which
workers are hired on demand to increase confidence in the
aggregated answer. With this policy, the confidence level is computed
taking into account the estimated expertise of workers, and the
difficulty of record tagging. The number of workers hired per
record in a phase is not fixed, but rather computed considering
the difficulty of tagging records, and the remaining budget.

For a given workflow, asynchrony is another key factor that
may influence the time and budget spent to realize a complex task.
Recall that for a phase \(p_x\) which input dataset is built as an union
of sets of records, asynchronous guards allow to start processing
records as soon as \(D_{in}(p_x) \neq \emptyset\). Synchronous guards force \(p_x\)
to wait for the termination of its predecessors. In Section 5, we
study the impact of synchronous/asynchronous guards on the
overall execution of a workflow.

3 AGGREGATION MODEL

As mentioned in previous section, crowdsourcing requires replica-
tion of micro-tasks, and aggregation mechanisms for the answers
returned by the crowd. For simplicity, we consider Boolean tasks,
i.e., with answer 0 or 1. However, the model easily extends to a
more general setting with a discrete set of answers.

Consider a phase \(p_x\), which input is a set of records \(D_x = \{r_1, r_2, \ldots, r_n\}\), and which goal is to associate a Boolean tag
to each record of \(D_x\). We assume a set of \(k\) independent workers
that return Boolean answers, and denote by \(l_{ij}\) the answer returned
by worker \(j\) for a record \(r_i\), \(L_i = \bigcup_{j=1}^{k} l_{ij}\) denotes the set of answers
returned by \(k\) workers for a record \(r_i\) and \(L = \bigcup_{1 \leq j \leq n} L_j\) denotes
the set of all answers. We assume that workers are independent (there
is no collaboration and their answers are hence independent),
and faithful (they do not give wrong answers intentionally).
The objective of aggregation is to derive a set of final answers
\(Y = \{y_j, 1 \leq j \leq n\}\) from the set of answers \(L\). Once a final answer
\(y_j\) is computed, it can be appended as a new field to record \(r_j\).
The set of produced results can be used to launch new phases and to forward records to successor phases of $p_x$.

We consider several parameters to model tasks and workers, namely the difficulty to tag a record, and the expertise of workers. The difficulty to tag a record $r_j$ is modeled by a real valued parameter $d_j \in [0, 1]$. Value 0 means that tagging $r_j$ is very easy, and $d_j = 1$ means that it is extremely difficult. Expertise of a worker is often quantified in terms of accuracy, i.e. as the ratio of correct answers. However, accuracy can lead to bias in the case of datasets with unbalanced ground truth. Indeed, consider a case where the number of records with ground truth $Y$ is much higher than the number of records with ground truth 0. If a worker annotates most of records with ground truth 1 as 1 but makes errors when tagging records with ground truth 0, her accuracy will still be very high. We hence prefer a more precise model, where expertise of a worker is given as a pair $\xi_j = (a_j, \beta_j)$, where $a_j$ is the recall and $\beta_j$ the specificity of worker $i$. The recall $a_j$ is the probability that worker $i$ answers 1 when the ground truth is 1, i.e. $a_j = \Pr(l_{ij} = 1|y_j = 1)$. The specificity $\beta_j$ is the probability that worker $i$ answers 0 when the ground truth is 0, i.e. $\beta_j = \Pr(l_{ij} = 0|y_j = 0)$. We do not have a priori knowledge of the behavior of workers, so we define a generative model to determine the probability of correct answers when $a_j, \beta_j$ are known. This probability depends on the difficulty of a task, on recall and specificity of the considered worker, and on the ground truth. We set $\Pr(l_{ij} = y_j|d_j, a_j, y_j = 1) = (1 + (1 - d_j)^{(1 - a_j)})/2$ and $\Pr(l_{ij} = y_j|d_j, \beta_j, y_j = 0) = (1 + (1 - d_j)^{(1 - \beta_j)})/2$.

Figure 2 shows probability to get $l_{ij} = 1$ when $y_j = 1$. The horizontal axis represents the difficulty of a task, the vertical axis the probability to get the final answer $y_j = 1$. Each curve represents this probability for a particular value of recall. Note that the vertical axis ranges from 0.5 to 1.0 as a random guess by a worker can still provide a correct answer with probability 0.5. As the difficulty of task increases, the probability of giving a correct answer decreases and when the task difficulty is 1, all workers cannot do better than a random guess. For a fixed difficulty of a task, the higher recall is, the more accurate answers are.

We equip complex workflows with an aggregation technique that uses Expectation Maximization (EM) [13] to estimate jointly latent variables $(a_i)_{i\in 1..k}, (\beta_i)_{i\in 1..k}, (d_j)_{j\in 1..n}$ and derive the final answer $y_j$ for each record $r_j$. We denote by $\theta$ the values of $(a_i)_{i\in 1..k}, (\beta_i)_{i\in 1..k}, (d_j)_{j\in 1..n}$. EM iterates two alternating steps. In the E-step, we compute the posterior probability of $y_j \in \{0, 1\}$ for a given record $r_j$ given the difficulty $d_j$, workers expertise $(a_i, \beta_i)_{i\in 1..k}$ and the answers $L_j = \{l_{ij} | i \in 1..k\}$. In the M-Step, we compute the parameters $\theta$ that maximize $Q(\theta, \theta^\prime)$ with respect to the estimated posterior probabilities of $Y$ computed during the E-step of the algorithm. Let $\theta^\prime$ be the value of parameters computed at step $t$ of the algorithm. We use the observed values of $L$, and the previous expectation for $Y$.

$$Q(\theta, \theta^\prime) = \mathbb{E}[\log \Pr(L, Y | \theta) | L, \theta^\prime]$$

We can hence compute the next value as: $\theta^{t+1} = \arg\max_{\theta} Q(\theta, \theta^t)$. We maximize $Q(\theta, \theta^t)$ using optimization techniques provided by the scipy\(^2\) library. We iterate E and M steps, computing at each iteration $t$ the posterior probability and the parameters $\theta^t$ that maximize $Q(\theta, \theta^t)$. The algorithm converges, and stops when the difference between two successive joint log-likelihood values is below a threshold (fixed in our case to $10^{-5}$). It returns values for parameters $(a_i)_{i\in 1..k}, (\beta_i)_{i\in 1..k}, (d_j)_{j\in 1..n}$. The final answer is the most probable $y_j$.

## 4 COST MODEL FOR WORKFLOW

The objective of a complex workflow $W$ over a set of phases $P = \{p_0, \ldots, p_k\}$ is to transform a dataset input to the initial phase $p_0$ and eventually produce an output dataset. The final answer is the result of the last processed phase $p_f$. The simplest scenario is a workflow that adds several binary tags to input records. The realization of a micro-task by a worker is paid, and workflows come with a fixed maximal budget $B_0$ provided by the client. For simplicity, we consider that each worker receives one unit of credit per realized task. As explained in section 2, each phase receives records, each record is tagged by one or several workers. Answers are then aggregated, and the records produced by a phase $p_k$ are distributed to its successors if they meet some conditions on the data. A consequence of this filtering done by conditions is that records have different lifetimes and follow different paths in the workflow. Further, one can hire more workers to increase confidence in an aggregated result if needed and if a sufficient budget remains available. Several factors influence the realization of a workflow and its cost: the number of tagging tasks that have to be realized, the available initial budget, the confidence in produced results, workers expertise, the size and nature of input data, the difficulty of tagging, and the policies chosen to realize a workflow and to hire workers.

Existing crowdsourcing platforms usually follow static allocation schemes, i.e. fix a number $K_r$ of workers to hire for each micro-task. An obvious drawback of this approach is that the same effort is spent to solve easy and difficult tasks.

In section 2, we have defined synchronous and asynchronous schemes to allocate workers on-the-fly to tasks. In this section, we define the cost model associated with these schemes, and in particular, the threshold measure used to decide whether more workers should be hired. We show in section 5 that the algorithm achieves a good trade-off between cost and accuracy. Recall that at each round, we allocate new micro-tagging tasks to workers, to obtain answers for records that are still open. EM aggregation is used to obtain a plausible aggregated tag $y_j^\ast$ for each record $r_j$ from a set of answers $L_j^\ast$ obtained in each active phase $p_x$. The algorithm gives an estimation of difficulty $d_j^\ast$ of tagging record $r_j \in p_x$, and evaluates the expertise level of every worker $w_i$, i.e. its recall $a_i$ and its specificity $\beta_i$. We also obtain a confidence score $\xi_j^\ast$ for the aggregated answer. This score is used to decide whether we need more answers or conversely consider $y_j^\ast$ as a definitive result. Let $k_j^\ast = |L_j^\ast|$ denote the number of answers for record $r_j \in p_x$. The confidence $\xi_j^\ast$ in final label $y_j^\ast$ is defined as:

$$\xi_j^\ast(y_j^\ast) = \frac{1}{k_j^\ast} \sum_{i=1}^{k_j^\ast} \frac{1}{|L_i^\ast|} \left( \left( (1 - d_i^\ast) \cdot (1 - a_i) + (1 - a_i) \cdot (1 - d_i^\ast) \right) \cdot \frac{1}{2} \right)$$

---

\(2\) docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

---

\(1\) Chap. 9 and \(2\) -Chap. 9
we first compute a bound on the number of remaining phases that we may have to fund up to $\tau.n.h$. Each worker adds its probability to readability.

Intuitively, in a realization of a workflow is a non-exclusive fork phase, then there is no other fork along a branch before the merge phase. A phase $p_y$ that is a merge phase on maximal number of active copies or a record at a given instant. The notions of height and width are interesting to evaluate the realization of a workflow is the initial budget $B_0$. The algorithm to compute the foreseeable workload in a phase $p_x$ is a node $p_y$ such that all path originating from $p_x$ visit $p_y$, and no predecessor of $p_y$ satisfy this property. To decide whether to $p_y$ is the corresponding node of $p_x$, one can compute the set of nodes that are both successors of $p_x$ and predecessor of $p_y$, and decide for each of them whether they have a successor that is not a predecessor of $p_y$. This can be done in polynomial time. Then, the maximal number of phases that have to be realized between $p_x$ and its corresponding node $p_y$ is the length of the maximal path from $p_x$ to $p_y$ (which can also be computed in PTIME).

The algorithm to compute the foreseeable workload in a phase $p_x$ consists in computing the most expensive path in a workflow once the cost of parallel processes contained between a non-exclusive fork node and its corresponding phase is evaluated and this set of nodes is replaced by a single phase of corresponding cost. This is described more precisely in Algorithm 1.

Algorithm 1: $FW(p_n)$

Data: Workflow $W$, phase $p_n$
Result: $fw(p_n)$

1. for each node $p_x$ in $Succ(p_n)$ do
2. $\text{Cost}(p_x) := \emptyset$
3. end

4. for each non-exclusive fork node $p_x$ in $Succ(p_n)$ do
5. Find $p_y$ the corresponding node of $p_x$;
6. $Z = Succ(p_x) \cap pre(p_y)$;
7. Replace $Z$ by a fresh node $p'_x$;
8. $\text{Cost}(p'_x) = |Z|$
9. end

10. Compute the most expensive path $p_n.p_{i1} \ldots p_{ik}.p_f$ from $p_n$ to $p_f$;
11. $fw := 1 + \sum \text{Cost}(p_{ik})$
12. return $fw$;

Definition 4.2 (Foreseeable workload). Let $n_e$ denote the total number of active records at a phase $p_x$ in a configuration $C$. The foreseeable task number from $p_x$ in configuration $C$ is denoted $f_{TC}(p_x)$ an defined as $f_{TC}(p_x) = n_e \times FW(p_x)$. The foreseeable task number in configuration $C$ is the sum $FTN(C) = \sum_{p_x \in P} f_{TC}(p_x)$. Let us now define a threshold function based on the current configuration of the workflow. This function must considers all records that still need processing, the remaining budget, and an upper bound on the number of tagging tasks that will have to be realized to complete the workflow. Further, the execution policy will influence the way workers are hired, and hence the budget spent. In a synchronous execution, records in a phase $p_x$ can be processed only when all records in preceding phases have been processed. On the contrary, in asynchronous execution mode, processing of records input to a phase $p_x$ can starts without waiting for closure of all records input to preceding phases. A consequence is that in synchronous modes, decisions can be taken locally to each phase, while in an asynchronous mode, the way to tune threshold must be taken according to a global view of the remaining work in the workflow. Hence, for an asynchronous execution policy, we will consider a global threshold function, computed for the whole workflow. On the contrary, for an asynchronous execution policy, we will define a more local threshold function computed phase by phase.
Asynchronous execution. We now define a global ratio $R_t \in [0,1]$ of executed work in a workflow. The execution of a workflow starts from a configuration $C_0$ with an expected workload of $FTN(C_\ell)$. It is an upper bound, as all records do not necessarily visit the maximal number of phases. We define a ratio depicting the proportion of already executed or avoided work as $R_t = \frac{(FTN(C_0) - FTN(C_\ell))}{FTN(C_\ell)}$. Note that at the beginning of an execution, $R_t = 0$ as no record is processed yet. When records are processed and moved to successor phases, $R_t$ increases, and we necessarily have $R_t = 1.0$ when no record remains to process. Now, the threshold value has to account for the remaining budget to force the progress of records processing. Let $B_{\ell\ell}$ denote the budget at the beginning of execution, and $B_\ell$ be the budget consumed. We denote by $\mathcal{B}$ the fraction of budget consumed at a given execution time, i.e. $\mathcal{B} = \frac{B_{\ell\ell}}{B_\ell}$. Note that, at the beginning of an execution, $\mathcal{B} = 0$. $\mathcal{B}$ increases at every round of the execution, and takes value $\mathcal{B} = 1$ when the whole budget is spent. We now define a global threshold value $Th \in [0.5,1.0]$ that accounts for the remaining work and budget.

$$Th = \frac{1 + (1 - \mathcal{B})R_t}{2} \quad (1)$$

We remind that in a phase $p_x$, a record with confidence level $c > Th$ is considered as processed for phase $p_x$. In an asynchronous execution policy, the threshold is a global value and applies to all records in the workflow at a given instant. The intuition for $Th$ is simple: when only a few record remain to be processed, and the remaining budget is sufficiently high, then one can afford final answers with a high confidence threshold. This means that many records will obtain new answers and probably increase their current confidence level. Conversely, if the number of records to be processed is high and the remaining budget is low, then the threshold decreases, and even records which current answer have a low confidence level are considered as processed and moved to the next phase(s).

Synchronous execution. In asynchronous execution, records are processed individually, and a phase does not wait for completion of its predecessors to start. As a consequence, all phases of the workflow can be active at the same time. As in each round, records can be processed in all phases, we consider a global threshold, and hence a global budget allocation policy. However, in synchronous executions, records are processed phase wise, i.e. a phase does not start processing its input dataset until all records in the preceding phases are processed. Here, the global threshold used in asynchronous executions may not work for synchronous executions. As task are realized phase wise, in the early phases of workflows execution, the threshold will be high. One may face situations where at the end of an execution, the number of records to process is still high, but a significant part of the budget has already been used. Within this setting, at the end of the execution, the threshold is low. It forces to accept final answers with low confidence. This may affect the overall quality of the final output of the workflow. To avoid this problem, we propose to allocate the budget phase by phase. The idea is to divide the budget among phases based on the number of records processed.

In synchronous executions, phases start processing all their records once preceding phases have completed their work (i.e. all records are processed and for each of them, a final answer with sufficient confidence score have been aggregated). We will say that a task becomes active when it starts processing records, and denote by $Init(p_x)$ the number of records input to $p_x$ when the phase becomes active.

As for asynchronous execution, synchronous execution starts with an initial budget $B_0$, and in each configuration $C$, one knows the remaining budget $B(C)$. The key idea in synchronous execution is to compute resources needed for each active phase, a ratio of input records that still need additional answers to forge a trusted answer, and a local threshold.

The initial budget allocated to a phase $p_x$ with $n_x$ records in a configuration $C$ is

$$B_0^x = \frac{B(C)}{\sum_{p_x \text{ active phase}} FTN(p_x)} \times n_x$$

Intuitively, one shares the remaining budget among active phases to allow termination of the workflow from each phase. Then for each phase, we maintain the consumed budget $B_x^T$ for phase $x$, and the ratio $\mathcal{B}_x = \frac{B_x^T}{B_x}$ of consumed budget in phase $p_x$.

Now, for each active phase $p_x$, we compute the ratio

$$R_x^T = \frac{|\{r_j \mid c_j \leq T_h(x)\}|}{Init(p_x)} \quad (2)$$

where $T_h(x)$ is the threshold computed for the previous round. A local threshold for the realization of the next round of an active phase can then be computed as in asynchronous execution, using the formula

$$T_h(x) = \frac{1 + (1 - \mathcal{B}_x^T)R_x}{2} \quad (3)$$

With the convention that the initial threshold $T_h(x)$ for a starting active phase, as no record is processed yet is $T_h(x) = \frac{1 + \mathcal{B}_x^T}{2}$.

Realization of Workflows. Regardless of the chosen policy, the execution of a workflow always follows the same principles. The structure of workflow $W$ is static and does not change with time. It describes a set of phases $P = \{p_0, \ldots, p_f\}$, their dependencies, and guarded data flows from one phase to the next one. A set of $n$ records $R = \{r_1, \ldots, r_n\}$ is used as input to $W$, i.e., is passed to initial phase $p_0$, and must be processed with a budget smaller than a given initial budget $B_0$. As no information about the difficulty of a task $d_j^x$ is available at the beginning of phase $p_0$, $\tau$ workers are allocated to each record for an initial estimation round. The same principle is followed for each record when it enters a new phase $p_x \in W$. After collection of $\tau$ answers, at each round we first apply EM aggregation to estimate the difficulty $d_j^x$ of active records $r_j \in p_x$, $c_j^x$ the confidence in the final aggregated answer $y_j^x$ and the recall $a_j$ and specificity $b_j$ of each worker $w_j$. Then we use a stopping threshold to decide whether we need more answers for each of the records in a phase. In asynchronous execution, the threshold $Th$ is a global threshold, and in synchronous mode, the confidence of each record $r_j$ in a phase $p_x$ is compared to the local threshold $T_h(x)$. Records with sufficient confidence are passed to the next phase(s), for other records we hire new workers to obtain more answers. This can increase the confidence level, but also decrease the threshold, as a part of the remaining budget is consumed. The execution stops when the whole budget $B_0$ is exhausted or when there is no additional record left to process. At the end, the final phase $p_f$ returns the aggregated answer for each of the record.

Termination. One can easily see that as the remaining available budget decreases, the threshold used to decide whether the aggregated answer for a record is final decreases too. However, there are situations where the confidence in each answer remains
low, and the remaining budget reaches 0 before the threshold
attains the lower bound 0.5 (that forces moving any record to
the next phase(s)). Similarly, when records do not progress in the
workflow, the ratio of remaining work \( R_t \) remains unchanged for
many rounds. As a consequence, the realization of a workflow
with our synchronous and asynchronous realization policies may
not terminate. We will see in the experimental results section that
even with poor accuracy of workers, this situation was never met.
Non-termination corresponds to situations where the weighted
answers of workers remain balanced for a long time. The thresh-
old decreases slowly, and the confidence on aggregated answers
remains lower. In that case, when threshold and confidence val-
ues coincide (in the worst case at value 0.5), the remaining budget
is too low to realize the remaining work. Solutions to solve this
issue and guarantee termination is to bound the sojourn time of
a record in a phase, or to keep a sufficient budget to terminate
the workflow with a static worker allocation policy hiring only a
small number of workers per record.

5 EXPERIMENTS AND RESULTS

In this section, we evaluate execution policies on typical work-
flows. We consider a standard situation, where a client wants to
realize a complex task defined by a workflow on a crowdsourcing
platform. The client provides input data, and has a budget \( B_0 \). We
assume that crowd workers do not collaborate and hence realize
their micro-tasks independently. As there exists no platform to
realize complex task, there is no available data to compare the
realization of a workflow with our approach to existing complex
task executions. To address this issue, we design several typical
workflows, synthetic data, and consider realizations of these
workflows for various the execution policy, characteristics of
data, and accuracy of workers.

We consider 5 different workflows, represented in Figure 4. Workflow \( W_1 \) is a sequence of tasks, \( W_2 \) is a standard fork-join
pattern i.e. parallel processing of data followed by a merge of
branches results, \( W_3 \) and \( W_4 \) are fork-join patterns with equal and
different lengths on branches, and \( W_5 \) is a more complex workflow
with two consecutive forks followed by merges on each branch.
We consider that each micro task simply tags records, and simple
exclusive guards sending each record to one successor, depending
on the tag obtained at this phase. In Figure 4 we depict these
choices by pairs of letters \((k_0, k_1)\) representing the binary decision
taken on each phase, and assume guards of the form \( f \Rightarrow l_0 \) or
\( f \Rightarrow l_1 \), where \( f \) is the field of records produced by the phase.
For example, in workflow \( W_2 \) phase \( p_4 \) considers two possible
tags denoted \( A \) and \( B \). After realization of the tasks, if the records
are tagged as \( A \) by the workers then records are moved to the
phase \( p_7 \) and if tagged with \( B \) the records are assigned to phase
\( p_1 \) for further processing. Each phase of workflows implements
similar tagging and decision.

We evaluate average costs and accuracies achieved by work-
flows realizations with the following parameters. First, the input of
each complex task is a dataset of 80 records. Notice that despite
this fixed size, the number of micro-tasks to realize by workers
vary depending on the execution policy, on the value of data
fields produced by workers, but also on the initial dataset, on
the initial budget, etc. Each record in the original dataset has ini-
tially known data fields, and new fields are added by aggregation
of workers answers during the execution of the workflow. For
these fields, we assume a prior ground truth, which influences
the probability that a worker answers 0 or 1 when filling this

<table>
<thead>
<tr>
<th>Workflow</th>
<th>( W_1 )</th>
<th>( W_2 )</th>
<th>( W_3 )</th>
<th>( W_4 )</th>
<th>( W_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_{\text{max}} ), ( k )</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>( B_0 )</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>400</td>
<td>500</td>
</tr>
<tr>
<td>Data Type</td>
<td>Balanced</td>
<td>Unbalanced</td>
<td>Balanced</td>
<td>Unbalanced</td>
<td>Balanced</td>
</tr>
<tr>
<td>Mechanism</td>
<td>Static MV</td>
<td>Asynchronous</td>
<td>Static MV</td>
<td>Asynchronous</td>
<td>Static MV</td>
</tr>
</tbody>
</table>

Table 1: Evaluation Parameters

field. We generate balanced (equal numbers of 0 and 1 in fields) and
unbalanced datasets (unbalanced numbers of 0 and 1).

We run the experiment with 4 randomly generated pools of 50
crowd workers, making their accuracy range from low to high
expertise. For each pool, we sampled accuracies of workers ac-
cording to normal distributions ranging respectively in intervals
\([0.2, 0.7]\) (low expertise of workers), \([0.4, 0.9]\) (low to average
expertise), \([0.6, 0.99]\) (average expertise) and \([0.8, 0.99]\) (high
expertise). The composition of pools is shown in Figure 3.

![Figure 3: Distributions of workers accuracy.](image)

The last parameter to set is the initial budget \( B_0 \). We first
evaluated the cost for the realization of workflows with a static
allocation policy that associates a fixed number of \( k \) workers
to each record in each phase, and aggregate their answers with
Majority Voting. We call this policy Static Majority Voting (SMV).
For each workflow, we performed random runs of SMV to eval-
uate the maximal budget \( B_{\text{max}} \) needed for three different values
\( k = 10/20/30 \). Note that the total budget \( B_{\text{max}} \) consumed by
static MV technique cannot be fixed a priori, as it depends on
the execution path followed by records during execution, with
random answers of workers. However, SMV is a naive approach,
that was shown inefficient in most benchmarks \([28]\), so starting
with a budget \( B_0 = B_{\text{max}} \) for realization techniques tailored to
save budget when accuracy is sufficient is a sensible approach.

In a second step, we used the total budget \( B_{\text{max}} \) spent by the
SMV approach as initial budget for synchronous and asynchro-
nous execution. The idea is to achieve at least the same accuracy
as SMV with synchronous and asynchronous execution with the
same initial budget \( B_0 = B_{\text{max}} \), while spending a smaller fraction
of this budget. Overall, our experiments cover realization of 5
different workflows with different values for initial budget, work-
ers accuracy, characteristics of data, and realization policy. This
represents 72 different contexts, represented Table 1 (one type of
experiment represents a selection of one entry in each row). We
run each experiment 15 times to get rid of bias. This represents a
sample of 1080 workflow realizations.

We can now analyze the outcomes of our experiments. A
first interesting result is that all workflow executions terminated
without exhausting their given initial budget, even with low
competences of users. A second interesting result is that for all
realization policies, and for all workflows, complex workflows
realization ends with poor accuracy when expertise is low. Con-
sider for instance the results of Figure 5. This Figure gives the
consumed budget and achieved accuracy for a given workflow and a given initial budget when workers have a low expertise. The first series of results concern Workflow 1 with a budget allowing respectively 10, 20, 30 workers per record in each phase. The overall expended budget with an SMV approach is around 1200, 2800, 4000, respectively. Regardless of the initial budget, synchronous and asynchronous approaches spend only a fraction of the budget allowed by SMV. Accuracy is not conclusive, as the best realization policy varies with each experiment: for instance, for W₁ with 10 workers per record to tag in each phase, SMV seems to be the best approach, while with a budget of 20, the synchronous approach is the best. However, most of the experiments achieve accuracies below 0.2, which is quite low. An explanation is that, as shown in Fig. 2, with low expertise, workers answers are almost random choices. Hence when all workers have a low expertise, individual errors are not corrected by other answers, and the ground truth does not influence the results. At each phase, the algorithms take their decisions mostly based on wrong answers provided by the low expert workers and in consequence the errors accumulate. The system’s behavior is then completely random, which results in poor performance. This tendency shown for all workflows and initial budgets with balanced data is confirmed on unbalanced data (the results of the experiments are shown in Appendix B, Figure 10).

Next, we consider experiments with mid-level to high expertise, which is the most common case in a crowdsourcing setting. The experiments with competent workers and synchronous and asynchronous execution policies clearly show that dynamic allocation schemes outperform the staticMV approach both in terms of cost and accuracy. One can easily see these results Figures 6, 7, 8, and 9, that represent executions of workflows W₁ and W₂ with three levels of expertise, 3 initial budgets, and all execution policies, both for balanced and unbalanced data. We show similar results in Appendix B for workflows W₃, W₄, W₅.

In the worst cases, synchronous and asynchronous executions achieve accuracies that are almost identical to that of SMV, but often give answers with better accuracy. With a sufficient initial budget, dynamic approaches achieve an accuracy greater than 0.9. An explanation for this improvement of synchronous and asynchronous executions w.r.t. SMV is that in SMV, one does not consider the expertise of the worker, whereas the synchronous and asynchronous executions are EM based algorithms that derive the final answers by weighting individual answers according to worker’s expertise. This makes EM-based evaluation of final answers more accurate than static MV. This improvement
workflow, whereas synchronous and asynchronous execution with respect to static MVs. Synchronous and asynchronous executions need a greater budget. During an asynchronous execution, this precise estimation helps of records per phase in synchronous execution, while it might evaluated by the EM aggregation improves with a larger number of variables such as the difficulty of tasks and workers expertise in the whole workflow. A consequence is that evaluation of hid-
phases, while asynchronous execution starts tasks independently
intuitive reason behind this result is that the way records are
processed the asynchronous execution in terms of accuracy. The
experiment is that in most cases synchronous execution out-
improves static allocation in terms of cost and accuracy.

As a short term extension for this work is to consider termina-
tion of complex tasks realization with dynamic policies. Indeed,
workflows realized with dynamic policies may not terminate: this happens when, for some record, all workers agree to re-
turn the answers that do not increase the confidence. However,
this situation was never met during our experiments, even with
low expertize of workers. The probability of non-terminating
executions with synchronous/asynchronous policies seems neg-
ligible. In our future work, we plan to demonstrate formally that
\( P(B_{\text{final}} = 0 \land \text{FTN}(C) > 0) \), the probability of reaching a configura-
tion with exhausted budget and remaining work to do is very low.

This work opens the way to new challenges. The next step is to test our approach with existing crowdsourcing platforms
on a real case study. We are targeting citizen science initiatives,
that typically require orchestration of various competence to
reach a final objective. Now that our model is settled, another
objective is to consider various strategies to hire workers in the
most efficient way. A possibility to address this challenge is to
see complex workflows as stochastic games, in which one player
tries to maximize accuracy and reduce costs, while its opponent
tries to achieve the opposite objectives.

### REFERENCES

   systems. Trans. on Database Systems 34, 4 (2009), 23:1–23:44.
   dependencies and arithmetic. Trans. on Database Systems 37, 3 (2012), 22.

![Figure 8: Workflow 2 on Balanced Data](image)

<table>
<thead>
<tr>
<th>Budget</th>
<th>Cost</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>20</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>30</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

![Figure 9: Workflow 2 on Unbalanced Data](image)

<table>
<thead>
<tr>
<th>Budget</th>
<th>Cost</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>20</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>30</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
to derive the most probable values for $d_j, \alpha_j, \beta_j$. We use a standard EM approach to infer the most probable actual answer $Y = \{y_1, \ldots, y_n\}$ along with the latent variables $\Theta = \{d_j, \alpha_j, \beta_j\}$. $E$ and $M$ phases of the algorithm is illustrated below.

**E Step:** We assume that all the answers $i$ are independently given by the workers. There is no collaboration between them. So, in every $L_j = \{\ell_j, \ldots, \ell_k\}$, $\ell_j$’s are independently sampled variables. We compute the posterior probability of $y_j \in \{0,1\}$ for a given task $j$ given the difficulty of task $d_j$, worker expertise $\alpha_j, \beta_j$, $i \leq k$ and the worker answers $L_j = \{\ell_i \mid i \in 1..k\}$. Using Bayes’ theorem, for a particular value $\lambda \in \{0,1\}$ we have:

$$
Pr(y_j = \lambda \mid L_j, d_j, \alpha_j, \beta_j) = \frac{Pr(\lambda \mid d_j, \alpha_j, \beta_j) Pr(y_j \mid \lambda, d_j, \alpha_j, \beta_j)}{Pr(y_j \mid d_j, \alpha_j, \beta_j)}
$$

(4)

Observe that, that $\alpha_j$ and $\beta_j$ are independent variables. We assume that both values of $y_j$ are equiprobable, i.e. $Pr(y_j = 0) = Pr(y_j = 1) = \frac{1}{2}$. Hence we get:

$$
Pr(y_j = \lambda \mid L_j, d_j, \alpha_j, \beta_j) = \frac{Pr(\lambda \mid d_j, \alpha_j, \beta_j) Pr(y_j \mid \lambda, d_j, \alpha_j, \beta_j)}{2}
$$

(5)

Similarly, the probability to obtain a particular set of labels $L_j$ is given by:

$$
Pr(L_j \mid \lambda, d_j, \alpha_j, \beta_j) = \prod_{i=1}^{k} Pr(y_j = \lambda \mid L_j, d_j, \alpha_j, \beta_j)
$$

(7)

Overall we obtain:

$$
Pr(L_j \mid y_j, \lambda, d_j, \alpha_j, \beta_j) = \prod_{i=1}^{k} Pr(y_j = \lambda \mid L_j, d_j, \alpha_j, \beta_j)
$$

(8)

Let us consider one of these terms, and let us assume that every $\ell_i$ in $L_j$ takes a value $\lambda_i$. We have

$$
Pr(L_j \mid y_j = \lambda, \alpha_j, \beta_j, d_j) = \prod_{i=1}^{k} Pr(y_j = \lambda \mid \ell_i , \alpha_j, \beta_j, d_j)
$$

If $\lambda = 0$ then $Pr(L_j = \lambda) = \alpha_j \beta_j (d_j + y_j - 1) = 1$, i.e. the probability to classify correctly a 0 as 0, and is the value $\delta_j = \frac{1}{\alpha_j \beta_j (d_j + y_j - 1)}$. Similarly, if $\lambda = 1$ then $Pr(L_j = \lambda) = \alpha_j \beta_j (d_j + y_j + 1) = \frac{1}{1+\alpha_j \beta_j (d_j + y_j - 1)}$ is the recall, i.e. the probability to classify correctly a 1 as 1, and is the value $\gamma_j = \frac{1}{\alpha_j \beta_j (d_j + y_j)}$. Then the probability to classify $y_j = 1$ as $\lambda$ is $\lambda_j = (1-\delta_j)$ and the probability to classify $y_j = 0$ as $\lambda_j = 0$ is $(1-\delta_j)$. Hence we have $Pr(L_j = \lambda) = \alpha_j \beta_j (d_j + y_j - 1) = \alpha_j \beta_j (d_j + y_j + 1) = \delta_j$. Similarly, we can write $Pr(L_j = \lambda) = \alpha_j \beta_j (d_j + y_j + 1) = \gamma_j$. Then equation 7 rewriting as:

$$
P_r(L_j \mid \ell_j, y_j, \lambda, d_j, \alpha_j, \beta_j) = \prod_{i=1}^{k} \frac{Pr(\lambda \mid \ell_j, \alpha_j, \beta_j, d_j) Pr(y_j \mid \lambda, \ell_j, \alpha_j, \beta_j, d_j)}{2}
$$

(9)

In the E step, as every $\alpha_j, \beta_j, d_j$ is fixed, one can compute $Pr(y_j \mid L_j, \alpha_j, \beta_j, d_j)$ and also choose as final value for $y_j$ the value $\lambda \in \{0,1\}$ such that $Pr(y_j = \lambda \mid L_j, \alpha_j, \beta_j, d_j) > Pr(y_j = (1-\lambda) \mid L_j, \alpha_j, \beta_j, d_j)$. We can also estimate the likelihood for the values of variables $P(L \mid y, \theta)$ for parameters $\theta = \{\alpha, \beta, d\}$, as

$$
Pr(y_j = \lambda \mid L, \theta) = Pr(y_j = \lambda, L) Pr(L) = \frac{Pr(L \mid y_j = \lambda, \theta)}{Pr(L \mid y_j = \lambda, \theta) + Pr(L \mid y_j = \lambda, \theta)}
$$

(10)

**M Step:** We compute the parameters $\theta$ that maximize $Q(\theta, \theta')$ with respect to the estimated posterior probabilities of $Y$ computed during the E phase of the algorithm. Let $\theta'$ be the value of parameters computed at step $t$ of the algorithm. We use the observed values of $L$, and the previous expectation for $Y$. We maximize $Q(\theta, \theta') = E \left[ \log(Pr(L, Y \mid \theta') \mid L, \theta' \right]$ (we refer interested readers to [11]. Chap. 9 and [9] for explanations showing why this is equivalent to maximizing $Q(\theta, \theta')$). We can hence
compute the next value as: \( \theta_{t+1} = \arg \max \ Q'(\theta, \theta') \). Here in our context the values of \( \theta \) are \( \alpha_i, \beta_i, d_j \). We maximize \( Q'(\theta, \theta') \) using bounded optimization techniques provided by the standard SCIPy implementation. We iterate E and M steps, computing at each iteration \( t \) the posterior probability and the parameters \( \theta^t \) that maximize \( Q'(\theta, \theta') \). The algorithm converges, and stops when the improvement (difference between two successive joint log-likelihood values) is below a threshold (in our case \( 1 \times 10^{-7} \)).

**B ADDITIONAL RESULTS**

For completeness, we give the experimental results for low accuracy (Fig. 10), and for the 72 contexts considered (all workflows, 3 initial budgets, all levels of expertise, and all types of data).