Cost and Quality in Crowdsourcing Workflows

Loïc Hélouët, Zoltan Miklos, Rituraj Singh

Cost and Quality in Crowdsourcing Workflows*

Loïc Hélouët[1], Zoltan Miklos[2], and Rituraj Singh[2]

[1] INRIA Rennes loic.helouet@inria.fr
[2] Univ. Rennes 1 {zoltan.miklos,rituraj.singh}@irisa.fr

Abstract. Crowdsourcing platforms provide tools to replicate and distribute micro tasks (simple, independent work units) to crowds and assemble results. However, real-life problems are often complex: they require to collect, organize or transform data, with quality and costs constraints. This work considers dynamic realization policies for complex crowdsourcing tasks. Workflows provide ways to organize a complex task in phases and guide its realization. The challenge is then to deploy a workflow on a crowd, i.e., allocate workers to phases so that the overall workflow terminates, with good accuracy of results and 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 define new *dynamic* worker allocation techniques that consider progress in a workflow, quality of synthesized data, and remaining budget. Evaluation on a benchmark shows that dynamic approaches outperform static ones 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 [23]. A platform allows employers to post tasks, that are then realized by workers in exchange for some incentives [3]. Common tasks include image annotation, surveys, classification, recommendation, sentiment analysis, etc. [10]. The existing platforms support simple, repetitive and independent *micro-tasks* which require a few minutes to an hour to complete.

However, many real-world problems are not simple micro-tasks, but rather complex orchestrations of dependent tasks, that process input data and collect workers answers for tasks requiring human expertize. 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 specify more involved tasks over existing crowd platforms.

* Work supported by the Headwork ANR
[1] www.mturk.com, [2] www.appen.com, [3] www.wirk.com

A natural solution is to define complex tasks as workflows, i.e., orchestrations of phases that exchange data to achieve a final objective [27]. The data output by an individual phase is passed to the next one(s) according to the workflow rules.

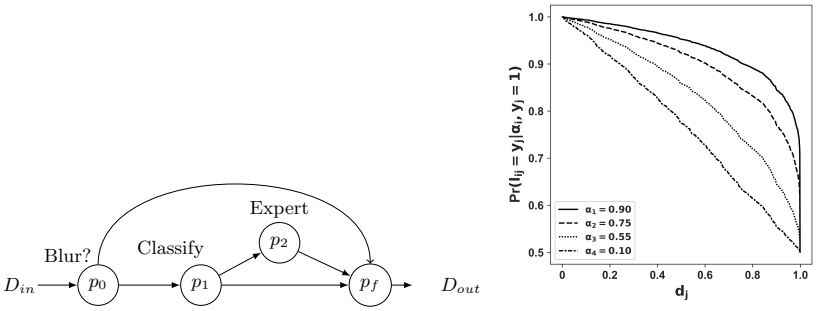

Fig. 1. *a*) A workflow from SPIPOLL, *b*) Generating functions $Pr(l_{ij} = y_j | d_j, \alpha_i, y_j = 1)$

We illustrate complex workflows in Figure 1-*a*). This workflow is an image annotation process on SPIPOLL [5], a platform to survey populations of pollinating insects. Contributors take pictures of insects that are then classified by crowdworkers. Pictures are grouped in a dataset D_{in}, input to node p_0 . The process is the following. First, received images are filtered to eliminate bad pictures (fuzzy or blurred ones) in phase p_0. The remaining pictures are sent to workers who try to classify them with the help of the SPIPOLL website. If classification is too difficult, the image is sent to an expert. Initial classification is represented by phase p_1 in the workflow, and expert classification by p_2. Pictures that were discarded, classified easily or studied by experts are then assembled in a result dataset D_{out} in phase p_f, to do statistics on insect populations.

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 allows hiring large pools of workers to assemble reliable answers for each micro-task, but in general, a client for a complex task has a limited budget. This forces to replicate micro-tasks in an optimal way to achieve the best possible quality, but without exhausting the given budget. The objective is hence to obtain a *reliable* result, forged through a *complex orchestration*, at a *reasonable cost*.

This paper proposes 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 its input dataset. The output of a phase is used as input for the next ones in the workflow. A complex task terminates when the last of its phases has completed its tagging. For simplicity, we consider simple Boolean tagging tasks that associate a tag in $\{0, 1\}$ to every record in a dataset. Each tagging task on each record is performed by several workers to reduce errors, and the answers are assembled using an aggregation technique. We assume that workers are uniformly paid. For each

record, one of the possible answers (called the *ground truth*) is correct, and an aggregated answer is considered as reliable if its probability to be the ground truth (computed by the agregation technique) 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 a given 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.

Design choices influence realization and quality of workflows realization. First, the chosen aggregation technique influences the quality of the final results. Furthermore, the mechanisms used to hire workers impacts costs and 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 all records 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 an *asynchronous* execution.

We then study execution strategies for complex workflows in different contexts. We consider several types of workflows, different aggregation mechanisms (namely Majority Voting (MV) and Expectation Maximization (EM) [12]), several distributions of data, difficulty of tasks and workers expertize. We evaluate the cost and accuracy of workflows execution in these contexts under static, synchronous and asynchronous assignment of workers to tasks. Unsurprisingly, dynamic distribution of work saves 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 consider data centric models, deployment on crowdsourcing platforms, and aggregation techniques to improve data quality. Due to lack of space, we only mention some of them and refer readers to the long version of this work [14] for a more complete bibliography.

Coordination of tasks has been considered in many languages such as BPMN [22], ORC [16], BPEL [21], or workflow nets [28], 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. Some works propose empirical solutions for complex data acquisition, mainly at the level of micro-tasks [10, 19]. Crowdforge uses Map-Reduce techniques 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. Turkomatic [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 [31] 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 is majority voting (MV), i.e., a mechanism that takes the most returned answer as final result for a tagging task. Several approaches have improved MV by giving more weight to competent workers. Other approaches use aggregation mechanisms based on Expectation Maximization (EM), and consider 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 [32] 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 [32] for a more complete survey of the domain. Zencrowd [6] considers workers competences in terms of accuracy and aggregates answers using EM. Workers accuracy and ground truth are hidden variables that must be discovered in order to minimize the deviations between workers answers and aggregated conclusion. D&S [4] 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 a correct value (i.e., the ground truth) that applies to all types of tasks. EM is used by [24] to discover recall and specificity of workers and propose a maximum-likelihood estimator that jointly learns a classifier, discovers the best experts, and estimates the ground truth. Most of the works cited above consider expertise of workers but do not address tasks difficulty. Approaches such as GLAD [30] or [2] also estimate tasks difficulty to improve quality of answers aggregation on a single batch of Boolean tagging tasks.

A few papers on data aggregation focus on costs optimization. CrowdBudget [26] is an approach that divides a budget B among K existing tasks to replicate them and then aggregate answers with MV. Crowdinc [25] is an EM-based aggregation technique that considers task difficulty, recall and specificity of workers to realize a single batch of micro tasks with a good trade-off between costs and data quality. It computes accuracy of an aggregation, and launches new tasks dynamically. 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 [1], and it uses the aggregation technique of Crowdinc [25] to forge reliable answers.

Some works consider deployment of tasks, i.e., synthesis of strategies to hire workers and parallelize realization of batches of tasks. The objective is to improve costs and latency, i.e., the time needed to treat a complete batch with an optimal deployment. CLAMSHELL [13] focuses on latency improvement. It affects workers to batches of tagging tasks and detects staggers. To speed up tasks completion, some batches are replicated. Pools are assembled and maintained by rewarding workers for waiting. This approach improves latency, but increases costs. [9] uses Markov decision processes to dynamically adapt a pric-

ing policy so that batches of tasks are completed with the lowest latency within a fixed budget, or at the lowest price given some time constraint. [11] proposes a solution to compute the best static deployment policies in order to achieve an optimal utility (i.e., a weighted sum of overall cost and accuracy) using sequencing or parallelization of tasks. This approach is an exhaustive search which limits the number of workers and orchestrations that can be considered. [29] is a recommendation technique for deployments, that allows parallelization of tasks, sequential composition, and use of machines to solve open tasks such as translation or text writing. This approach builds on optimization techniques to find deployments that reduces latency and improves quality of data.

2 Complex Workflows with aggregation

Complex Workflows are inspired by data centric workflows [1], but allow tasks replication, and consider aggregation and budget management. 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 $r(a_1, \dots a_k)$ where each a_i is a value for a field of the record, chosen from a domain Dom_i. One can use First-Order statements with variables denoting fields values to address properties of a record (e.g. write $v_i == true$), or of a set of records in a dataset (e.g. $\exists r(v_1, \dots v_k) \in D, v_k == true$). We will denote by FO^R the FO formulas for records, and by FO^D 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 $r(v_1, \dots, v_k)$ into a new record $r'(v_1, \dots, v_k, v_{k+1})$ 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, phases are not unique micro tagging tasks, but rather replications of batches of tagging tasks allocated to several workers. The returned answers are then aggregated before proceeding to the next phase. 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) it should be forwarded. This allows to split datasets according to the value of a particular field, process differently records depending on their contents, create concurrent threads, etc.

Definition 1 (Complex Workflow). *A complex workflow is a tuple* $W = (\mathcal{P}, \longrightarrow, G, \bigotimes, p_0, p_f)$ *where* $\mathcal{P}$ *is a finite set of phases,* p_0 *is a particular phase without predecessor,* p_f *a phase without successor,* $\longrightarrow \subseteq \mathcal{P} \times FO^R \times \mathcal{P}$ *is a flow relation and* $G : \mathcal{P} \rightarrow FO^D$ *associates a guard to every phase, and for every* $p_x \in \mathcal{P}$, $\bigotimes^x$ *is an operator used to merge datasets input to* p_x.

Intuitively, a phase performs a batch of tagging tasks (one for each record in a dataset), but replicates and distributes them to several workers. The answers

returned by all hired workers are then aggregated to get a final trusted answer. We assume that workers answers are independent. For a triple $(p_x, g_{x,y}, p_y)$ in $\longrightarrow$, we will say that p_x is a predecessor of p_y. We denote by $Succ(p_x) = \{p_y \mid p_x \longrightarrow^* p_y\}$ the set of phases that must occur after p_x, and by $Pred(p_x) = \{p_y \mid p_y \longrightarrow^* p_x\}$ the set of phases that must occur before p_x. The meaning of guard $g_{x,y}$ is that every record produced by phase p_x that satisfies guard $g_{x,y}$ is forwarded to p_y. We will see in the rest of this section that "producing a record" is not done in a single shot, and requires to duplicate a tagging micro-task, aggregate answers, and decide if the confidence in the aggregated answer is sufficient. When a phase p_x has several successors $p_y^1, \dots p_y^k$ and the guards $g_{x,y^1}, \dots g_{x,y^1}$ are exclusive, each record processed by p_x is sent to at most one successor. We will say that p_x is an *exclusive fork* phase. On the contrary, when guards are not exclusive, a copy of each record processed in p_x can be sent to each successor (hence increasing the amount of data processed in the workflow), and p_x is called a *non-exclusive fork* phase. For a phase $p_x \in \mathcal{P}$, we denote by $G_x \in FO^D$ the guard attached to phase p_x. G_x addresses properties of the datasets input to p_x by its predecessors. This allows in particular to require that all records in preceding phases have been processed (we will then say that phase p_x is *synchronous*), that at least one record exists in a dataset produced by a predecessor (the task is then fully *asynchronous*), or more generally satisfaction of any FO expressible property on datasets produced by predecessors of p_x. The operator $\bigotimes^x$ can be either a simple union of datasets, or a more complex join operation. If $\bigotimes^x$ is a join operation, we impose that p_x is synchronous. This is reasonable, as one cannot start processing data produced by a join operation when the final set of records is not known. When $\bigotimes^x$ is a simple union of datasets, as tasks are independent, any record processed on a predecessor of p_x can be forwarded 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 input to p_x must be processed 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, aggregated answers quality and workers competences.

Definition 2. *A* configuration *is a tuple* $C{=}(\mathcal{D}_{in}, W_{in}, W_{out}, conf, B)$ *where*

- $\mathcal{D}_{in} : \mathcal{P} \to Dsets$ *associates a (possibly empty) dataset to every phase* $p_x{\in}\mathcal{P}$.
- $W_{in} : \mathcal{P} \times \mathbb{N} \times \to 2^W$ *is a partial map that associates a set of workers to each record in* $\mathcal{D}_{in}(p_x)$.
- $W_{out} : \mathcal{P} \times \mathbb{N} \times W \to \{0,1\} \cup \emptyset$ *is a partial map that associates a tag or the empty set to a worker, a phase and a record.* $W_{out}(p,n,w)$ *is defined only if* $w \in W_{in}(p,n)$. *We denote by* $l^x_{i,j}$ *the answer returned by worker* w_i *when tagging record* r_j *during phase* p_x.
- $conf : \mathcal{P} \times \mathbb{N} \to [0,1]$ *is a map that associates to each record in* $\mathcal{D}_{in}(p_x)$ *a confidence score in* $[0,1]$ *computed from answers in* W_{out}.

– $B \in \mathbb{N}$ *is the remaining budget.*

$W_{out}(p_x, k, w) = \emptyset$ 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 from $\mathcal{D}_{in}(p_x)$ if there is no worker w such that $W_{out}(p_x, w, r_k) = \emptyset$. As soon as phase p_x is completed for r_k, we can derive an *aggregated answer* $r'_k(v_1, \ldots, v_n, y^x_k)$ for each record $r_k(v_1, \ldots v_n)$ from the set of all answers returned by the workers in $W_{in}(p_x, k)$. Similarly, we can compute a confidence score $conf(p_x, k)$ for value y^x_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 Th, we will say that p_x is *finished* for a record r_k from $\mathcal{D}_{in}(p_x)$ if p_x is completed for r_k and $conf(p_x, k) > Th$. Record $r'_k(v_1, \ldots, v_n, y^x_k)$ will then be part of the input of phase p_y if $(p_x, g_{x,y}, p_y) \in \longrightarrow$ and $r'_k(v_1, \ldots, v_n, y^x_k)$ 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 workers 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, leaving the system ready for the next round. From a configuration $C = (\mathcal{D}_{in}, W_{in}, W_{out}, conf, exp, B)$, a round produces a new configuration $C' = (\mathcal{D}_{in}, W_{in}, W_{out}, conf, exp, B)$ as follows:

– **Answers**: Workers hired in preceding round produce new data. For every phase p_x, every record $r_n \in \mathcal{D}_{in}(p_x)$, 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 $l^x_{i,n} \in \{0, 1\}$ and set $W_{out}(p_x, w_i, n) = l^x_{i,n}$.
– **Aggregation**: The system aggregates answers in every active phase p_x. For every record r_k in $\mathcal{D}_{in}(p_x)$, we compute an aggregated answer y^x_k from the set of answers $A_k = \{l^x_{i,k} \mid w_i \in W_{in}(p_x, n)\}$. We also compute a new confidence score $conf'(p_x, n)$ for the aggregated answer (this confidence depends on the aggregation technique), and evaluate workers expertize and the difficulty of tagging each record(with the algorithm shown in Section 3).
– **Data forwarding**: We distinguish asynchronous and synchronous phases. Let p_y be an asynchronous phase ($\bigotimes^y$ can only be a union of records). Then p_y accept every new record $r'(v_1, \ldots v_k, y^x_n)$ 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^x_n is high enough. Formally, $\mathcal{D}'_{in}(p_y) = \mathcal{D}_{in}(p_y) \cup \{r'(v_1, \ldots v_k, y^x_n)\}$ if $(p_x, g_{x,y}, p_y) \in \longrightarrow$, $conf'(p_x, n) \geq Th$ and $r'(v_1, \ldots v_k, y^x_n) \models g_{x,y}$. 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 $n, r_n \in \mathcal{D}_{in}(p_x), conf(p_x, n) \geq Th$. If there exists a predecessor p_x of p_y that is not closed, then $\mathcal{D}'_{in}(p_y) = \emptyset$. Otherwise we can compute an input for phase p_y as a join over datasets computed by all preceding phases. Formally, $\mathcal{D}'_{in}(p_y) = \bigotimes^y \{D_x \mid p_x \longrightarrow p_y\}$, where $D_x = \{r'(v_1, \ldots v_k, y^x_n) \mid r(v_1, \ldots v_k) \in \mathcal{D}_{in}(p_x) \wedge r'(v_1, \ldots v_k, y^x_n) \models g_{x,y}\}$ i.e., $\mathcal{D}'_{in}(p_y)$ merges data

produced by all predecessors of p_y. Hence, for a synchronous phase p_y, the input dataset is obtained by a join operation computed over datasets filtered by guards, and realized only once preceding tasks have produced all their results. In synchronous and asynchronous settings, a phase p_y becomes active if $\mathcal{D}'_{in}(p_y) \models G(p_y)$. We set $conf'(p_y, n) = 0$ for every new record in $\mathcal{D}'_{in}(p_y)$.

- **Worker allocation**: For every p_x that is active and every record $r_n = r(v_1, \dots v_k) \in \mathcal{D}'_{in}(p_x)$ such that $conf'(p_x, n) < Th$, we allocate k new workers $w_1, \dots w_k$ to record r_n for phase p_x, i.e., $W'_{in}(p_x, n) = W_{in}(p_x, n) \cup \{w_1, \dots, w_k\}$. This number k of workers depend on the chosen policy (see details 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 $W'_{out}(p_x, n, i) = \emptyset$.
- **Budget update**: We then update the budget. The overall number of workers hired is $nw = \sum_{p_x \in \mathcal{P}} \sum_{r_n \in \mathcal{D}'_{in}(p_x)} |W'_{in}(p_x, n) \setminus W_{in}(p_x, n)|$. We consider, for simplicity, that all workers and tasks have identical costs, we hence set $B' = B - nw$.

An execution starts from an initial configuration C_0 in which only p_0 is active, with an input dataset affected to p_0, and begins with workers allocation. Executions end successfully in a configuration C_f where all records in $\mathcal{D}_{in}(p_f)$ are tagged with a sufficient threshold, or fail if they reach a configuration $C \neq C_f$ with a remaining budget $B = 0$. 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 [25], in which workers are hired on demand to increase confidence in aggregated answers. With this policy, the confidence in answers is computed taking into account the estimated expertise of workers, and the difficulty of records 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.

Recall that for a phase p_x, asynchronous execution allows to start processing records as soon as $\mathcal{D}_{in}(p_x) \neq \emptyset$. Conversely, synchronous execution forces p_x to wait for the termination of its predecessors. Choosing a synchronous or asynchronous execution policy may hence influence the time and budget spent to realize a complex task. 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 replication of microtasks, 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. $L_i = \bigcup_{j \in 1 \ldots k} l_{ij}$ denotes the set of answers returned by k workers for a record r_i and $L = \bigcup_{j \in 1..n} L_j$ 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 forwarded to successor phases of p_x, which may launch new phases.

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 1 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 $\xi_i = \{\alpha_i, \beta_i\}$, where α_i is the **recall** and β_i the **specificity** of worker i. The *recall* α_i is the probability that worker i answers 1 when the ground truth is 1, i.e., $\alpha_i = Pr(l_{ij} = 1 | y_j = 1)$. The *specificity* β_i is the probability that worker i answers 0 when the ground truth is 0, i.e., $\beta_i = Pr(l_{ij} = 0 | y_j = 0)$. 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 α_i, β_i 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 $Pr(l_{ij} = y_j | d_j, \alpha_i, y_j = 1) = (1 + (1 - d_j)^{(1-\alpha_i)})/2$ and $Pr(l_{ij} = y_j | d_j, \beta_i, y_j = 0) = (1 + (1 - d_j)^{(1-\beta_i)})/2$.

Figure 1-b) shows probability to get $l_{ij} = 1$ when $y_i = 1$. The horizontal axis represents the difficulty of a task, the vertical axis denotes the probability to get answer $l_{ij} = 1$. 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 workers only make random guesses. 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) [12]. EM is an iterative method that alternates between an expectation (E) step and a maximization (M) step. For a pool of k workers processing n records, we estimate jointly latent variables $(\alpha_i)_{i \in 1..k}$, $(\beta_i)_{i \in 1..k}, (d_j)_{j \in 1..n}$ and derive a set of *final answers* $Y = y_1 \ldots y_n$. We denote

by θ the values of $(\alpha_i)_{i\in 1..k}$, $(\beta_i)_{i\in 1..k}$, $(d_j)_{j\in 1..n}$. In the E-step, we compute for each record r_j the posterior probability of $y_j = 0$ and $y_j = 1$, given the difficulty d_j, workers expertise $(\alpha_i, \beta_i)_{(i\in 1..k)}$ and the answers $L_j = \{l_{i,j} \mid i \in 1..k\}$. In the M-Step, we compute the parameters θ that maximize $Q(\theta, \theta^t)$, the expected value of the log likelihood function, with respect to the estimated posterior probabilities of Y computed during the E-step of the algorithm. Let θ^t 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 find parameters θ that maximize $Q'(\theta, \theta^t) = \mathbb{E}[logPr(L, Y \mid \theta) \mid L, \theta^t]$ (we refer interested readers to [8]-Chap. 9 and [7] for explanations showing why this is equivalent to maximizing $Q(\theta, \theta^t)$). We take as next value for parameters $\theta^{t+1} = \arg\max_{\theta} Q'(\theta, \theta^t)$. This maximization is done with optimization techniques provided by the scipy[4] library. We iterate E and M steps, computing at each iteration t the posterior probability and the parameters θ^{t+1} maximizing $Q'(\theta, \theta^t)$. The algorithm converges, and stops when the difference between two successive joint log-likelihood values is below a threshold (set in our case to $1 \cdot e^{-7}$). It returns values for parameters $(\alpha_i)_{i\in 1..k}$, $(\beta_i)_{i\in 1..k}$, $(d_j)_{j\in 1..n}$. The final answers are the most probable y_j's.

4 Cost Model for Workflow

The objective of a complex workflow W over a set of phases $P = \{p_0, \ldots, p_f\}$ 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_x are distributed to its successors if they meet some conditions on their data. A consequence of this filtering 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 often use static allocation, i.e., fix a number K_s of workers to hire for each micro-task. An obvious drawback of this approach is that the same effort is spent on 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

[4] docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

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 compute a plausible aggregated tag y_j^x for each record r_j from a set of answers L_j^x obtained in each active phase p_x. The algorithm also gives an estimation of difficulty d_j^x (the difficult of the micro-task that consists in tagging record $r_j \in p_x$), and evaluates the expertise level of every worker w_i, i.e., its recall α_i and its specificity β_i. We also obtain a *confidence score* $\hat{c}_j^x$ for the aggregated answer y_j^x. This score is used to decide whether one needs more answers or conversely has to consider y_j^x as a definitive result. Let $k_j^x = |L_j^x|$ denote the number of answers for record $r_j \in p_x$ at a given instant. The *confidence* $\hat{c}_j^x$ in final label y_j^x is defined as:

$$\hat{c}_j^x = \begin{cases} \frac{1}{k_j^x} \cdot \sum_{i=1}^{k_j^x} \left\{ l_{ij}^x \times \left(\frac{1+(1-d_j^x)^{(1-\alpha_i)}}{2}\right) + (1-l_{ij}^x) \times \left(1 - \frac{1+(1-d_j^x)^{(1-\alpha_i)}}{2}\right)\right\} \text{ if } y_j^x{=}1 \\ \frac{1}{k_j^x} \cdot \sum_{i=1}^{k_j^x} \left\{ (1-l_{ij}^x) \times \left(\frac{1+(1-d_j^x)^{(1-\beta_i)}}{2}\right) + (l_{ij}^x) \times \left(1 - \frac{1+(1-d_j^x)^{(1-\beta_i)}}{2}\right)\right\} \text{ if } y_j^x{=}0 \end{cases}$$

Confidence $\hat{c}_j^x$ is a weighted sum of individual confidence of workers in the aggregated result. Each worker adds its probability of answering correctly (i.e., choose $l_{ij}^x = y_j^x$) when aggregating the final answer. This probability depends on y_j^x, but also on worker's competences. If confidence $\hat{c}_j^x$ is greater than a current threshold Th, then answer y_j^x is considered as definitive and the record r_j is closed. Otherwise, the record remains active. We fix a maximal number $\tau \geq 1$ of workers that can be hired during a round for a particular record. Let T_{ar} denote the set of active records after aggregation and D_{max}^x the maximal difficulty for an active record in T_{ar} tagged by phase p_x. For every record $r_j^x \in T_{ar}$ with difficulty d_j^x, we allocate $\mathbf{a}_j^x = \lceil (d_j^x / D_{max}^x) \times \tau \rceil$ new workers for the next round. Intuitively, we allocate more workers to difficult tasks. Now, T_{ar} and hence $\mathbf{a}_j^x$ depend on the threshold computed at each round. An appropriate threshold must consider the remaining budget, the remaining work to do, that depends on the number of records to be processed, on the structure of the workflow, and on the chosen policy. The first parameter to fix for the realization of a workflow is the initial budget B_0. The height and width of a workflow can be used to find a coarse overapproximation of the budget allowing to complete an execution of a workflow. To obtain sharper bound, we first bound the number of remaining phases that a record have to go through to the final phase p_f when it is processed in a phase p_x. We call this number the *foreseeable workload* at phase p_x and denote it by $fw(p_x)$.

Definition 3 (Foreseeable workload). *The foreseeable workload $fw(p_x)$ at phase p_x is the maximal number of phases visited by a record processed in p_x.*

We give an algorithm to compute the foreseeable workload in [14]. Intuitively, it considers the structure of the workflow to compute the number of phases visited between a fork node n_1 and the corresponding merge node n_2: it is the longest path in case of an exclusive fork node, and the total number of nodes between n_1 and n_2 otherwise.

Definition 4 (Foreseeable task number). *Let C be a configuration, and n_x denote the total number of active records at a phase p_x in C. The foreseeable task number from p_x in C is denoted $ft_C(p_x)$ and defined as $ft_C(p_x) = n_x \times fw(p_x)$. The foreseeable task number in C is the sum $FTN(C) = \sum_{p_x \in \mathcal{P}} ft_C(p_x)$.*

Let us now define a threshold function based on the current configuration of a workflow. This function must consider 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 start without waiting for the closure of all records input to preceding phases. A consequence is that in synchronous modes, the decision to hire workers to improve accuracy of answers for a task can be taken *locally* to each phase, while in an asynchronous mode, this decision depends on a *global* view of the remaining work in the workflow. Hence, for a synchronous execution policy, we will define a local threshold computed for each phase, and for an asynchronous execution policy, we will consider a global threshold, computed for the whole workflow.

Asynchronous execution: The execution of a workflow starts from a configuration C_0 with an expected workload $FTN(C_0)$. It is an upper bound, as all records do not necessarily visit this maximal number of phases. We define a global ratio $\Gamma_C \in [0,1]$ of already executed or avoided work in configuration C as $\Gamma_C = \frac{(FTN(C_0)-FTN(C))}{FTN(C_0)}$. Note that at the beginning of an execution, $\Gamma_{C_0} = 0$ as no record is processed yet. When records are processed and moved to successor phases, Γ increases, and we necessarily have $\Gamma_{C_f} = 1$ when no record remains to process in a final configuration C_f. Now, the threshold value has to account for the remaining budget to force the progress of records processing. Let B_0 denote the initial budget at the beginning of execution, and B_C be the budget consumed in configuration C. We denote by β_C the fraction of B_0 consumed in configuration C, i.e., $\beta_C = \frac{B_C}{B_0}$. In the initial configuration , $\beta_{C_0} = 0$. The value of β increases at every round of the execution, and takes value $\beta = 1$ when the whole budget is spent. However, our objective is to end executions with $\beta < 1$. We now define a global threshold value $Th_C \in [0.5, 1.0]$ that accounts for the remaining work and budget.

$$Th_C = \frac{1 + (1 - \beta_C)^{\Gamma_C}}{2} \tag{1}$$

We remind that in a phase p_x, a record r_j with confidence level $\hat{c}_j^x > Th_C$ 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_C is simple: when only a few records remain to be processed, and the remaining budget is sufficiently high, then one can hire more workers. With more contributions, the confidence in aggregated final answers is expected to increase for several records. 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, a phase does not wait for the completion of its predecessors to start. As a consequence, records can be processed in all phases, and we consider a global threshold Th_C, and hence a global policy to hire workers. However, in synchronous execution, records are processed phase by phase, i.e., a phase does not start processing its input dataset until all records in the preceding phases have been processed. Using our global threshold Th_C may produce data with poor quality: as a phase is not launched as long as a preceding phase has an unprocessed record, one can easily meet situations where the larger part of the budget B_{in} is spent to hire workers in the first phases of the workflow, forcing to accept final answers with low confidence in the next phases. 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.

We will say that a task becomes *active* when it starts processing records, i.e., once preceding phases have tagged *all* their records with a sufficient confidence on aggregated answer and the obtained datasets meet guard G_x. We denote by $init(p_x)$ the number of records input to p_x when the phase becomes active. As for asynchronous execution, synchronous execution of a workflow starts from an initial configuration C_0 with an initial budget B_0, and in each configuration C, the remaining budget is denoted by $B_r(C)$. The key idea in synchronous execution is to compute resources needed for each active phase, and to maintain after each round a ratio of input records that still need additional answers to forge a trusted answer, and a local threshold per phase. Let p_x be a phase that becomes active when the execution reaches configuration C. The initial budget allocated to p_x with $init(p_x)$ records in a configuration C is:

$$B_{in}^x = \frac{B_r(C)}{\sum_{p_i \text{ active phase}} FTN(p_i)} \times init(p_x) \tag{2}$$

Intuitively, the remaining budget is shared among active phases to allow termination of the workflow from each phase. Then for each active phase p_x, we maintain the consumed budget B_c^x, and the ratio $\beta^x = \frac{B_c^x}{B_{in}^x}$ of consumed budget. At the end of each round, for each active phase p_x, we compute the ratio of processed tasks

$$\Gamma_C^x = \frac{|\{r_i \mid \hat{c}_i \leq Th^x\}|}{Init(p_x)} \tag{3}$$

where Th^x is the threshold computed at previous round. Obviously, if $\Gamma_C^x = 1$, phase p_x becomes inactive. Otherwise, a local threshold Th'^x for p_x to be used in the next round is computed, using the formula:

$$Th'^x = \frac{1 + (1 - \beta^x)^{\Gamma_C^x}}{2} \tag{4}$$

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

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, τ 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 τ answers, at each round we first apply EM aggregation to estimate the difficulty d_j^x of active records $r_j \in p_x$, $\hat{c}_j^x$ the confidence in the final aggregated answer y_j^x and the recall α_i and specificity β_i of each worker w_i. Then we use a stopping threshold to decide whether we need more answers for each record. In asynchronous execution, the threshold Th is a global threshold, and in synchronous mode, the confidence of each record r_j in p_x is compared to the local threshold Th_x. Records with sufficient confidence are passed to the next phase(s). We hire new workers to obtain more answers for other records. This can increase the confidence level, but also decrease the threshold, as a part of the remaining budget is consumed. Executions stop when the whole budget B_{in} is exhausted or when there is no additional record left to process. Last, the final phase p_f returns the aggregated answer for each record.

Termination: Obviously, when the remaining budget decreases, the threshold(s) decrease too. However, there are situations where the confidence in some answers 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 realized work Γ_C remains unchanged for many rounds. As a consequence, synchronous and asynchronous realization of a workflow may fail. We will see in the experimental results section that even with poor accuracy of workers, this situation was never met. Failure corresponds to situations where the weighted answers of workers remain balanced for a long time. The threshold decreases slowly, and the confidence in aggregated answers remains lower. In that case, when threshold and confidence values 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. Another solution is to limit allocation of workers to tasks with the highest remaining workload. Yet, realization can still fail if records remain stuck in the last but one phase.

5 Experiments and results

In this section, we evaluate execution policies on typical workflows. 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

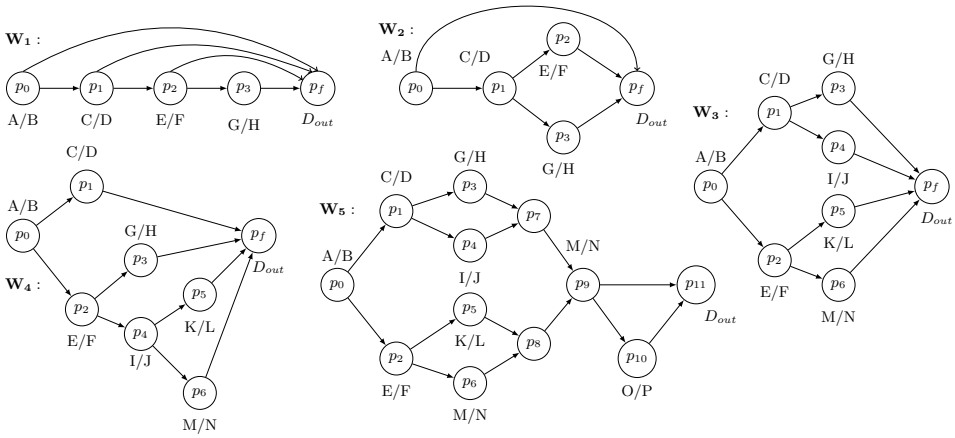

Fig. 2. Five different workflows. W_1: Sequence of phases, W_2: Parallel data transformations followed by an aggregation of results, W_3: Fork-join patterns with uniform lengths of branches, W_4, W_5: Fork-join patterns with nonuniform lengths of branches.

a budget B_0. Crowd workers do not collaborate and hence realize their micro-tasks independently. As there exists no platform to realize complex tasks, 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 execution policies, characteristics of data, and accuracy of workers.

We consider 5 different workflows, represented in Figure 2. 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 micro-tasks that simply add Boolean tags to records. Guards from one phase p_x to the next phase p_y are simple exclusive guards sending each record to one successor, depending on the tag obtained at phase p_x. Formally, guards are FO formulas of the form $f == l_0$ or $f == l_1$, where f is the new field produced by the phase. To avoid unnecessary blocking of workflow progressions, we set $G_x == true$ for every phase $p_x \in \mathcal{P}$. In Figure 2 we depict these choices by pairs of letters (l_0, l_1) representing the binary decision taken on each phase, For example, in workflow W_1, phase p_0 considers two possible tags denoted A and B. Phases p_7, p_8 in workflow W_5 are simple aggregations, and hence are not labeled by choices. After realization of the tasks, if the records are tagged as A by the workers then records are moved to the phase p_f 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 workflows 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 realized during executions depend on workers competences, on the execution policy,

Workflow	W_1	W_2	W_3	W_4	W_5
Parameter	*Value*				
Worker Accuracy	Low	Mid	Average	High	
Value of k_{smv}	10	20	30		
Data Type	Balanced	Unbalanced			
Mechanisms	Static MV	Synchronous	Asynchronous		

Table 1. Evaluation Parameters

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 initially 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 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 according to normal distributions ranging respectively in intervals $[0.2, 0.7]$ ($Expertise_0$, low expertise of workers), $[0.4, 0.9]$ ($Expertise_1$, low to average expertise), $[0.6, 0.99]$ ($Expertise_3$, average expertise) and $[0.8, 0.99]$ ($Expertise_4$, high expertise).

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_{smv} workers to each record in each phase, and aggregates their answers with Majority Voting. We call this policy *Static Majority Voting (SMV)*. A priori, running SMV with a chosen value for k_{smv} should consume a budget lower than $k_{smv}.ft_{C_0}(p_0)$, i.e., $80.k_{smv}.fw(p_0)$. This is however a coarse upper bound for the total budget B_{mv} consumed during the realization of a workflow with an SMV policy, as B_{mv} depends on the execution path followed by records during execution, and hence on random answers of workers. Yet, SMV was shown to be a naive and costly approach in most benchmarks (see for instance [25]), so starting with a budget $B_0 = B_{mv}$ for realization techniques tailored to save costs when accuracy is sufficient is a sensible approach. For each workflow, and for three different values $k_{smv} = 10/20/30$, we performed random runs of SMV to evaluate the maximal budget B_{mv} needed.

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

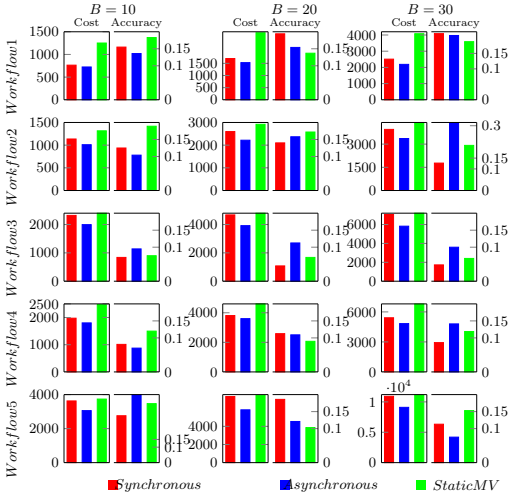

Fig. 3. Budgets and accuracies with low expertize

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 workers. A second interesting (but rather expected) result is that for all realization policies, and for all workflows, executions end with poor accuracy when expertize is low. Consider for instance the results of Figure 3. This Figure gives the consumed budget and achieved accuracy for a given workflow and a given initial budget when workers have a low expertize. The first series of results concern Workflow 1 with a parameter k_{smv} set to 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 W_1 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. $1-b$), 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 workers and as a consequence 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 with unbalanced data are available in [14]).

Next experiments consider mid-level to high expertise, which is a common setting in crowdsourcing. The experiments with competent workers and *synchronous* and *asynchronous* execution policies clearly show that dynamic alloca-

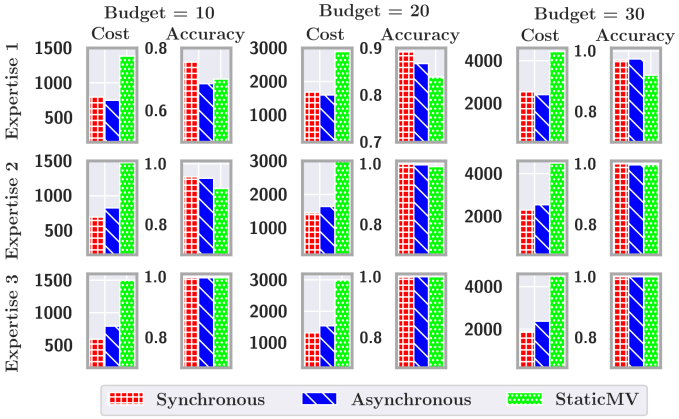

Fig. 4. Workflow 1 on Balanced Data

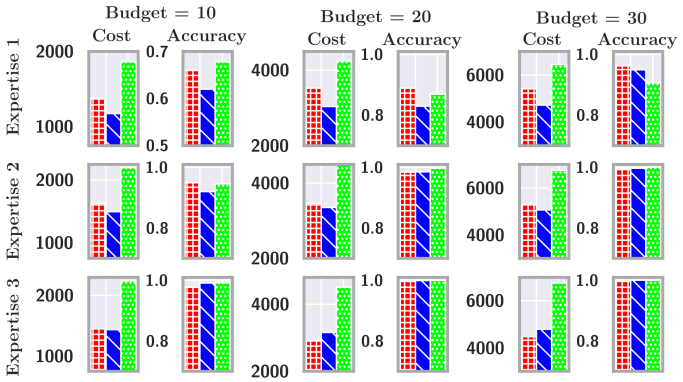

Fig. 5. Workflow 1 on Unbalanced Data

tion schemes outperform the SMV approach both in terms of cost and accuracy. One can easily see these results Figures 4 and 5, that represent executions of workflows W_1 with three levels of expertize, 3 values of k_{smv} (and hence 3 different initial budgets), and all execution policies, respectively for balanced and unbalanced data. We show similar results in [14] for workflows W_2, W_3, W_4, W_5.

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 compute the final answers by weighting individual answers according to worker's expertise. This makes EM-based evaluation of final answers more accurate than SMV. This improvement already occurs at the level of a single phase execution (this was also the conclusion of [25]). The reasons for cost improvement with respect to SMV are also easy to figure. SMV allocates a fixed number of workers to every record in every phase of

a workflow, whereas synchronous and asynchronous execution schemes allocate workers on-the-fly based on a confidence level which depends on the difficulty of tasks, workers expertise, and returned answers. By comparing confidence levels with a dynamic threshold, workers allocation considers the remaining budget and workload as well. This clever allocation of workers saves costs, as easy tasks call for the help of fewer workers than the fixed number imposed by SMV. The resources that are not used on easy tasks can be reused later for difficult tasks, hence improving accuracy.

These results were expected. A more surprising outcome of the experiment is that in most cases synchronous execution outperforms asynchronous execution in terms of accuracy. The intuitive reason behind this result is that the way records are spread in the workflow execution affects the evaluation of expertize and difficulty. The synchronous execution realizes tasks in phases, while asynchronous execution starts tasks independently in the whole workflow. A consequence is that evaluation of hidden variables such as the difficulty of tasks and workers expertise in the EM aggregation improves with a larger number of records per phase in synchronous execution, while it might remain imprecise when the records are spread in different phases during an asynchronous execution. This precise estimation helps synchronous execution to allocate workers as well as to derive the final answers more efficiently and hence outperform asynchronous execution. A third general observation is that both synchronous and asynchronous executions need a greater budget to complete a workflow when data is unbalanced. Observe the results in Figure 4 and Figure 5: the budgets spent are always greater with unbalanced data. A possible explanation is that with balanced data, records are sent uniformly to all phases, which helps evaluation of workers expertize and difficulty of tasks, while with unbalanced data, some phases receive only a few records, which affects evaluation of hidden variables.

Unsurprisingly (see for instance Figure 4), for a fixed budget, when worker expertise increases, accuracy increases too, and consumed budget decreases. Competent workers return correct answers, reach a consensus earlier, and hence achieve better accuracy faster. Similarly for a fixed expertise level, increasing the initial budget increases the overall accuracy of the workflow. Again, the explanation is straightforward: a higher budget increases the threshold used to consider an aggregated answer as correct, giving better accuracies. To summarize, for a fixed initial budget and high enough expertize, synchronous and asynchronous policies usually improve both cost and accuracy.

6 Conclusion

This work has proposed a model to realize complex tasks with the help of a crowd of workers. It fosters on the advantages of crowdsourcing systems and workflow. A particular attention is paid to quality of the data produced, and to the overall cost of complex tasks realization. We have compared several task distribution strategies through experiments and showed that dynamic distribution of work outperforms static allocation in terms of cost and accuracy.

A short-term extension is to consider termination of complex tasks realization with dynamic policies. Indeed, workflows realized with dynamic policies may not terminate: this happens when the guard associated with a phase is never satisfied, or when for some record, all workers agree to return the answers that do not increase the confidence. However, this latter situation was never met during our experiments, even with low expertize of workers. The probability of non-terminating executions with synchronous/asynchronous policies seems negligible. In our future work, we plan to demonstrate formally that $\mathbb{P}(B_r = 0 \wedge FTN(C) > 0)$, the probability of reaching a configuration 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

1. Bourhis, P., Hélouët, L., Miklos, Z., Singh, R.: Data centric workflows for crowdsourcing. In: Proc. of Petri Nets 2020. pp. 46–61 (2020)
2. Dai, P., Lin, C.H., Weld, D.S.: Pomdp-based control of workflows for crowdsourcing. Artificial Intelligence **202**, 52–85 (2013)
3. Daniel, F., Kucherbaev, P., Cappiello, C., Benatallah, B., Allahbakhsh, M.: Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions. ACM Comput. Surv. **51**(1), 7 (2018)
4. Dawid, A., Skene, A.: Maximum likelihood estimation of observer error-rates using the em algorithm. J. of the Royal Statistical Society: Series C (Applied Statistics) **28**(1), 20–28 (1979)
5. Deguines, N., Julliard, R., De Flores, M., Fontaine, C.: The whereabouts of flower visitors: contrasting land-use preferences revealed by a country-wide survey based on citizen science. PloS one **7**(9), e45822 (2012)
6. Demartini, G., Difallah, D., Cudré-Mauroux, P.: Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In: Proc. of WWW 2012. pp. 469–478. ACM (2012)
7. Dempster, A., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. J. of the Royal Statistical Society: Series B (Methodological) **39**(1), 1–22 (1977)
8. Flach, P.: Machine Learning - The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press (2012)
9. Gao, Y., Parameswaran, A.G.: Finish them!: Pricing algorithms for human computation. Proc. VLDB Endow. **7**(14), 1965–1976 (2014)
10. Garcia-Molina, H., Joglekar, M., Marcus, A., Parameswaran, A., Verroios, V.: Challenges in data crowdsourcing. Trans. on Knowledge and Data Engineering **28**(4), 901–911 (2016)

11. Goto, S.and Ishida, T., Lin, D.: Understanding crowdsourcing workflow: Modeling and optimizing iterative and parallel processes. In: Proc. of HCOMP 2016. pp. 52–58. AAAI Press (2016)
12. Gupta, M., Chen, Y.: Theory and use of the em algorithm. Foundations and Trends in Signal Processing **4**(3), 223–296 (2011)
13. Haas, D., Wang, J., Wu, E., Franklin, M.J.: Clamshell: Speeding up crowds for low-latency data labeling. Proc. VLDB Endow. **9**(4), 372–383 (2015)
14. Hélouët, L., Miklos, Z., Singh, R.: Cost and Quality Assurance in Crowdsourcing Workflows (Oct 2020), https://hal.inria.fr/hal-02964736, extended Version
15. Karger, D., Oh, S., Shah, D.: Iterative learning for reliable crowdsourcing systems. In: Proc. of NIPS'11. pp. 1953–1961 (2011)
16. Kitchin, D., Cook, W., Misra, J.: A language for task orchestration and its semantic properties. In: Proc. of CONCUR'06. pp. 477–491 (2006)
17. Kittur, A., Smus, B., Khamkar, S., Kraut, R.: Crowdforge: Crowdsourcing complex work. In: Proc. of UIST'11. pp. 43–52. ACM (2011)
18. Kulkarni, A., Can, M., Hartmann, B.: Collaboratively crowdsourcing workflows with turkomatic. In: Proc. of CSCW'12. pp. 1003–1012. ACM (2012)
19. Li, G., Wang, J., Zheng, Y., Franklin, M.: Crowdsourced data management: A survey. Trans. on Knowledge and Data Engineering **28**(9), 2296–2319 (2016)
20. Little, G., Chilton, L., Goldman, M., Miller, R.: Turkit: tools for iterative tasks on Mechanical Turk. In: Proc. of HCOMP'09. pp. 29–30. ACM (2009)
21. OASIS: Web Services Business Process Execution Language. Tech. rep., OASIS (2007)
22. OMG: Business Process Model and Notation (BPMN). OMG (2011)
23. Quinn, A., Bederson, B.: Human computation: a survey and taxonomy of a growing field. In: Proceedings of the SIGCHI conference on human factors in computing systems. pp. 1403–1412 (2011)
24. Raykar, V.C., Yu, S., Zhao, L., Valadez, G., Florin, C., Bogoni, L., Moy, L.: Learning from crowds. J. of Machine Learning Research **11**(Apr), 1297–1322 (2010)
25. Singh, R., Hélouët, L., Miklós, Z.: Reducing the cost of aggregation in crowdsourcing. In: Proc. of ICWS'20 (2020)
26. Tran-Thanh, L., Venanzi, M., Rogers, A., Jennings, N.: Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks. In: Proc. of AAMAS'13. pp. 901–908 (2013)
27. Tsai, C.H., Luo, H.J., Wang, F.J.: Constructing a bpm environment with bpmn. In: 11th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'07). pp. 164–172. IEEE (2007)
28. Van Der Aalst, W., van Hee, K., ter Hofstede, A., Sidorova, N., Verbeek, H., Voorhoeve, M., Wynn, M.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects of Computing **23**(3), 333–363 (2011)
29. Wei, D., Roy, S., Amer-Yahia, S.: Recommending deployment strategies for collaborative tasks. In: Proc. of the 2020 International Conference on Management of Data, SIGMOD Conference 2020. pp. 3–17. ACM (2020)
30. Whitehill, J., Wu, T., Bergsma, J., Movellan, J., Ruvolo, P.: Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In: Proc. of NIPS'09. pp. 2035–2043 (2009)
31. Zheng, Q., Wang, W., Yu, Y., Pan, M., Shi, X.: Crowdsourcing complex task automatically by workflow technology. In: MiPAC'16 Workshop. pp. 17–30 (2016)
32. Zheng, Y., Li, G., Li, Y., Shan, C., Cheng, R.: Truth inference in crowdsourcing: Is the problem solved? Proc. of VLDB Endowment **10**(5), 541–552 (2017)