Efficiently identifying a well-performing crowd process for a given problem

De Boer, Patrick ; Bernstein, Abraham

Abstract: With the increasing popularity of crowdsourcing and crowd computing, the question of how to select a well-performing crowd process for a problem at hand is growing ever more important. Prior work casted crowd process selection to an optimization problem, whose solution is the crowd process performing best for a user's problem. However, existing approaches require users to probabilistically model aspects of the problem, which may entail a substantial investment of time and may be error-prone. We propose to use black- box optimization instead, a family of techniques that do not require probabilistic modelling by the end user. Specifically, we adopt Bayesian Optimization to approximate the maximum of a utility function quantifying the user's (business-) objectives while minimizing search cost. Our approach is validated in a simulation and three real-world experiments. The black-box nature of our approach may enable us to reduce the entry barrier for efficiently building crowdsourcing solutions.

DOI: https://doi.org/10.1145/2998181.2998263

Efficiently identifying a well-performing crowd process for a given problem

Patrick M. de Boer
University of Zurich
Zurich, Switzerland
pdeboer@ifi.uzh.ch

Abraham Bernstein
University of Zurich
Zurich, Switzerland
bernstein@ifi.uzh.ch

ABSTRACT
With the increasing popularity of crowdsourcing and crowd computing, the question of *how to select a well-performing crowd process for a problem at hand* is growing ever more important. Prior work casted crowd process selection to an optimization problem, whose solution is the crowd process performing best for a user's problem. However, existing approaches require users to probabilistically model aspects of the problem, which may entail a substantial investment of time and may be error-prone. We propose to use black-box optimization instead, a family of techniques that do not require probabilistic modelling by the end user. Specifically, we adopt Bayesian Optimization to approximate the maximum of a utility function quantifying the user's (business-) objectives while minimizing search cost. Our approach is validated in a simulation and three real-world experiments.
The black-box nature of our approach may enable us to reduce the entry barrier for efficiently building crowdsourcing solutions.

Author Keywords
Crowd process design; Human computation algorithms; Crowd Sourcing; Collective Intelligence

ACM Classification Keywords
H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces – Computer-supported cooperative work.

INTRODUCTION
In recent years, human computation attracted the interest of the CSCW community [9,20,37], since the computer-mediated coordination of crowds can be seen as an example of computer-supported cooperative work. The question of identifying the optimal coordination structure ("*crowd process*") for a problem at hand caused prior work [12,36] to use methods from the field of optimization. These approaches rely on probabilistic modelling of (sub-)tasks of the problem or of its workers. The drawback of such white-box optimization based methods is, that modelling by the end-user can be high-effort and error prone. Additionally, practitioners who do not possess probabilistic modelling skills are widely unable to use them. For such practitioners, it is therefore often unclear how to efficiently design an inexpensive and accurate crowd process for a task at hand. Many resort to a manual approach, where they may rely on the various design patterns that have been proposed in literature. Unfortunately, it is often unpredictable which pattern works well for a given problem, since their performance, and hence their applicability, varies by problem setting [7].

This is costly and time-consuming. PPLib [7] addresses this problem. It is a method and tool for systematic crowd process design, that, given a problem definition, automatically derives a set of suitable candidate crowd processes. However, after arriving at a set of candidates, PPLib's approach to identifying the best suited crowd processes among them is very expensive. It entails the (repeated) execution of the whole set of applicable candidate crowd processes for their evaluation through a user-defined utility function. The goal of this paper is a *widely accessible method to drastically reduce the cost of crowd-process selection* (i.e., the selection of a well-performing process from a set of candidates), such that the PPLib methodology can be employed as a more efficient way to systematically design crowd processes. To that end we propose to employ global (black-box) optimization for what is called Auto-Experimentation. The goal is to approximate the optimal parameterization of a crowd process – according to a given utility function. Parameters in this optimization include the kinds of patterns employed (e.g. Find-Fix-Verify [27]), the coordination processes utilized [24], the crowd market to use (e.g., Mechanical Turk or CrowdFlower), as well as simple parameters such as the number of workers involved in a majority vote. Specifically, we propose the adoption of Bayesian optimization [28], as it has been shown to find near-optimal solutions in non-deterministic settings whilst minimizing the number of samples (or actual process executions in our application). The key contributions of this paper are:

- The use of black-box optimization for crowd process selection
- An efficient selection method robust to the non-determinism of crowd processes.
- A publicly[1] available Open Source system implementing this new Auto-Experimentation strategy.

CSCW '17, February 25-March 01, 2017, Portland, OR, USA

ACM 978-1-4503-4335-0/17/03…$15.00
DOI: http://dx.doi.org/10.1145/2998181.2998263

RELATED WORK

This paper continues and combines research streams in the fields of human computation, Auto-Experimentation, and global optimization.

Human computation patterns

A popular approach to scale the process of solving large problems in the field of crowdsourcing, is to harness smart task decomposition. For example, a crowd process translating a book could divide the book into its paragraphs, for each paragraph concurrently ask a crowd worker to provide a translation, and then compose a translated version out of all proposed paragraphs.

If one were to use the translation process outlined above, one would get a book with paragraphs of varying quality – some of which may be unsatisfying. Therefore, various *design patterns for quality assurance* have been proposed: For instance, *Find-Fix-Verify* [27] asks crowd workers to first select an item that needs improvement within a list of items and then employs multiple crowd workers to suggest alternatives to this item followed by multiple crowd workers selecting the best alternative in a majority vote. If the list of items to be worked on is large and inter-dependent, *Context-Trees* could be used instead [34]. More generally, Malone et al. [25] describe the *Contest* pattern as asking multiple crowd workers to propose an answer to a given question, followed by a majority vote to nominate the best answer. The *Iterative Refinement* [22] pattern iterates such contests until a majority vote decides that the improved version is not better than the version of the previous iteration. *Statistical methods for quality assurance* aim at weighting answers by crowd workers trust worthiness [18], or estimate the number of distinct crowd answers needed for a given problem [2,14,23].

Auto-Experimentation for Crowd computing

How can designers determine which design pattern is ideal for a problem at hand? Auto-Experimentation Engines, systems that autonomously plan and run the evaluation of a given set of hypothesis', may be adopted towards this cause. They have been used successfully in biology [19], machine learning [31], and, most recently, crowd computing [7]. Specifically, de Boer and Bernstein proposed PPLib, a method and tool capable of automating large parts of crowd process design by viewing crowd process suitability to a given problem as a hypothesis to be answered through Auto-Experimentation.

In order to employ PPLib, a crowd process designer needs to first specify the problem deep structure, i.e. the most abstract way of formulating a problem [10], using operators available through PPLib's Process Repository (PPR). PPLib will then automatically recombine existing crowd process fragments (some of which are part of the design patterns listed above) to arrive at an often large set of candidate processes suitable for the given problem definition [4]. When supplied with a user-defined utility function, PPLib's Auto-Experimentation module can then be employed to automatically find the best crowd process among these candidates by executing them and assessing their result desirability with the provided utility function. This function can incorporate various parameters to grade an individual crowd process-execution, e.g. crowd process result, cost or run time. Figure 1 shows the general workflow of using PPLib.

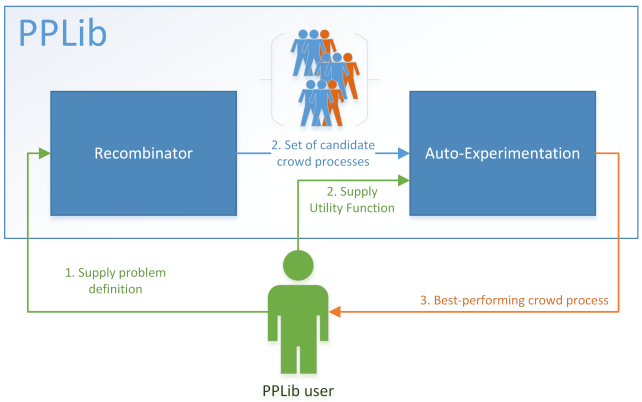

Figure 1: The PPLib approach

In PPLib, Auto-Experimentation works by taking a sample of the task's input data (e.g. if the task is to translate a book, a sample could be a few randomly selected pages of the book), testing the performance of a candidate crowd processes on the sample, and evaluating its results (along with other parameters, such as cost and duration) using the user-defined utility-function. The best crowd process is then selected as the one with the highest result of the utility function. PPLib's Auto-Experimentation module has initially been proposed with a single strategy: Naïve Auto-Experimentation (NAE), which simply executed all candidate crowd processes and compared their (average) results. By default, the NAE-strategy executes each process candidate 4 times, and returns the process with the highest median utility of its results. It repeats a candidate process' execution, because crowd processes are non-deterministic, i.e. their performance characteristics may vary with each execution due the cognitive- and error diversity of human actors [5] or other stochastic elements of the process. In the following, we denote the number of repetitions used in a NAE variant by appending it, e.g. NAE-4 for the default setting. This naïve approach to process evaluation presented by NAE is very expensive and reduces the usefulness of the PPLib method to settings where the costly experimentation can be amortized (e.g., over many runs or few executions with a very high payoff).

Optimization for workflow selection in Crowd Sourcing

The naïve approach to crowd process selection is expensive. There has therefore been interest to cast workflow selection to an optimization problem. Zhang et al. [36] compose crowd processes from generic tasks that

[1] Integrated into https://github.com/uzh/PPLib

resurface in different combinations. Using a manually created (probabilistic) model for task performance, one can then estimate the utility of all crowd processes compiled of these generic tasks. This potentially enables finding optima even for large sets of candidate workflows. However, their work is limited to tasks with finite outcomes such as Majority Votes and currently does not support open-ended outcomes such as writing texts. Dai et al. [12] use a decision-theoretic approach with Partially-Observable Mixed Markov models to capture crowd worker accuracy for repeated labelling in face of noisy workers.

Both approaches face challenges in practical applications as they require to manually build (probabilistic) performance models for tasks (Zhang et al) or for workers (Dai et al). More generally, these so-called white-box approaches are tied to specific crowd processes and would need to be adapted to support further processes. Wider adoption of such techniques might therefore be constrained, as probabilistic modeling can be high effort and error prone. To address this challenge, Weld et al [35] suggested to infer such models using machine learning techniques in a system named Clowder. Although promising, an evaluation and detailed description of Clowder are still pending.

Our method, in contrast, sidesteps the need for modeling by employing a pure black-box approach. This has the advantage that our approach is not tied to specific crowd processes and can therefore be applied to many problems out-of-the-box. The lower entry barrier aims to open optimized crowd process selection to a wider audience. However, the main disadvantage of our approach when compared to white-box methods, is that it only approximates optima rather than being guaranteed to find optima. In practice, this means that our approach nominates good processes as opposed to always identifying the best one. The validation section will quantify this effect.

Bayesian Optimization

In the realm of optimization, one aims to find either the minimum or the maximum value of a mathematical function, called *objective function*. Bayesian Optimization is an optimization technique which is especially useful when applied to objective functions that are expensive to evaluate and for which the closed-form expression is not available (i.e. no derivative is available). It has been applied to a plethora of problems, e.g. classifier hyper parameter optimization [33], robotics [26] and the configuration of a distributed computing framework [15]. The evolution of Bayesian Optimization to a general-purpose optimization strategy was supported by multiple high-quality frameworks implementing it, among them Spearmint [32], TPE [3] and SMAC [17].

Bayesian Optimization is based on two components: the *surrogate function*, approximating the (black box) objective function, and the *acquisition function*, used to determine where to sample next. The algorithm iterates the steps of obtaining the next point to sample by maximizing the acquisition function, followed by the costly sampling of the objective function and updating the surrogate with the resulting posterior. The acquisition function aims to trade off between exploration (i.e., where the surrogate's estimate is particularly uncertain) and exploitation (i.e., where the objective function is expected to be particularly high). Various strategies for the acquisition function have been proposed, such as Probability of Improvement (PI), Expected Improvement (EI), or GP Upper Confidence Bound. In this paper, we use EI, since it offers a good tradeoff between exploration and exploitation. For a more in-depth introduction, we would like to refer the reader to the tutorial by Brochu et al. [8] and Jonas Mockus' thorough book on Bayesian Optimization [28].

METHOD: PROCESS DESIGN FOR OPTIMIZATION

The goal of this paper was to find a method to efficiently identify a particularly fitting crowd process for a given problem from a set of candidate crowd processes. A simple, albeit inefficient, approach would be to execute all candidates and rank them according to how well their result fits the expectation of the task requester. To formalize her expectations, a task requester could specify a *utility function* μ, which takes all relevant parameters of an executed crowd process such as input/cost/duration and returns a numeric utility of how well a given crowd process execution fits the task requesters business objectives. Ways of quantifying these business objectives for μ can be borrowed from the field of decision analysis, e.g. judgment bootstrapping [13].

Additionally, we introduce an *execution function* σ, whose input is an executable *crowd process* c. σ executes the supplied crowd process c and uses μ to evaluate c's results. By definition, σ's value is highest for the crowd process best catering to the task requesters goals defined in μ.

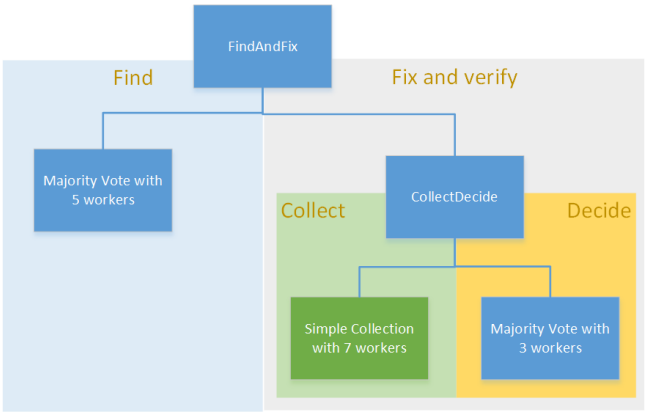

Figure 2: Tree-based visualization of the composition of an example variant of Find-Fix-Verify in PPLib [7]

Concrete crowd processes (or processes in general) can be seen as a composition of crowd process fragments as well as a specification of the fragments' possible parameters [7]. Figure 2 shows the composition of the popular Find-Fix-Verify [27] pattern from fragments, where, for example, the

additional parameter for the number of crowd workers finding a relevant element has been set to "5". Hence, every concrete crowd process c is defined by the values of its *parameters* P_c that specify the process fragments used and the concrete values for the fragments' parameters. For example, in Find-Fix-Verify (Figure 2), P_c includes categorical parameters such as a coordination mechanism (e.g. "Majority Vote") and numerical ones (such as the above-mentioned number of votes). Note that the size of P_c varies as some crowd processes may require more parameters than others. P_* denotes the set of all parameters known to the system, whereas each P_c therefore contains a subset of P_*. Each P_c contains one parameter that determines the root of the process composition. The process defined in Figure 2, for example, specifies *FindAndFix* as its root in its corresponding parameters. The *composition function* φ is then used to compose a crowd process based on a given subset of parameters from P_*. Essentially, given φ, the domain of P_* defines the *process design space*: a multi-dimensional space of possible process designs for a given problem. φ is generic, as it can construct any process specified via the parameters in P_c.

An optimizer can then be instructed to explore the process design space by varying the parameters to find the maximal value of σ. More formally, in each iteration i, an optimizer systematically choses a combination of parameter values V_i for the parameters from P_* relevant in iteration i and executes $\sigma(\varphi(V_i))$. Note, given that V_i also determines the structure of the process via some parameters. It stops iterating when it cannot reliably improve the maximally observed utility value or when a predefined number of iterations is reached. It then returns the parameter set that resulted in the highest value of σ as the best process for the user's µ. As an optimizer, we use Bayesian Optimization [8], where the objective function is modelled using a Gaussian Process (GP). Intuitively, a GP is a distribution over functions, which means that each data point on a GP (i.e. *a crowd process' utility*) is modelled by a Normal Distribution with its own variance. We use this property to encapsulate both, the optimizers uncertainty about the data point and the corresponding crowd process' performance's variance. Hence, the crowd processes non-determinism (i.e., its performance variance) is handled naturally by the GP. As the optimizer runs, it uses EI (see related work) to tradeoff the GP's current variance of different data points in the search space and exploitation of the acquired knowledge.

BOA: BAYESIAN OPTIMIZED AUTO EXPERIMENTATION

Based on the framework introduced in the past section, we now introduce its implementation within a new module in PPLib called BOA. For a step-by-step tutorial we would like to refer to BOA's user manual[1] instead.

OVERVIEW

PPLib first (A) derives a set of candidate crowd processes based on the problem deep structure via recombination. The set of candidate processes is then used (B) to inform the optimizer about the parameter space P_* and its bounds. Once started, the optimizer (C) iteratively requests samples of the crowd process design space for a given set of parameter values V_i. (D) Once the optimizer reaches convergence, the most reliable process maximizing the users µ is returned. We will now visit each of these steps in detail.

(A) Problem Definition & Recombination

Following the PPLib-approach (see Figure 1), the deep structure of the crowd sourced problem needs to be described using abstract operators. PPLib's recombination mechanism can then be employed to automatically derive a set of candidate crowd processes for the supplied problem definition. The result of this procedure is a set of all valid V_i's.

For a more in-depth explanation of Recombination, deep structures and the PPLib-method in general, we would like to refer to [7], since this paper's focus is only on Auto-Experimentation.

(B) Configuring the optimizer

Given the set of all valid V_i's, the optimizer can be instructed about the dimensions and parameter bounds of the crowd process design space. Specifically, the dimensions of this space are given by a set of the unique parameter definitions of all surface structures, including their nested crowd process fragments. This entails that all

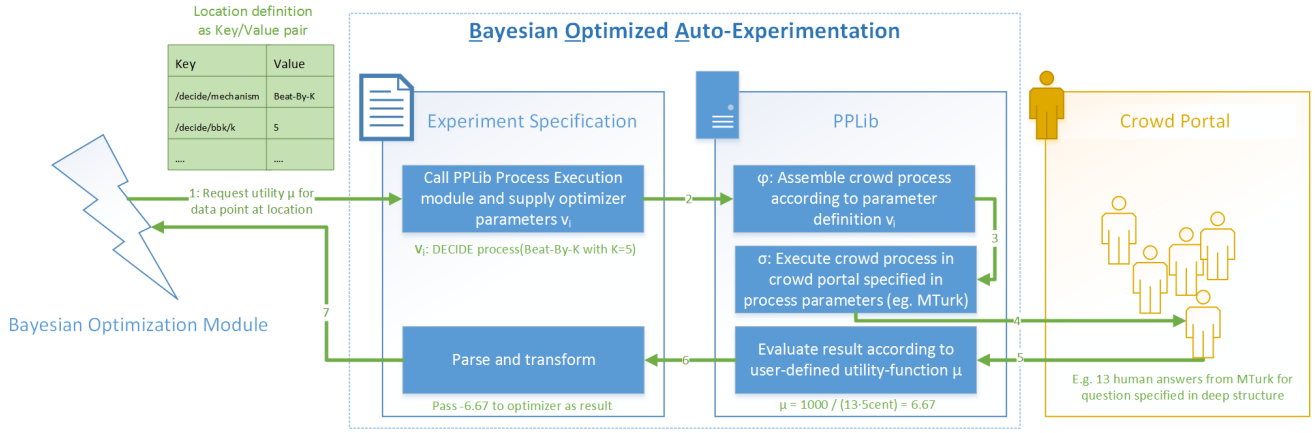

Figure 3: Workflow of (C) Running the optimizer with the use case of the optimizer requesting the utility for a crowd process using Beat-By-K (BBK) with K=5. The utility µ in the example is calculated by accuracy / cost, whereas accuracy of crowd answers is 1000

parameter declarations are mapped to a *hierarchical key* that uniquely identifies them. For example, the parameters for the Collection-fragment in Figure 2 (box highlighted in green), may be prefixed with their parent's key */FindAndFix/CollectDecide/Collection/*. Hence, each surface structure is represented by a data point in the optimizer's search space, whereas its coordinates are given by its (hierarchy-transformed) parameters as well as all other dimensions of the space set to 0.

(C) Running the optimizer

Once the optimizer is launched, it continuously samples the objective function at different locations by executing $\sigma(\varphi(V_i))$. In contrast to NAE, which iterated through all valid unique V_i's, the optimizer chooses the next V_i to sample by maximizing its acquisition function. It then composes and executes the corresponding crowd process $\varphi(V_i)$ using actual workers. The result is evaluated using σ and returned to the Bayesian Optimization Module (Spearmint [32]). The choice for the next sampling point is done using the common *Expected Improvement* strategy with an *ARD Matérn 5/2 covariance Kernel* in the Bayesian Optimization Module (please see [32] for more details and a comparison between strategies). Figure 3 shows the interactions between the different modules of BOA. Note that BOA supports maximization and minimization of objective functions. In the following we designate maximization objective functions by μ and minimization objective functions by $-\mu$.

(D) Optimizer convergence

We stop the optimizer after it was unable to produce an increase in the highest observed utility during a predefined number of iterations. Experimentally, 20 seemed to offer a good tradeoff between cost and quality. After stopping, the process BOA estimated to reliably lead to the highest utility is returned. As our heuristic to find this reliably best performing process, BOA uses all samples of the objective function obtained by the optimizer to (i) determine its best encountered value and (ii) retrieve all sampled crowd process candidates with values within one standard deviation of this value. Depending on the user's preference between process stability vs quality, one could also include more (or less) process candidates. To avoid negative outliers, we calculate standard deviation across all samples that are better than the median sample. From this set, we (iii) choose the parameter combination V_i that exhibited the lowest utility variation across multiple executions giving preference to processes with multiple executions, where we assumed the distribution is sampled more precisely.

EVALUATION

The central claims of this paper are that crowd process selection can be efficiently solves through black-box optimization. Consequently, we need to show that solving this optimization problem does indeed lead to good solutions and that the Bayesian Optimization based BOA does so more efficiently (e.g. 10x less expensive) than the only black-box alternative known to us: NAE. Hence, addressing these claims in order, the following hypotheses need to be evaluated:

H1: BOA nominates processes of comparably high quality as NAE as measured by μ

H2: Running BOA is significantly less expensive than NAE

H3: For a real world problem, BOA can find processes of equal or better quality (as measured by μ) than expert designed ones at 10% of the cost that NAE would have

One of the main difficulties when evaluating crowd process performance is the lack of ground truth: Since crowd processes are inherently non-deterministic, using their repeated executions can only sample their true performance. This problem is aggravated by the large cost in obtaining samples from crowd processes in a real-world setting. We therefore decided to run our first evaluation in a simulated setting, where we can compare thousands of processes (almost) for free, and where we can therefore approximate ground truth by re-executing each individual candidate process many times and using its median utility. This simulation allows us to answer H1 and H2 within the limitations of a simulated world. To ensure that these results generalize to the real world, the 2nd experiment investigates H1 and H2 in the context of a typical crowdsourcing task: text-shortening. This experiment also evaluates H3 in a problem with a small set of crowd process candidates. To show that BOA scales to a large set of crowd process candidates, we ran a 3rd (and 4th) experiment, where we replicated the Bayesian Truth Serum setting [29] that entailed 212 crowd process candidates, which also allows us to generalize H3.

Note, that in all evaluations, we specified our goals using a cost function $-\mu$ (to be minimized) instead of a utility function μ (to be maximized), since a cost function seemed more intuitive to explain the concepts. All real-world experiments were executed on Amazon Mechanical Turk, NAE was executed sequentially, compensation (if not stated differently) for multiple choice questions was 5 cents, free-text questions were priced at 15 cents. We only used US workers with less than 4% rejected HITs and more than 4000 approved HITs. All experiments were randomized and conducted on mornings of working days Eastern Time.

Experiment #1: BOA in a simulated environment

The goal of the simulation was to allow us to test H1 and H2, while, by running many repetitions of BOA and NAE, assessing their mean performance. To that end, we simulated the task of asking a group of crowd workers a multiple-choice question with 4 possible answers and aggregating these to elicit the correct answer. This is a typical crowdsourcing pattern used, e.g., to tag images or in text sentiment analysis. The simulation consisted of four experimental conditions (C1-C4), corresponding to different levels of bias or uncertainty for the crowd choosing the correct answer. As shown in Table 1, each

condition used a different set of probabilities for an answer to be chosen by a simulated crowd-worker, where the "correct" answer is always the one with the highest probability in a condition. Note that we have chosen the probabilities for a correct answer to be selected to decrease in each condition in favor of the others. This simulates higher levels of uncertainty about the correct answer. For example, in a multiple choice question with four options having the respective probabilities of 1%, 1%, 1% and 97% of being picked by a simulated crowd worker, an accordingly biased dice is rolled to decide which answer an individual agent will select.

	Answer 1	Answer 2	Answer 3	Answer 4
C1	1%	1%	1%	97%
C2	10%	10%	10%	70%
C3	20%	20%	20%	40%
C4	24%	24%	24%	28%

Table 1: Probabilities for each item to be picked in the different experimental conditions C1-C4

For all experimental conditions we employed PPLib to identify the crowd process that reliably leads to the correct answer. Each condition employed one DECIDE-type building block of PPLib, which is used to elicit the answer to a multiple choice decision, as problem deep structure [7]. Specifically, we set up PPLib to vary the following parameters:

- The coordination mechanism between crowd workers (Majority Vote, Beat-By-K [16], Confidence-based Voting [2])
- The number of (simulated) crowd workers to use (between 1-10)
- K in Beat-By-K (between 1-10)
- The confidence value in Confidence-based Voting (between 0.65 and 0.95)
- The maximal amount of iterations in Beat-By-K and confidence-based voting (between 20 and 30)

The variation of these parameters on the DECIDE block lead PPLib's Recombinator to generate a set of 208 candidate crowd processes (or surface structures). For more information on recombination, please refer to the related work section and the PPLib paper [7].

Following the PPLib workflow, either a utility function or a cost function needs to be defined to enable Auto-Experimentation to identify the best suited crowd process from the candidates. We defined the cost function $-\mu$ to comply with our goal to favor answers closer to the "correct" one (i.e., the answer with the highest probability for being selected in the experimental condition) as:

$$-\mu(c_s, k_s) = \max_i(\pi(c_i)) - \pi(c_s) + k_s,$$

where the c_i's are the possible answers of the given experimental condition (see Table 1), c_s denotes the answer selected by the crowd process in a single execution s, k_s is the cost of execution s, and the function $\pi(c_i)$ returns the probability for the choice supplied in the argument (for example, when executing π with Answer 2 as an argument in C2, the result would be 10%). Hence, $-\mu$ calculates the difference between the highest result of π in an experiment condition $[\max_i(\pi(c_i))]$ and the probability of the item selected by the crowd $[\pi(c_s)]$ whilst adding the expense of the process k_s.

We ran each Auto-Experimentation algorithm (BOA, NAE1-5) 100 times for the four experimental conditions (C1-C4). The result's mean and standard deviation for both cost and utility of the 100 runs for each algorithm in condition C4 are shown in Table 2. We highlighted the relative cost of each algorithm when compared to BOA in red, emphasizing BOA's cost-efficiency. Indeed, BOA seems to be the least expensive algorithm, reliably finding well-performing crowd processes (as identified by their low average $-\mu$). NAE2-5 (where each crowd process gets executed 2-5 times, respectively) nominate slightly better processes on average; but at a much higher cost. NAE4-5 are omitted in Table 2, since they repeat NAE-3's performance on utility, whilst costing even more. The results of conditions C1-C3 follow the same trend.

		BOA	NAE-1	NAE-2	NAE-3
$-\mu$	Average	7.50	11.12	6.84	6.00
	Stdev	5.00	5.80	2.76	0.00
cost	Average	$8.18 1x	$139.68 17x	$279.34 34x	$418.63 51x
	Stdev	$1.94	$2.31	$3.37	$3.88

Table 2: BOA vs NAE[1-3]'s utility and cost in C4 of the simulation.

When inspecting the differences in utility for all experimental conditions C1-C4 with a T-Test, we find that BOA clearly outperforms NAE-1 ($p<0.001$). BOA does however get slightly, but significantly, outperformed by NAE-2 ($p<0.001$). We can, hence, partially confirm H1 in that BOA performs comparably to a NAE process, but some NAE variants slightly outperform BOA, all at a huge cost expense (more than 30x).

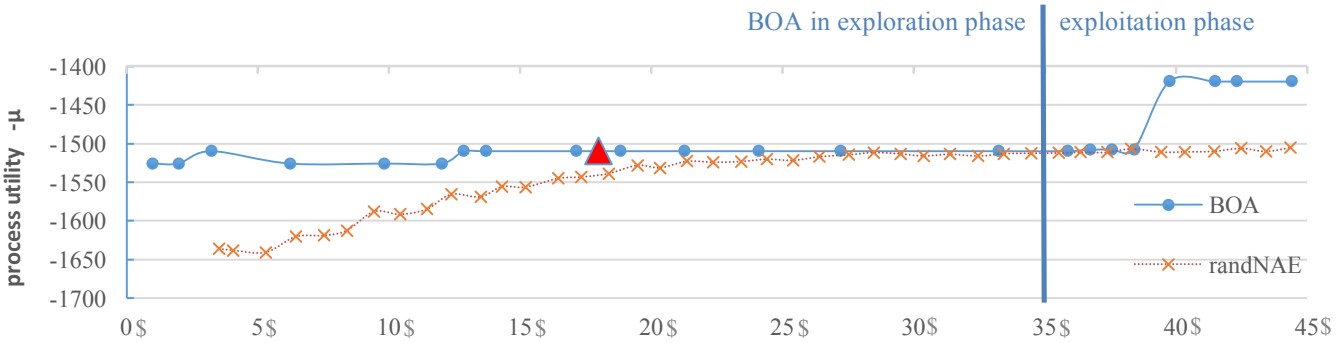

Figure 4: Average utility -μ of a strategy's nominated process (Y axis, higher is better) compared to money spent by that strategy (X-axis, in USD). Red triangle designates the first time BOA sampled its ultimately nominated process. For the same amount of money, BOA in average finds better processes than randNAE.

We also found that each experimental condition (C1-C4) yielded different optimal processes, which emphasizes the need for systematic crowd process design and confirms a similar finding using real world-data [7]. Conditions with a small difference in selection probabilities (C3, C4) are generally a lot more expensive than the conditions with high difference thereof (C1, C2). In addition to a higher cost in Auto-Experimentation, the nominated crowd processes are also more expensive. This makes sense intuitively: The more ambiguous the correct answer to a question, the lower the share of crowd workers answering correctly – therefore requiring more work to obtain the correct answer. Our simulation, hence, supports the hypothesis that BOA finds good (though not necessarily the best) crowd process inexpensively.

Experiment #2: BOA in the a real-world

Since an evaluation in a simulated environment is fundamentally limited by the assumptions taken to build the simulation, its generalizability to the real world is not always clear. To address this shortcoming, this section reports on a comparison between BOA and NAE in a real-world experiment. Given its ubiquity in crowdsourcing we chose the problem of text shortening, where crowd workers were employed to shorten a given article[2] from the popular news platform *The Verge* as much as possible without losing relevant information. The article had 1440 characters.

We set the crowd-process's deep structure to a single CREATE-type building block (to *create* a shortened version of a given paragraph; see also [7]) that operates on each paragraph of the original article. Since NAE entails running all generated crowd processes multiple times, it can become very costly. We therefore had to limit the Recombinator to vary only the following coordination mechanisms and associated parameters:

- Majority vote with 3 or 5 voters
- Beat-By-K with K=2
- Statistical Confidence Voting with 85% confidence
- Collections with 5 or 7 workers
- Sigma-pruned Collections with 5 or 7 workers

We used a trade-off between crowd process cost and the shortened text-version as utility function μ:

$$-\mu(l, c) = l + c,$$

where l is the resulting text-length and c is the crowd process execution cost in cents. Using this problem definition for text shortening, the Recombinator generated 41 different processes. For NAE-4 we could reuse experimental results acquired in [7]. We configured BOA to sample (without replacement) from these.

The process nominated by BOA was a variation of Find-Fix-Verify [6], using 3 crowd workers to propose shortened alternatives to a paragraph, and 5 crowd workers to select the best alternative in a majority vote. It was the 2nd best of all evaluated processes as measured by $-\mu$ according to NAE-4, with a small delta μ of 12.3 (mean μ was 1643.6 with a standard deviation of 219.6). The fact that the BOA-nominated process was almost the best one found by NAE-4 suggests that H1 can also be confirmed in the real-world setting. The total cost of running BOA was $44.40 as opposed to the $499.41 consumed by NAE-4, which confirms H2 in a real-world setting as well. This evaluation also gives first evidence supporting H3, since BOA nominated the same crowd process that was proposed by an expert for the problem of text shortening (see [27]), at less than 10% of the cost of NAE).

To illustrate the cost-efficiency of BOA, we modified NAE to randomly sample configurations without replacement rather than search the whole space. We call this setting randomized NAE (randNAE). We ran randNAE for budgets ranging from 1$ to BOA's convergence budget ($44.50), stopping it whenever it exceeded the budget in a given iteration (e.g., for a 1$ budget, the actual aggregate cost might be 3$, as one only knows it after running a process). This procedure was repeated 1000 times for each budget and the mean -μ and actual cost returned. As evidenced in the Figure 4, BOA's result quality (Y-Axis) is higher in average than randNAE's at a given budget. The figure shows how BOA initially samples various locations of the process search space in what we call the *exploration phase*. During that phase, BOA primarily builds its knowledge about the search space and therefore appears to perform

[2]http://www.theverge.com/2015/1/8/7517361/google-getting-ready-to-sell-auto-insurance-and-maybe-buy-coverhound

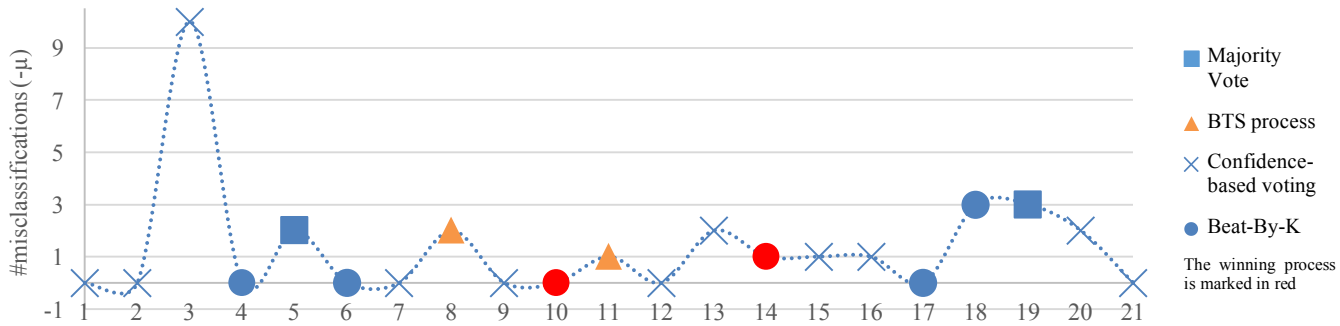

Figure 5: Results of cost functions $-\mu$ of evaluated candidate crowd processes (Y-axis, lower is better) as experienced by BOA in each step (X-axis) when looking for the best crowd process in terms of accuracy. $-\mu$ is the number of misclassifications. The shape of a data point designates the used control mechanism and captures any possible parameterization. Except for red shapes, which designate the winning variant of a control mechanism: Beat-By-K with K=9, while blue circles designate any other flavor.

similar to random sampling. In this example, after having spent $35, BOA starts rechecking promising processes executed in the exploration phase and ultimately converges onto its final nomination. During this *exploitation phase*, one can see a clear difference between randNAE and BOA. BOA has encountered the process it would ultimately nominate in the middle of its exploration phase already (see red triangle in Figure 4). At that time, BOA was not confident enough to nominate it though, since another process had returned a better result when sampled once. BOA resampled it in the exploitation phase and found it to perform well and stable.

Experiment #3/4: BOA vs NAE for an expert process

The goal of the third experiment is to test whether H3 can be confirmed for a larger experimental setup, where NAE would be prohibitively expensive. Given that the literature provides a large catalogue of intricate crowd processes proposed by (academic) experts, we decided to replicate the experimental setup of an existing paper and investigate if BOA would nominate a crowd process with a comparable performance to the one proposed in the replicated paper.

Given its rich discussion and theoretical founding, we chose Prelec's Bayesian Truth Serum (BTS), formally introduced in [29]. The author verified his model in a recent field experiment involving human subjects [30], where BTS is used to '*find truth even if the crowd is wrong*'. In this paper, we replicated the real-world study. BTS can be applied to a multiple choice question, where it essentially weights a person's choice by the accuracy of that person's estimate on how often each answer option to the question is selected by the other people polled. For example, when being asked to pick the capital city of a US state among four cities, a person's answer contains her/his own reply and an assessment of that person's belief what fraction of the population would pick each of the four choices. In case of US states, some capitals are not among the four most populous cities of a given state, which may lead to wrong answers. For example, in our experiment, Philadelphia was often mistakenly selected as capital of Pennsylvania – even though the actual answer is Harrisburg. BTS is based on the assumption that people knowing the right answer to a question may be able to better predict what others confuse the right answer with and can therefore estimate the percentage of people giving that answer more accurately.

Since our main goal is to test the Auto-Experimentation module (not the Recombination mechanism), we extended PPLib's Process Repository with a new DECIDE-type building block implementing BTS. Furthermore, we have used the same questions as in the BTS paper: Determining the capital of US states among four cities, including at least three of the four most populous cities of that state besides the actual capital. The deep structure was, therefore, a simple DECIDE-type building block, which was executed on 10 randomly sampled US states (without replacement). We have limited the Recombinator to the same parameter ranges as in Experiment #1, but also included the new BTS building block and its variations on the amount of workers used (between 1-10) as alternatives to choose from. Our main goal was to find the crowd process leading to the most accurate labels. We therefore defined the cost function $-\mu$ as the amount of misclassifications of the candidate crowd process, which is to be minimized.

The problem definition stated above lead the Recombinator to generate 212 candidate processes, including BTS. When executing BOA on the set of the 212 recombined candidate crowd processes, it was able to identify a process that performed even better in this task setting than Bayesian Truth Serum: A Beat-By-K voting pattern, with K=9, was nominated as the best crowd process (measured by process stability and result accuracy). Figure 5 shows each step of BOA in reaching this conclusion and displays the cost function values $-\mu$ of each executed crowd process candidate (Y-axis, lower is better) per step (X-axis). BOA took 21 steps to identify the winning process (by pure chance, it encountered the minimal -μ in the first step. Following our convergence criterion, the optimizer then stopped 20 iterations after being unable to produce a new minimal -μ). As the Figure indicates, it tested two variants of BTS (orange triangles; varying the number of involved crowd workers) and multiple versions of Beat-By-K (circles) eventually choosing Beat-By-K with K=9 as the winning process (red circle). Note that the run in step #3

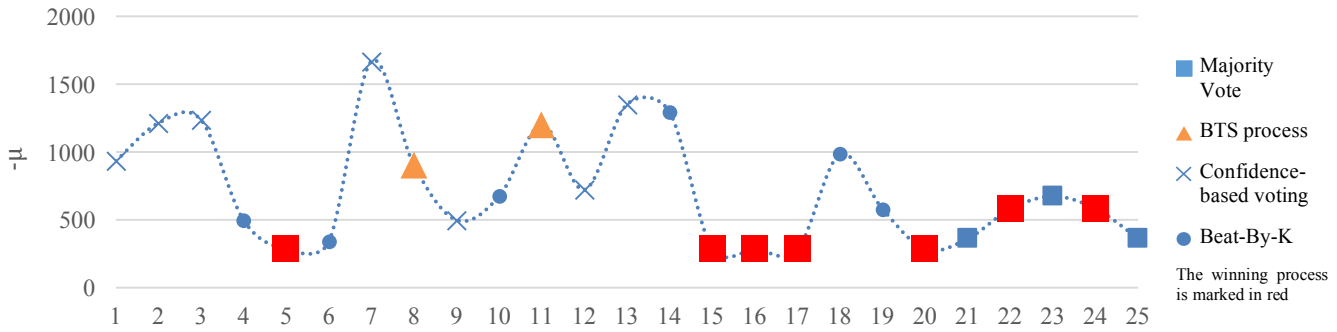

Figure 6: Values of cost functions $-\mu$ of evaluated candidate crowd processes (Y-axis, lower is better) as experienced by BOA in each step (X-axis) when optimizing for a trade-off between accuracy and cost. As in Figure 5, markers only show control structures

aborted due to an internal error. In order to verify H3, we need to determine the cost of executing NAE-4.

Given the prohibitively expensive cost of executing NAE-4 for all 212 candidates, we chose to calculate a lower bound using the easily predictable cost of Majority Vote as a baseline. Majority vote is the least expensive pattern of all coordination structures in the set, as all other processes sample the same number of people, but then add some complications (such as requiring a difference of at least K, often leading to additional votes required). NAE-4's total cost of evaluating all variations of a Majority Vote with worker counts between 1-10 is

$$s \cdot t \cdot \sum_{i=1}^{w} i \cdot k = 10 \cdot 4 \cdot \sum_{i=1}^{10} i \cdot 4 = 88\$,$$

for 10 states (s), 4 NAE iterations (t), 4 cent per question (k), 1-10 workers (w). Since Beat-By-K includes 10 different parameter values for its maximal budget, its lower cost-bound is therefore ten times Majority Vote's. BTS' lower bound is five times Majority Votes, since it asks each crowd worker 5 questions in one task. Leaving Confidence-based Voting aside (as its cost is complicated to assess ex-ante), all of these four process account for a pessimistic total lower bound cost for NAE-4 of $88\$ + 10 \cdot 88\$ + 5 \cdot 88\$ = 1408\$$. BOA's actual cost amounts to less than 10% of that: \$96.24. Combined with the fact, that BTS was designed by an expert and that the nominated process was superior in performance, this leads us to confirm H3 in this experimental setting.

Beat-By-K with K=9, seems to lead to very accurate results for finding US states capitals. It is, however, very expensive[3], since it requires the winning city to receive at least 9 more votes than the 2nd most popular city – and continues requesting more (costly) crowd votes until that condition is satisfied. Since our utility function only looked at accuracy, there was no punishment for Beat-By-K's high cost. Hence, we decided to rerun the experiment whilst incorporating crowd process cost into the utility as well:

$$-\mu(m, c) = m \cdot 300 + c$$

where m is the number of misclassifications of a given crowd process and c the amount spent on running the process in cents. Loosely, each misclassified state capital lead to a punishment of 3\$ for the Auto-Experimentation Engine on top of the process' actual execution cost. Note that since the problem definition remains the same, we executed BOA on the same 212 generated crowd processes (only with a different cost function $-\mu$).

Figure 6 again shows each step taken by BOA in for this 2nd iteration. As expected, the cost based penalization of Beat-By-K-9 lead BOA to explore the tradeoff between accuracy and cost, nominating a simple majority vote with 7 crowd workers (red squares). The tested variations of the Bayesian Truth Serum (triangles) were among the most costly crowd processes, which significantly influenced their ranking: PPLib by default prices crowd tasks automatically according to the number of questions crowd workers need to answer in a single task (and therefore, the amount of time spent by a crowd worker to answer an individual crowd task). Since Bayesian Truth Serum relies on crowd workers estimating the probability of other crowd workers answering each individual city on top of her/his own choice, a single task was priced at 18 cents. While a single multiple choice question asking crowd workers to identify the capital city is only priced at 4 cents. BOA evaluated 2 variations of BTS, both leading to rather high (i.e. unattractive) values of $-\mu$. In Figure 6, it is also interesting to observe that BOA repeatedly executes the winning process after step #15 (red squares) as it tries to assess its stability and then starts comparing it to other flavors of Majority Vote (blue squares).

In our 2nd iteration we again find BOA nominating a well-performing process below 10% of NAE's cost (\$79.24) and can therefore confirm H3 with this different utility function as well.

When comparing Figure 5 with Figure 6, another interesting observation is that the first 14 processes executed by BOA are the same in both runs. The reason is that the optimizer initially builds up its knowledge of the objective function by concurrently asking for 14 function evaluations and only then starts updating its priors for the next evaluation (at step 15). The number of concurrent

[3] Its average cost was \$6.16 for 10 state capitals

initial data requests depends on the size of the set of the recombined surface structures, whereas a larger space leads to more initial observation requests to make reasonable judgments about where to sample next. Afterwards BOA progresses sequentially as it tries to minimize the spending for Auto-Experimentation by taking as much prior knowledge into account as possible. One could parallelize BOA more by sampling multiple points according to their acquisition value in each iteration. This would result in higher overall cost of BOA but faster execution. On average NAE would not incur differences in cost/quality when executed in parallel. In our evaluation the average execution of BOA took 10.1 hours (σ=3.6).

LIMITATIONS AND FUTURE WORK

Given that BOA is a module of PPLib, it inherits some of PPLib's limitations such as the inability to explore crowd process that cannot be assembled from PPLib building blocks [7]. When focusing on BOA specifically, a few additional things need to be considered.

Most importantly, BOA is limited by the accuracy of the user's quantification of her (business-) objectives in the utility function μ. As the outcomes for the two different utility functions used in Experiment #3 clearly evidence, BOA's nomination is heavily dependent on μ. Defining the utility function accurately is therefore a paramount to successfully applying BOA in practice. We think, Judgment bootstrapping [1] poses one path to an accurate μ. Please also note, that the utility functions used in our evaluation were defined by us according to our intuition of importance for a given problem. In our experiments, we only looked at deterministic utility functions μ, always returning the same result when supplied the same arguments.

The technique of Bayesian Optimization requires the error distribution of candidate crowd processes to be (almost) normal due to using GP's — a fact that was true in our simulation, but could not be guaranteed in any real-world experiments. However, the positive outcomes of the real-world experiments shed light on BOA's robustness for violations of its assumptions. Nonetheless, the robustness of BOA should be explored in the future.

Crowdsourcing markets and their characteristics may change over time. Therefore, one might want to correct for some confounding variables, such as day-of-week, for example by rerunning BOA on different days of the week or parameterizing execution time for BOA to optimize. Changes during BOA's execution are modelled as part of the crowd processes' variance and are hence accounted for.

We evaluated our approach to crowd process selection with up to 212 candidate processes to select from and saw that the optimizer found good processes relatively quickly. However, it is possible that our approach scales less well to (many) thousands of candidates than some methods that use insights about tasks to guide process selection [12,36]. Future research should quantify the difference between these two approaches and possibly bridge the gap by introducing a hybrid between task-agnostic optimization, applicable to any crowd process, and task-based optimization, applicable only to crowd processes whose tasks performances have been modelled.

Running BOA (or any Auto-Experimentation strategy) is limited by the quality of a tasks sample, e.g., for the running example problem definition used in this paper of translating a book, a sample could be a set of paragraphs, pages or even chapters. The problem of finding representative samples is not limited to crowd process design though. Various approaches have been proposed to guide sampling, among them power analysis (with respect to sample size) [11].

There are many different ways of applying Bayesian Optimization to a problem. Further investigation is required to contrast different configurations for this problem. Instead of an optimization problem, crowd process selection could also be seen as a multi-armed bandit problem, where a gambler needs to decide on what machine to play, in which order to play them and how many times to play, whereas machines would correspond to crowd processes. Lin et al [21] found that switching between multiple active crowd processes may yield better performance than using a single crowd process.

CONCLUSION

In this paper we proposed a strategy to efficiently select crowd processes from a potentially large set of candidates independent of these candidates' inner workings. When combining this technique with generative crowdsourcing libraries such as PPLib, a method to efficiently find well-performing crowd processes for a problem at hand emerges. Our approach is based on the idea, that crowd processes can be composed of process fragments. These process fragments (and their individual configuration) can be seen as parameters to a black-box optimization problem, which we propose to solve using the Bayesian Optimization technique. We implemented our method as an extension to the PPLib system, an open source programming library to support crowd process designers. We evaluated our prototype by virtue of a simulation as well as three real-world experiments. The evaluation showed large cost-reductions when compared to the baseline algorithm for crowd process selection at a comparably high quality. Due to its black-box nature, our approach has a lower entry barrier for efficient crowd-process selection than current state-of-the-art. This allows practitioners to easily and affordably compare several candidate processes, which in turn has the potential to simplify the design of crowdsourcing solutions for many users.

ACKNOWLEDGMENTS

We would like to thank Michael Feldman and Yiftach Nagar for their feedback on our first draft of this paper. This work was supported in part by the Swiss National Science Foundation (SNSF- Project: 200021- 143411/1).

Additionally, we'd like to express our gratitude to our anonymous reviewers and everybody else involved in the reviewing process, whose constructive feedback was extremely valuable in improving our paper.

REFERENCES

1. Js Armstrong. 2001. Judgmental bootstrapping: Inferring experts' rules for forecasting. *Principles of forecasting: A Handbook for Researchers and Practitioners*: 169–192. Retrieved from http://link.springer.com/chapter/10.1007/978-0-306-47630-3_9
2. Daniel Barowy, Charlie Curtsinger, Emery Berger, and Andrew McGregor. 2012. AutoMan: A platform for integrating human-based and digital computation. *OOPSLA '12 Proceedings of the ACM international conference on Object oriented programming systems languages and applications*.
3. James Bergstra, Remi Bardenet, Yoshua Bengio, and Balazs Kegl. 2011. Algorithms for Hyper-Parameter Optimization. *Advances in Neural Information Processing Systems*: 2546–2554.
4. Abraham Bernstein, Mark Klein, and Thomas W Malone. The Process Recombinator: A Tool for Generating New Business Process Ideas. *ICIS*.
5. Abraham Bernstein, Mark Klein, and Thomas W. Malone. 2012. Programming the global brain. *Communications of the ACM 55*, 41.
6. Michael S Bernstein, Greg Little, Robert C Miller, et al. 2010. Soylent : A Word Processor with a Crowd Inside. *UIST*.
7. Patrick M. de Boer and Abraham Bernstein. 2016. PPLib: Towards the Automated Generation of Crowd Computing Programs using Process Recombination and Auto-Experimentation. *ACM Transactions on Intelligent Systems and Technology*, Special Issue: Crowd in Intelligent Systems.
8. Eric Brochu, Vlad Cora, and Nando De Freitas. 2010. *A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning*.
9. Justin Cheng and Michael S Bernstein. 2015. Flock : Hybrid Crowd-Machine Learning Classifiers. *CSCW 2015*: 600–611.
10. Noam Chomsky. 1965. Aspects of the Theory of Syntax. *MIT Press*.
11. Jacob Cohen. 1988. Statistical power analysis for the behavioral sciences. *Statistical Power Analysis for the Behavioral Sciences 2nd*, 567.
12. Peng Dai, Christopher H. Lin, Mausam, and Daniel S. Weld. 2013. POMDP-based control of workflows for crowdsourcing. *Artificial Intelligence* 202: 52–85.
13. Robyn M. Dawes. 1971. A case study of graduate admissions: Application of three principles of human decision making. *American Psychologist* 26, 2: 180–188.
14. Eyda Ertekin, Haym Hirsh, and Cynthia Rudin. 2012. Selective Sampling of Labelers for Approximating the Crowd. *2012 AAAI Fall Symposium Series*, 7–13.
15. Lorenz Fischer, Shen Gao, and Abraham Bernstein. 2015. Machines Tuning Machines: Configuring Distributed Stream Processors with Bayesian Optimization. *2015 IEEE International Conference on Cluster Computing*: 22–31.
16. Sergiu Goschin. 2014. Stochastic dilemmas: foundations and applications.
17. Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6683 LNCS: 507–523.
18. Oana Inel, Khalid Khamkham, Tatiana Cristea, and Anca Dumitrache. 2014. CrowdTruth : Machine-Human Computation Framework for Harnessing Disagreement in Gathering Annotated Data. *International Semantic Web Conference (ISWC)*.
19. Ross D King, Jem Rowland, Stephen G Oliver, et al. 2009. The automation of science. *Science (New York, N.Y.)* 324: 85–89.
20. Anand Kulkarni, Matthew Can, and Björn Hartmann. 2012. Collaboratively crowdsourcing workflows with turkomatic. *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW '12*, 1003.
21. Ch Lin and Ds Weld. 2012. Dynamically Switching between Synergistic Workflows for Crowdsourcing. *26th Conference on Artificial Intelligence, AAAI*, 132–133. Retrieved from http://www.aaai.org/ocs/index.php/WS/AAAIW12/paper/viewPaper/5334
22. Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. 2010. TurKit: Human Computation Algorithms on Mechanical Turk. *Proceedings of the 23nd annual ACM symposium on User interface software and technology - UIST '10*, 57.
23. Benjamin Livshits and Todd Mytkowicz. 2014. Saving Money While Polling with InterPoll Using

Power Analysis. *Conference on Human Computation and Crowdsourcing*, 159–170.

24. Thomas W. Malone, Kevin Crowston, and George A. Herman. 2003. *Organizing Business Knowledge: the MIT Process Handbook*.

25. Thomas W. Malone, Robert Laubacher, and Chrysanthos Dellarocas. 2010. The collective intelligence genome. *IEEE Engineering Management Review 38*, 38.

26. Jan Hendrik Metzen, Alexander Fabisch, and Jonas Hansen. 2015. Bayesian Optimization for Contextual Policy Search. 1–2.

27. Katrina Panovich Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell. 2015. Soylent: A Word Processor with a Crowd Inside. *Communications of the ACM* 58, 8: 85–94.

28. Jonas Mockus. 2012. *Bayesian approach to global optimization: theory and applications*. Kluwer Academic Publishers.

29. Drazen Prelec. 2004. A Bayesian truth serum for subjective data. *Science* 306, 5695: 462–466.

30. Drazen Prelec, H. Sebastian Seung, and John McCoy. 2014. Finding truth even if the crowd is wrong. *Working paper* MIT: 1–13.

31. Floarea Serban. 2010. Auto-experimentation of KDD workflows based on ontological planning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 313–320.

32. Jasper Snoek, Hugo Larochelle, and Rp Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. *Nips*: 1–9.

33. Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*: 847–855.

34. Vasilis Verroios and Michael S Bernstein. 2014. Context Trees: Crowdsourcing Global Understanding from Local Views. *HCOMP 2014*, Stanford InfoLab.

35. Daniel S Weld, Mausam, and Peng Dai. 2011. *Human Intelligence Needs Artificial Intelligence*.

36. Haoqi Zhang, Eric Horvitz, and D Parkes. 2013. Automated workflow synthesis. *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*: 1020–1026. Retrieved from http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/download/6457/7327

37. Haiyi Zhu, Steven P. Dow, Robert E. Kraut, and Aniket Kittur. 2014. Reviewing versus doing. *Proceedings of the 17th ACM conference on Computer supported cooperative work - CSCW '14*, 1445–1455.