Chapter 8

Automated Workflow Synthesis

In the last chapter, we introduced an approach for automating the design of human computation tasks with identical, parallel subtasks. In this chapter, we develop a general framework for automating the synthesis of human computation algorithms and workflows involving heterogeneous tasks.

There are often many ways to coordinate a crowd to solve a problem. Different human computation algorithms or workflows embody different approaches, and may utilize distinct tasks or allocate effort differently among the same tasks. Given a space of possible human computation algorithms for solving a problem, figuring out which algorithms are the most efficient requires understanding how the crowd performs on individual tasks within an algorithm and how this in turn influences the quality of the final solution and the cost of effort incurred. The goal of the designer is to discover efficient workflows that make effective use of human effort to achieve high quality solutions, and in doing so take into account crowd characteristics and any time or resource constraints the designer may face.

Since inputs from the crowd are inherently noisy and submitted answers can be incorrect, the output from any task is *probabilistic*. An algorithm may apply quality control mechanisms that use redundancy or voting to mitigate potential errors in tasks, which helps to mitigate errors in the final solution but incurs additional cost of effort. Choosing an algorithm that makes efficient use of human effort involves reasoning about which tasks to employ and how much effort to devote to each task. These decisions rely on understanding human performance on individual tasks, and on understanding how the probabilistic and possibly erroneous outputs from each task affect the final solution, either directly or through other tasks that take its output as their input.

The crowd's performance on any given task is often imprecisely known *a priori*, and the space of possible algorithms for solving a problem—involving different combinations of tasks and allocations of effort to tasks—is potentially very large. It is often costly if not infeasible for a designer to empirically compare a large number of algorithms, or to conduct a large number of experiments to learn about the crowd's performance on different tasks. In practice, experiments are often conducted on an ad hoc basis, with designers relying mostly on their intuitions and common practices to determine which algorithms to deploy. Even when deployed algorithms effectively coordinate a crowd to solve a problem, they are not necessarily efficient and may not make the best use of human effort.

To enable designers to discover more efficient algorithms and workflows with less experimentation and manual effort, we develop a general framework for *automated workflow synthesis*. Leveraging the active, indirect elicitation framework of automated environment design, we construct models of human performance on tasks for the purpose of improving a workflow. Over repeated interactions, an automated system selects experiments to refine current models, with the intent of quickly discovering an efficient workflow built on (a subset of) tasks that meets desired objectives and satisfies resource constraints.

To learn quickly, we develop a *value of information* based elicitation strategy that at any time chooses which task to experiment on based on which experiment is expected to provide information that best informs the choice of algorithm for solving a problem. This is done by comparing the expected difference in solution quality between the best algorithm generated using current information and algorithms optimized based on refined information that may be learned from experimentation. In order to reason about the effect of human task performance on the overall performance of an algorithm, we develop a simulation-based approach that uses available models to estimate the cost and solution quality associated with an algorithm. This allows us to compare workflows without having to deploy them, and is used for synthesizing the best workflow given currently available knowledge and for deciding which experiments to conduct.

We illustrate the effectiveness of our approach in a case study on *human sorting tasks*, in which human judgment is used to determine the ordering among objects being sorted. We focus on a class of quicksort algorithms in which pivot selection and pairwise comparison tasks are performed by the crowd, and consider the problem of determining how many workers to devote to each task at each level of recursion. Experimental results show that knowledge of crowd performance on tasks allows us to better optimize for algorithms that are tailored to the crowd and the designer's objective. Results also show that our elicitation strategy reveals better algorithms more quickly than selecting experiments to uniformly reduce uncertainty across models.

Section 8.1 reviews related work. Section 8.2 provides a model of the automated workflow synthesis problem. Section 8.3 introduces a general approach for automated workflow synthesis based on the active, indirect elicitation framework of automated environment design. We introduce a simulation-based approach for evaluating algorithms and present an elicitation strategy that refines current knowledge by selecting experiments to maximize the expected value of information. Section 8.4 describes the human sorting task. We introduce models for predicting human performance on pivot selection and pairwise comparison tasks, and provide a local search procedure for synthesizing sorting workflows. Section 8.5 presents experimental results. Section 8.6 discusses a number of possible extensions and directions for future work.

8.1 Related Work

A number of studies in human computation have developed optimization procedures and control strategies for enabling more efficient computation with humans and machines. For example, Shahaf and Horvitz [84] studied generalized task markets with human and machine problem solvers, and introduced formulations for optimally assigning and sequencing tasks to humans and machines to maximize the utility derived from the final solution. While we also optimize workflows by reasoning about effective combinations of tasks, we consider simultaneously the problem of learning about human performance on tasks. In addition, by utilizing simulations and local search algorithms, we are able to handle optimization problems over complex workflows in which the quantitative relationship between the crowd's performance on tasks and the quality of the final solution is difficult to capture analytically.

Drawing on techniques from decision-theoretic planning, Dai et al. [16, 17] introduced a framework for optimizing workflows by controlling at run-time the request for additional work (e.g., for the purpose of redundancy) based on costs and the inferred work quality. Recent work by Lin et al. [57] showed that a similar approach can be used to dynamically switch between workflows, which can sometimes lead to improvements over using a single workflow. In these works, the structure of the workflow or the set of workflows considered is predetermined and the goal is to efficiently control the computation given fixed designs. In contrast, our framework for automated workflow synthesis aims to tackle the complementary problem of discovering efficient designs in the first place by optimizing over the space of *possible* workflows, which determines the overall structure of the optimized algorithm and the allocation of effort within.

A number of studies have focused on enabling efficient human computation in the context of human-powered database systems that recruit a crowd to perform operations such as filters, sorts, and joins. For example, Marcus et al. [63, 62] introduced a declarative workflow engine called Qurk, and proposed optimizations for sorts and joins such as batching tasks, using numerical ratings, and pre-filtering tables before joins. Venetis et al. [94] studied human computation algorithms for retrieving the maximum item from a set, and proposed a framework for selecting algorithm parameters to optimize the tradeoff over quality, monetary cost, and execution time. By exploring the space of possible algorithms and providing performance models and optimization procedures, findings from these studies can be utilized within an automated workflow synthesis framework to help identify efficient crowd-tailored algorithms for these and related problems.

In machine computation, *program synthesis* considers the use of appropriate design tactics to systematically derive a program based on a problem specification. In the context of sorting, Darlington [18] and Smith [86] demonstrated how to derive a number of sorting algorithms using logical transformations and reductions. Closer to our work, Li et al. [55] demonstrated how to synthesize sorting algorithms that are optimized for particular computer architectures. As learning about crowd abilities incurs a cost, our work on synthesizing sorting algorithms for the crowd must tackle the added challenge of learning quickly, to synthesize efficient algorithms after few experiments.

From the machine learning perspective, our value of information based elicitation strategy can be viewed as taking an *active learning* approach to acquiring information. In the context of human sorting tasks, Pfeiffer et al. [72] introduced an algorithm that adaptively selects which pairwise comparison questions to ask a crowd in order to quickly derive an accurate aggregate ranking using the crowd's noisy answers. While in both this work and our work on synthesizing sorting algorithms the goal is to learn quickly and make efficient use of human effort, we consider through automated workflow synthesis different ways through which humans can contribute to solving a problem. In doing so, we seek to better understand how to structure efficient crowd problem solving by synthesizing algorithms involving heterogeneous tasks.

In artificial intelligence, the study of *metareasoning* [36, 80] focuses on enabling agents with bounded time and computational resources to make intelligent decisions about what to reason about, how long to deliberate for, and when to take action. Since deliberation can lead to better decisions but incurs a cost, it is often necessary to evaluate the benefit and cost of gathering information through additional computation [37, 9, 79]. Due to the cost of human effort, our automated system for workflow synthesis faces a similar problem in that it must decide on which experiments to conduct, how much resources to devote to experimentation, and when to stop experimenting. In using value of information computations to inform elicitation decisions, we adopt a decision-theoretic framework for active, indirect elicitation that draws on principles introduced by Horvitz [35, 36] for decision-theoretic metareasoning.

8.2 Automated Workflow Synthesis

We consider a situation in which an automated system seeks to identify an efficient human computation algorithm or workflow for solving a problem. Given a (potentially large) space of human computation algorithms $\mathcal{A} = \{A_1, \ldots, A_n\}$, we let S_i denote the set of base-level human tasks in A_i that can be assigned directly to individual workers in a crowd,[1] such that $S = S_1 \cup \ldots \cup S_n$ represents the entire set of human tasks under consideration. Each task $s \in S$ is associated with a *task function* f_s, which defines for each task s an output distribution on the space of possible answers, some of which may be incorrect. This captures the distribution over answers that

[1]For example, in the context of Amazon Mechanical Turk, these base-level tasks are the human intelligence tasks (HITs) assigned to workers.

individuals in the crowd may provide when assigned a task. For an algorithm A_i, we let F_i represent an *algorithm function* that maps problem instances into a distribution over solutions. The solution distribution based on F_i is itself constructed from the output distributions of tasks $s \in S_i$, which are based on f_s.

A task encompasses all the details of how the work is requested, which includes for example the user interface and instructions. Two tasks that request the same work may thus produce different distributions over answers. Furthermore, algorithms that share some or all of the same tasks may differ in the type of inputs that are passed to the tasks, and in when and how often each task is called. Algorithms that contain similar or even the same tasks may thus induce different distributions over solutions. Deciding which algorithm to use depends not only on the crowd's performance on tasks, but also on details of the algorithm that govern how outputs combine and propagate to form a final solution.

Given a distribution over problem instances and a measure of the solution quality, the system seeks to identify an algorithm $A^* \in \mathcal{A}$ that achieves a high solution quality on average while satisfying cost constraints.[2] We assume that each instance of a call to a task incurs a known cost, which may be monetary or be based on a measure of the time or effort required to complete the task. In contrast, we assume that the system does not know how well the crowd can perform each task *a priori* (that is, f_s is imprecisely known), and thus cannot perfectly predict the expected quality of solutions obtained through different algorithms.

[2]Our framework is agnostic to details of the objective. We can also consider optimizing for cost subject to constraints on quality or more complex utility-based objectives that define explicit tradeoffs between solution quality and cost.

In order to learn about the crowd's performance on tasks, the system can experiment with different tasks and observe the crowd's outputs. At any time, the system can select from a set of possible experiments $E = \{e_1, \ldots, e_m\}$, each of which corresponds to a particular task-input pair. Since the crowd's answers are probabilistic, the same experiment may result in different observations. For simplicity, we assume that all possible experiments are feasible, such that if an experiment is conducted the corresponding task will be completed by the crowd. A general goal is to quickly discover, after few experiments, an efficient algorithm that obtains high quality solutions and satisfies cost constraints. Since conducting experiments takes time and is also costly, this allows us to deploy better algorithms sooner, and also keeps the cost of experimentation low.[3]

8.3 An Active, Indirect Elicitation Approach

We introduce a general approach for automated workflow synthesis that leverages the active, indirect elicitation framework of automated environment design. For each task $s \in S$, we construct a *task performance model* $\hat{f}_s$ to predict the output from the actual task function f_s. Using observed outputs from experiments, an inference procedure updates $\hat{f}_s$ after each experiment to refine the system's knowledge of the crowd's performance on tasks. This allows the system to better predict the performance of different algorithms under consideration, based on which to optimize the choice of algorithm. In order to select experiments that lead the system to quickly

[3]The elicitation strategy we develop later in this chapter is able to consider explicit tradeoffs between the cost and value derived from experimentation. For simplicity, we do not model the cost of experimentation and focus instead on discovering efficient algorithms quickly.

discover efficient workflows, we introduce a simulation-based approach that allows us to compare different algorithms based on models, and an elicitation strategy that uses simulations to evaluate the value that can be derived from different experiments.

8.3.1 Simulating Human Computation Algorithms

At any point in the active, indirect elicitation process, we assume that the system can use the current task performance model $\hat{f}_s$ for task s to estimate a distribution over outputs for any input to f_s. Under this assumption, the system can simulate an algorithm A_i on a machine by sampling from the output distribution provided by $\hat{f}_s$, $s \in S_i$ whenever the algorithm makes a call to task s. For any algorithm applied to a problem instance, this allows the system to estimate a distribution over possible solutions. Simulations can thus be used to estimate the solution quality for any algorithm based on our current knowledge of the crowd's performance on tasks the algorithm calls upon. Furthermore, since the number of times each task is called in a run of an algorithm may in general depend on the crowd's performance on tasks, simulations also allow us to estimate, to the best of our current knowledge, the cost associated with running an algorithm.

In addition to evaluating algorithms based on current knowledge, we can also use simulations to estimate the solution quality of an algorithm under different hypotheses about f_s. This is helpful when deciding among a set of experiments to conduct.

Having the ability to simulate algorithms tackles two major obstacles for automated workflow synthesis. First, it allows us to compare algorithms without having to necessarily deploy them, which is useful when we are trying to determine which experiment to conduct next. Second, it allows us to reason about complex algorithms that may be difficult to analyze analytically, which makes this approach applicable to a broad range of settings.

8.3.2 Elicitation Strategy

An automated workflow synthesis problem may consider a large space of possible algorithms that draw on diverse tasks. Models for each task may be complex and difficult to learn accurately with few examples. Given this, we would like to be able to determine which experiment to conduct at any time, for which the knowledge acquired may significantly affect our choice of algorithm. Since our goal is ultimately to discover efficient workflows and not to learn about the crowd's performance on tasks, it is not necessary to learn about the task whose model has the most variance, or on which the fewest experiments have been conducted thus far. For example, if we have reason to believe that a task is unlikely to help an algorithm achieve high quality solutions anyway, it is unlikely that learning about this task will provide useful information for improving our choice of algorithm.

Following this intuition, we consider an elicitation strategy that selects experiments based on which task-input pair is most likely to reveal information that improves the choice of the optimal algorithm. Let $A^*_{\hat{f}}$ denote the optimal choice of algorithm to deploy based on current task performance models, such that $A^*_{\hat{f}}$ achieves the highest average solution quality across all algorithms that satisfy cost constraints when simulated using $\hat{f}_s$ for task s on problem instances drawn from a known distribution. For each experiment $e \in E$ that involves the task s_e, let $O_e = \{o^1_e, \ldots, o^k_e\}$

denote the set of potential outcomes from experiment e based on $\hat{f}_{s_e}$. Depending on the realized outcome of an experiment, we may be in one of k possible worlds, corresponding to the state of task performance models after the inference procedure performs an update based on the result of the experiment. We let $\hat{f}^{o_e^i}$ denote the updated task performance models under the assumption that we conduct experiment e and observe outcome o_e^i, and let $A^*_{\hat{f}^{o_e^i}}$ denote the optimal choice of algorithm with respect to $\hat{f}^{o_e^i}$.

Since we can potentially deploy different algorithms based on the outcome of an experiment, the difference in solution quality between $A^*_{\hat{f}}$ and each of the algorithms $A^*_{\hat{f}^{o_e^i}}$, evaluated with respect to our knowledge after observing o_e^i, captures the expected value to be gained if we were to update our choice of algorithm to deploy after conducting a single experiment e. By comparing experiments in this way, we can find the experiment that (myopically) maximizes the expected value of information by solving the following optimization problem:

$$max_{e \in E} \sum_{o_e^i \in O_e} \Pr(o_e^i | \hat{f}_{s_e})[v(A^*_{\hat{f}^{o_e^i}} | \hat{f}^{o_e^i}) - v(A^*_{\hat{f}} | \hat{f}^{o_e^i})] \tag{8.1}$$

$\Pr(o_e^i|\hat{f})$ is an estimate of the likelihood of observing outcome o_e^i when conducting experiment e based on the task performance model $\hat{f}$, and $v(A|\hat{f})$ is a measure of the expected quality of solutions provided by algorithm A based on task performance model $\hat{f}$.

An elicitation strategy based on this objective focuses experimentation on where there is the most value to be derived from learning. Since an individual experiment only obtains a single output from the crowd, it may not contain enough information to change the decision about the best algorithm. The myopic value of information may be zero for all experiments, but the choice of experiment still matters because conducting an experiment can enable subsequent experiments to become (myopically) valuable. For this reason, it can be important to conduct experiments in batch, where at any given time the elicitation strategy selects a set of experiments to conduct whose potential outcomes best inform the choice of algorithm. Outside of any computational concerns, the elicitation strategy remains essentially the same, but with each experiment representing a set of experiments.

As a technicality, in the context of constrained optimization, an algorithm optimized based on current information may be infeasible in light of information derived from observing the outcome of an experiment. In particular, the term $v(A^*_{\hat{f}}|\hat{f}^{o^i_e})$ may not be well defined. For example, an experiment may reveal that an algorithm that repeatedly calls the same task until multiple solutions agree incurs higher costs than expected if observed outputs are more varied than expected. In these situations, such an algorithm may, by nature of being infeasible, achieve a higher solution quality than an algorithm optimized based on newly derived information. To avoid uninformative comparisons to an infeasible algorithm when making value of information computations, we can apply a "primal heuristic" that transforms an infeasible algorithm into a similar, feasible algorithm. We can then perform any comparisons using the transformed algorithm instead, with the view that the difference in performance between an algorithm optimized based on new information and this transformed algorithm captures the value of information that can be derived from experimentation. Later in the chapter, we construct a primal heuristic for use in the sorting setting we consider.

8.4 Human Sorting Tasks

Having presented a general approach for automated workflow synthesis, we consider as a case study the problem of finding efficient human computation algorithms for *human sorting tasks.* In a human sorting task, human perception and judgment are used to determine the ordering among objects. Examples of human sorting tasks include sorting images by their visual appeal, sorting traffic photos by the severity of traffic conditions presented, sorting edited versions of a paragraph by how well written they are, and sorting web pages by their relevance to a query. Human sorting tasks may vary in their level of objectiveness, but share the common feature that machines often cannot accurately determine the desired ordering among objects.

There are many possible ways to sort, and designing computer algorithms for sorting is of course a well-studied problem. While it is sometimes straightforward to adapt a sorting algorithm for a human sorting task, the effectiveness of the resulting human computation algorithm will depend on how well the crowd can perform the human tasks that the algorithm calls upon. Since people can make mistakes even for objective tasks, solutions may not be perfectly sorted, and redundancy may be needed to achieve good solutions. The algorithm design space thus includes not only different types of sorting algorithms, but also different allocations of effort to tasks within algorithms. Given a constraint on the total cost of effort that can be incurred, the goal is to synthesize a human computation algorithm that maximizes the expected solution quality for an objective of interest.

We focus on the problem of automatically synthesizing a workflow from a class of human computation algorithms based on quicksort, that leverages the crowd to

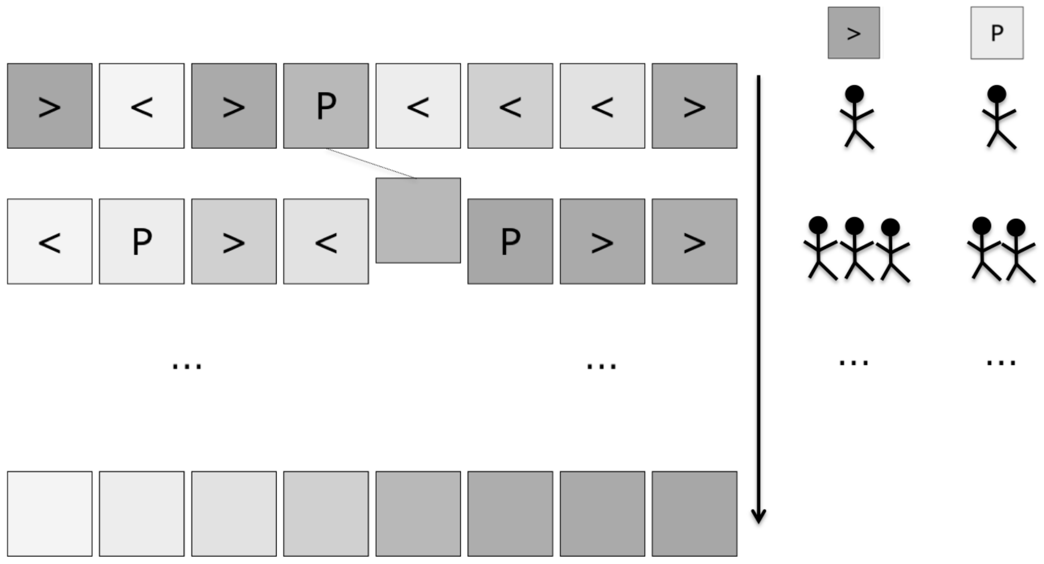


Figure 8.1: The progression of sorting with human quicksort applied to ordering grayscale tiles from light to dark. The algorithm determines the amount of human effort to allocate to each pairwise comparison and pivot selection task at different levels of recursion.

perform pairwise comparison and pivot selection operations. Quicksort is a divide-and-conquer sorting algorithm that sorts a list of elements by first identifying groups of elements that are less than or greater than a *pivot* element, and then recursively applying quicksort on each group. The choice of the pivot affects the algorithm's running time, and for example can be chosen based on the median of three elements selected randomly from the list.

In adapting quicksort for human sorting tasks, we consider how much redundancy to require for each pairwise comparison and pivot selection task that is assigned to the crowd at different points in the computation (see Figure 8.1). These decisions affect the quality of the solution, as well as the number of operations and thus cost required to compute a solution. For example, allocating more effort to pairwise comparisons early in the computation helps to place elements in roughly the correct order, whereas allocating more effort later in the computation increases the likelihood that adjacent elements are in the correct order.

Specifically, we consider optimizing over two sets of parameters r_d and k_d, that determine the number of people to recruit for identifying the median of three randomly chosen elements as the pivot (r_d), and the number of people to recruit for comparing a pair of objects (k_d), at the d-th level of recursion. In cases where $r_d = 0$, a random element is chosen as the pivot. For any task, the algorithm takes the majority answer from people recruited to perform the task as output, breaking ties randomly as needed. Algorithm 8.1 presents the pseudocode for the class of human quicksort algorithms as a function of r_d and k_d, in which MedianOfThree() and PairwiseCompare() represent the pivot selection and pairwise comparison tasks respectively.

The performance of an algorithm in this class depends on how well the crowd can identify the median and perform pairwise comparisons, and on the implications of the crowd's performance on the quality of the solution and the cost incurred. We assume that each call to a pairwise comparison or pivot selection task incurs known costs c_c and c_p respectively, which are additive and independent of the input to a task. To evaluate solution quality, we consider *inversions* as a measure of sortedness. Given a list $\{l_1, \ldots, l_n\}$ that should be sorted in ascending order, the number of inversions is the number of pairwise elements that are out of order, which occurs whenever $l_j < l_i$ for $j > i$. The goal is to find parameter values r_d, k_d such that Human Quicksort based on these values produces solutions with few inversions on average, while staying within a cost budget C.

Algorithm 8.1 Human Quicksort

Require: $\{r_d\}$, $\{k_d\}$

```
1: procedure HumanQuicksort($\{l_1, \ldots, l_n\}$, $t$)
2:     if $n = 1$ then
3:         return $\{l_1\}$
4:     else if $n = 2$ then
5:         if PairwiseCompare($l_1, l_2, k_t$) then
6:             return $\{l_1, l_2\}$
7:         else
8:             return $\{l_2, l_1\}$
9:     else
10:        $L = \{\}, R = \{\}$
11:        $p \leftarrow$ MedianOfThree($l, r_t$)
12:        for $i = 1 \rightarrow n$ do
13:            if PairwiseCompare($l_i, p, k_t$) then
14:                Add $l_i$ to $L$
15:            else
16:                Add $l_i$ to $R$
17:        return HumanQuicksort($L, t + 1$) $\cdot \{p\} \cdot$ HumanQuicksort($R, t + 1$)
```

8.4.1 Task Performance Models

In order to discover efficient algorithms quickly, we apply our framework for automated workflow synthesis to sorting by first constructing models for the pairwise comparison and pivot selection tasks. Given two distinct objects a and b, a pairwise comparison task outputs the correct answer with some probability p, and the incorrect answer with probability $1 - p$. The probability of error may depend on aspects of the task such as its user interface and instructions, on how “close” a and b are, etc. Given three elements a, b, and c, the median-of-three pivot selection task outputs the median element with probability q, the smallest element with probability p, and the largest element with probability $1 - p - q$. Similarly, the probability of error may depend on aspects of the task, the relative closeness of a, b, and c, and so on.

While we do not have access to the actual task functions and thus do not know these probabilities *a priori*, we can construct a probabilistic task performance model as follows. Since it is infeasible to learn probabilities for every combination of input values separately, we consider grouping sets of input values into *clusters*, and learning a model for each task-cluster pair. In the simplest instantiation, there may only be a single cluster per task, and the model may only attempt to learn an input-independent probability distribution over outputs. We can consider arbitrarily more complex models by considering finer-grained clusters.

For each model, we use Beta and Dirichlet distributions to represent our knowledge and uncertainty over the actual output distributions for pairwise comparison and pivot selection tasks, respectively. Distributions can capture any prior knowledge we may have about human performance on each task and be updated based on observations from experiments.[4] With Beta and Dirichlet distributions, we can incorporate observations from experiments by simply updating the corresponding model's parameters based on a worker's output. For example, for pairwise comparisons, a

[4]For simplicity, we treat each model as independent, and only perform updates on a model whose cluster matches the inputs to the task in an experiment.

Beta distribution's α and β parameters can capture the number of correct and incorrect answers respectively, and be updated by incrementing α by 1 if the answer to an experiment is correct or incrementing β by 1 otherwise. For pivot selections, a Dirichlet distribution with three parameters maintains counts over the frequency of the correct output and the two possible incorrect outputs (for each cluster), and can be similarly updated.

8.4.2 Simulating Algorithms

Each task performance model maintains a distribution over the actual output distribution for the task. As we conduct more experiments, a model becomes more certain about the crowd's performance on the task and thus allows us to more accurately predict the performance of algorithms. To measure the performance of an algorithm using current models, we can sample using the current task performance models probability distributions over the possible outputs to each task. Each sample represents a "guess" of the probability distribution over outputs based on the actual task function. For each sample, we can simulate the algorithm using the sample as the task function, and obtain a distribution over possible solutions. By aggregating results across samples, we can obtain a "best guess" over the distribution of possible solutions based on current knowledge of crowd performance on tasks as captured by our models.

In order to compute the value of information that can be derived from selecting an experiment, our elicitation strategy needs to be able to simulate algorithms with respect to hypothetically refined models that incorporate updates based on outcomes that may be observed from an experiment. To do so, we can sample output distributions using hypothetically refined task performance models whose Beta and Dirichlet parameters have been updated to take into account possible observed outcomes, but otherwise simulate an algorithm as we would when using current models.

8.4.3 Optimizing Quicksort Algorithms

Our elicitation strategy evaluates the expected value of information that can be derived from conducting an experiment by computing the difference in expected solution quality between (a) the optimal algorithm with respect to current models and (b) the optimal algorithms with respect to information derived from the experiment. For the class of quicksort algorithms we consider, the number of possible algorithms is exponential in the assignment of effort to tasks at different levels of recursion. Computing the optimal algorithm exactly is thus likely to be intractable. To avoid potential computational difficulties, we take a heuristic approach and focus on finding and comparing algorithms that are approximately optimal with respect to task performance models. We do this by adapting for our setting the local search procedures introduced by Venetis et al. [94] for optimizing human computation algorithms for finding the maximum element in a set.

To perform our search, we assume that there is a fixed, finite set of possible values to assign to parameters r_d and k_d. Given task performance models, we first compute the optimal *constant sequence* algorithm, which selects fixed values for r^* and d^* such that $r_d = r^*$ and $k_d = k^*$ for all recursion levels d. Since the space of such algorithms is small, we can obtain the algorithm that maximizes solution quality while satisfying the cost constraint by simply simulating and evaluating every possible constant sequence algorithm within the class.

The optimal constant sequence algorithm serves as a starting point for our search. Better algorithms may exist that allocate effort non-uniformly at different levels of recursion. Fixing the number of people to assign to pivot selection tasks (r_d), we consider a hill-climbing procedure that iteratively varies the number of people to assign to pairwise comparison tasks (k_d). For every pair of parameter values k_i and k_j for which $k_i > 1$, we consider the effect of decrementing k_i and incrementing k_j up to the point that the resulting algorithm *just* satisfies cost constraints. If any such swaps improve the solution quality, we apply the best such swap, and repeat the process to incrementally improve the choice of algorithm until no such improvements exist.[5]

8.4.4 Applying the Elicitation Strategy

While this local search procedure may not necessarily find the optimal algorithm with respect to a set of task performance models, we can nevertheless use the algorithms it produces to evaluate the value of information that can be gained from an experiment. Since individual experiments may not contain enough information to affect the choice of algorithm, we only consider batch experiments which obtain multiple observations at once. Assuming that the set of experiments to consider is

[5]We can construct a generalized local search procedure that considers all possible constant values $r_d = r$. We can also allow for varying values of r_d by fixing the values for r_d and k_d one level of recursion at a time. To do this, given fixed values $k_1, \ldots, k_{i-1}$ and $r_1, \ldots, r_{i-1}$, we identify the values k_i and r_i based on the solution of the generalized local search procedure applied to searching over values of r_d and k_d that have yet to be fixed. See Venetis et al. [94].

not too large, we can compute the expected value of information for each experiment and select the experiment with the largest expected value.

As discussed at the end of Section 8.3.2, one problem we may encounter in making value of information computations is that a human quicksort algorithm optimized based on current information may be infeasible in light of information derived from observing the outcome of an experiment. In particular, the term $v(A^*_{\hat{f}}|\hat{f}^{o^i_e})$ may not be well defined. When this occurs, an algorithm optimized based on current information may appear to be better (by nature of being infeasible) than an algorithm optimized based on an experiment's outcome. To avoid uninformative comparisons to an infeasible algorithm and to evaluate the value of an experiment even in such situations, we apply a "primal heuristic" that makes an infeasible human quicksort algorithm feasible by reducing the number of people it assigns to some of the tasks. We do this by iteratively decrementing some pairwise comparison parameter k_d until the algorithm becomes feasible. At each step, we select a parameter to decrement that leads to the largest (myopic) decrease in cost incurred per unit decrease in solution quality. This procedure seeks to identify a version of the original algorithm that has similar performance but does not violate cost constraints when evaluated based on hypothetically refined performance models. In this way, the difference in solution quality between the optimal algorithm given refined information and the transformed algorithm still represents the value gained when reoptimizing the choice of algorithm based on new information derived from an experiment.

8.5 Experiments

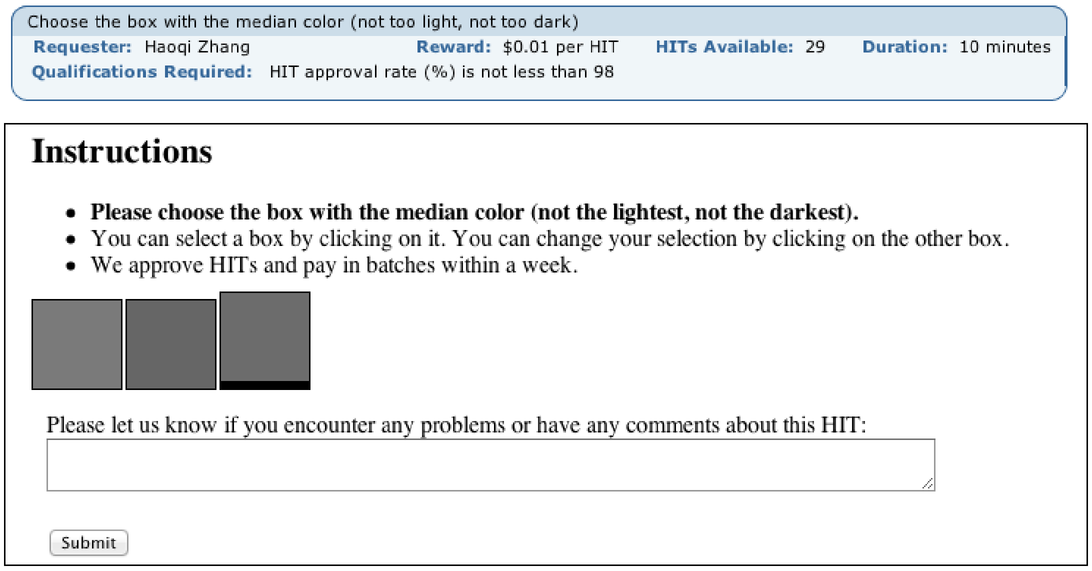
To test the effectiveness of our approach on human sorting tasks, we consider experiments for learning task performance models and optimizing human quicksort algorithms. We focus on two main questions: (a) does learning task performance models help to discover more efficient algorithms, and (b) does our value of information based elicitation strategy lead to more efficient algorithms more quickly than a simple elicitation strategy?

8.5.1 Setup

We consider a human sorting task in which the goal is to sort a list of grayscale tiles from light to dark. We chose this domain because comparisons are objective, tasks are easy to describe, and tasks may vary in difficulty (e.g., depending on how close tiles are in their grayscale value). This makes it easier for us to evaluate answers, increases the likelihood that workers understand the goal of the task, and allows for interesting models that depend on characteristics of particular task instances.

To understand human performance on this task, we recruited workers from Amazon Mechanical Turk (Turkers) to complete pairwise comparison and median-of-three pivot selection tasks. For pairwise comparison tasks, we posted 100 HITs and requested 10 assignments for each HIT. We sampled pairs of grayscale values for tiles at random, restricting the difference in value to between 1 and 10.[6] For pivot selection tasks, we also posted 100 HITs each with 10 assignments. We sampled three

[6]We used a scale with 128 values, such that black is 0 and white is 127. We chose this scale over a 256 valued scale so that minimal differences in darkness are barely distinguishable.


Choose the box with the median color (not too light, not too dark)
Requester: Haoqi Zhang **Reward:** $0.01 per HIT **HITs Available:** 29 **Duration:** 10 minutes
Qualifications Required: HIT approval rate (%) is not less than 98

Instructions

- **Please choose the box with the median color (not the lightest, not the darkest).**
- You can select a box by clicking on it. You can change your selection by clicking on the other box.
- We approve HITs and pay in batches within a week.

Please let us know if you encounter any problems or have any comments about this HIT:

Figure 8.2: An example HIT of the pivot selection task.

grayscale values for tiles at random, restricting the difference in value between the median element and the other 2 elements to between 1 and 10. Workers were required to have a 98% approval rating, and were paid $0.01 per HIT. Figure 8.2 shows an example HIT of the pivot selection task.

To simplify our evaluation, we use the Turkers' responses to construct *ground truth models* of task functions that provide distributions over answers to tasks based on the empirically observed answers from the crowd. When evaluating the active, indirect elicitation approach, instead of actually posting jobs on Mechanical Turk for experiments an elicitation strategy chooses, we instead sample from the ground truth distribution to simulate the answers the crowd would provide in an experiment. Assuming that models are accurate, results of the simulation experiments would still be indicative of the crowd's actual performance, but with the evidence obtained a priori to allow for a simpler evaluation.

For both the ground truth models and the task performance models, we cluster inputs to tasks based on the closeness of the objects being compared, according to our hypothesis that tiles are more difficult to compare when their grayscale values are closer. For pairwise comparison tasks, we consider clusters that correspond to different distances in grayscale value between pairs of objects. For pivot selection tasks, we consider clusters based on the *minimum* distance between the grayscale value of the median element and any non-median element. For both tasks, we consider five clusters each, for distances of 1, 2, 3, 4, and 5+. The models for the pivot selection task maintain counts or probabilities for three possible outcomes: (1) the median is selected, (2) the element closer to the median is selected, and (3) the element farther from the median is selected.[7] We hypothesize that if some of the elements being compared are very close together, people are more likely to make mistakes in favor of the element closer to the median than the element farther from it.

In the active, indirect elicitation process, we maintain a model for each task-cluster pair, which also forms the set of experiments that we can conduct at any given time.[8] We batch experiments to sets of five observations each, such that any update to a model is based on five outcomes drawn from the ground truth distribution for the task-cluster pair. To evaluate the value of information based elicitation strategy, we compare it to a *uniform* strategy that chooses the next experiment based on whichever model has been experimented on the fewest times thus far. We hypothesize that

[7]Whenever two non-median elements are equidistant to the median, a model for the pivot selection task chooses between them with equal probability whenever the median is not chosen.

[8]Pairwise comparison models are initialized with $\alpha = 4$ and $\beta = 1$. Pivot selection models are initialized with $\alpha_1 = 6$, $\alpha_2 = 1$, and $\alpha_3 = 1$.

	Pairwise		Pivot		
Difference	Pr(correct)	Pr(incorrect)	Pr(median)	Pr(closer)	Pr(farther)
1	0.74	0.26	0.585	0.27	0.145
2	0.87	0.13	0.69	0.16	0.15
3	0.94	0.06	0.74	0.13	0.13
4	0.98	0.02	0.82	0.10	0.08
≥ 5	0.997	0.003	0.85	0.08	0.07

Table 8.1: Ground truth models based on Turkers' performance on pairwise comparison and pivot selection tasks as a function of the (minimum) difference in grayscale value between tiles.

(regardless of elicitation strategy) learning will lead to better algorithms, but that the value of information elicitation strategy will lead to better algorithms after fewer experiments.

When synthesizing human quicksort algorithms, we consider optimizing with respect to random permutations of a list with 20 tiles holding grayscale values 1 through 20, with costs $c_c = c_p = 1$ and budget $C = 250$.[9] We consider $k_d \in \{1, 3, 5, 7\}$ and $r_d = r \in \{0, 1, 3\}$ as the possible values to assign to parameters k_d and r_d, where $d \in \{1, 2, 3, 4, 5, 6+\}$.

8.5.2 Results

From the Mechanical Turk experiment, we found that people indeed make more mistakes in pairwise comparison tasks when tiles are closer in grayscale value. We observe from Table 8.1 that when tiles only differ in value by 1, the crowd makes twice as many mistakes (26% error rate) as when tiles differ in value by 2 (13%), and

[9]Note that since our models only consider the difference in grayscale value between tiles, the exact grayscale values we assign to tiles are inconsequential for the purposes of our experiments.

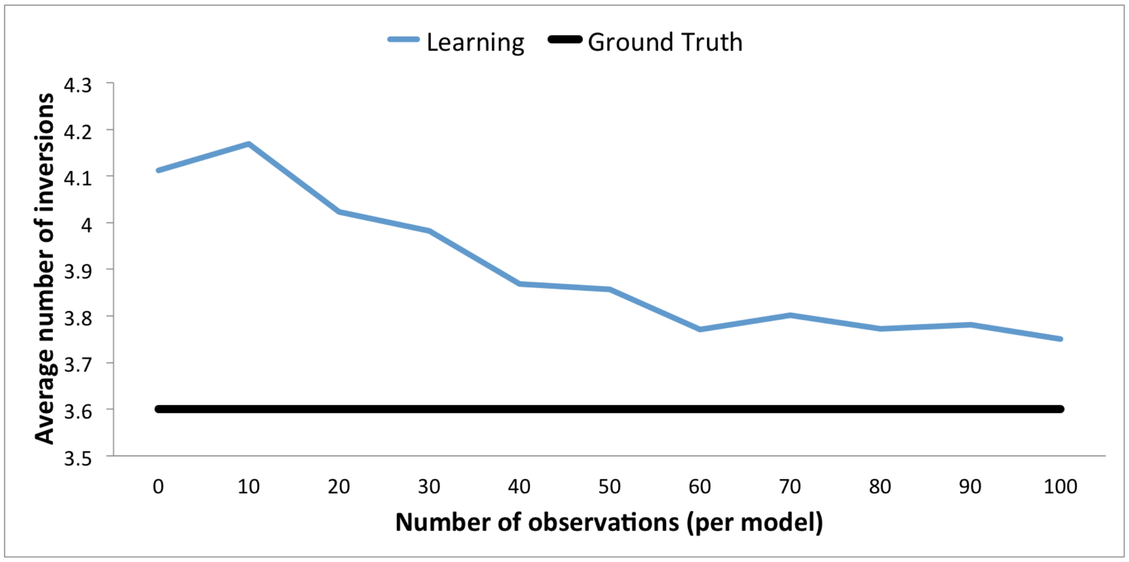


Figure 8.3: The performance (with respect to ground truth models) of algorithms optimized through the process of learning. Values represent averages over 50 trials.

about four times as many mistakes as when tiles differ in value by 3 (6%). The crowd makes very few mistakes for any larger differences in value, which suggests that after a certain point the tiles are noticeably different. For pivot selection tasks, we also found that people make more mistakes when one or more of the non-median elements is close to the median. We observe that when a non-median element is very close to the median (i.e., differ in grayscale value by 1), people are much more likely to make mistakes in favor of selecting that element than the farther non-median element.

Based on workers' answers, we constructed ground truth models using the empirically observed probabilities for each task-cluster pair (Table 8.1). Figure 8.3 shows that the average performance of the algorithm optimized using current task performance models (evaluated with respect to the ground truth models) improves over time as we conduct more experiments in simulation and observe more samples drawn from

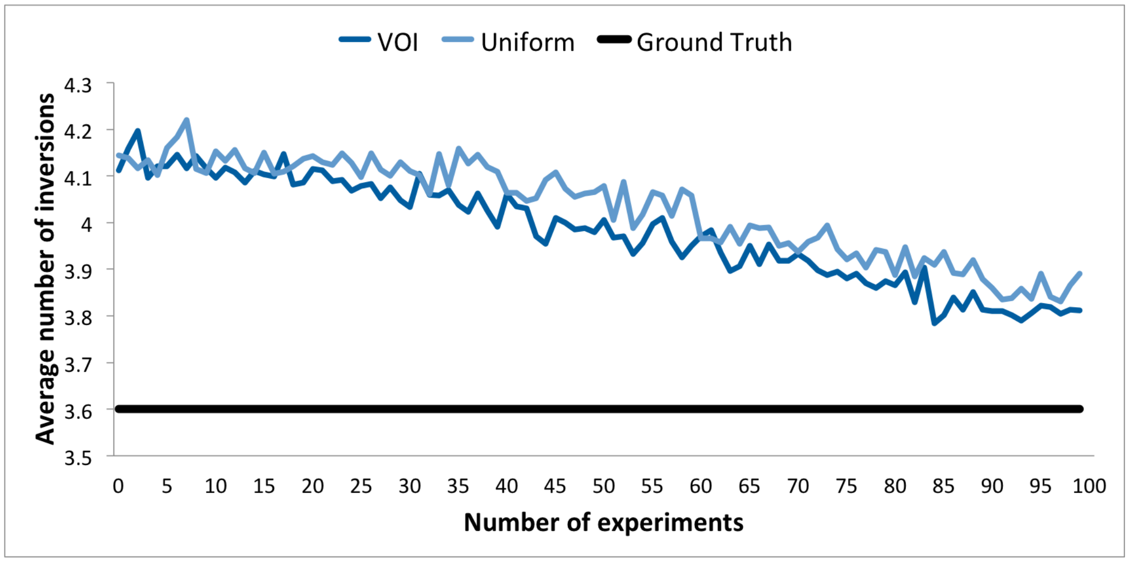


Figure 8.4: Comparison of the performance (with respect to ground truth models) of algorithms optimized based on current models through the process of learning for the value of information (VOI) and uniform elicitation strategies. Values represent averages over 50 trials.

the ground truth distribution.[10] This demonstrates that knowledge acquired from experiments reduces noise and uncertainty in task performance models and informs better decisions when synthesizing workflows based on learned models.

Figure 8.4 compares the average performance of algorithms optimized using current models over the course of learning based on the value of information and uniform elicitation strategies. We observe that for both strategies, solution quality generally improves as more information is collected from experiments. Comparing the two strategies, we observe that at any given point in time, algorithms optimized based on information obtained using the value of information elicitation strategy tend to

[10]As with making value of information computations, we may encounter scenarios in which an optimized algorithm using current task performance models does not satisfy cost constraints with respect to the ground truth distribution. In these cases we apply the primal heuristic earlier discussed and evaluate the performance of the feasible, transformed algorithm instead.

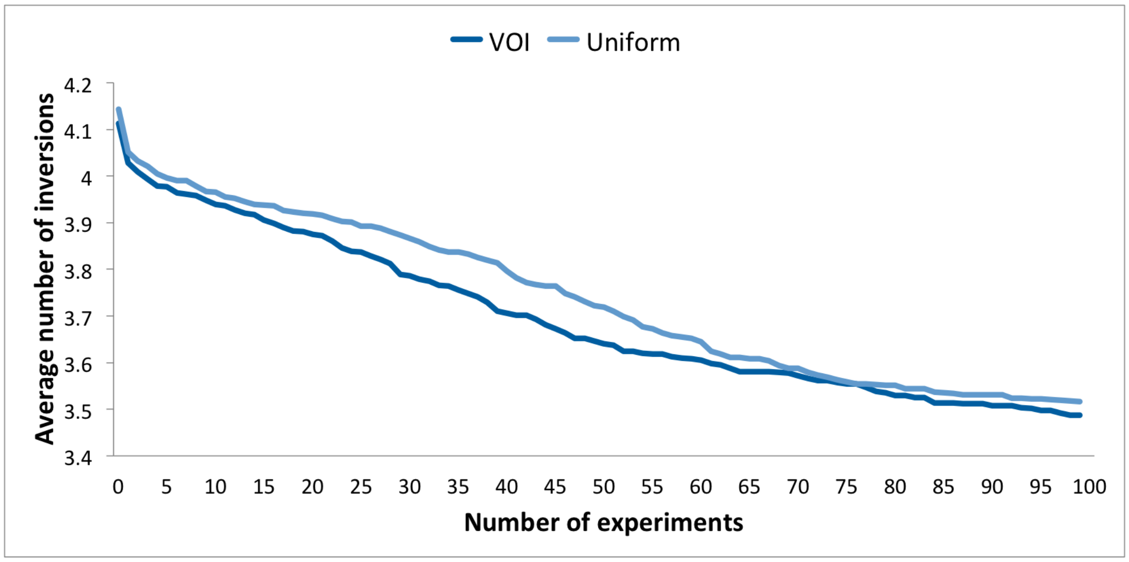


Figure 8.5: Comparison of the performance (with respect to ground truth models) of the best algorithms discovered thus far through the process of learning for the value of information (VOI) and uniform elicitation strategies. Values represent averages over 50 trials.

outperform algorithms optimized based on information obtained using the uniform elicitation strategy (90% of the time). That is, given the same amount of experimentation, the value of information elicitation strategy allows the system to synthesize better algorithms on average than the system can synthesize based on information derived from following the uniform strategy. Viewed differently, for any desired solution quality, the value of information elicitation strategy allows the system to optimize for algorithms achieving that solution quality after fewer experiments.

Since learned task performance models are inherently probabilistic and noisy, there is no guarantee that a piece of evidence obtained from experimentation will necessarily lead to an optimized algorithm with strictly better performance. An algorithm optimized based on current models thus serves as a best guess of what may be a good

	$k_1, k_2, k_3, k_4, k_5, k_{6+}$	r	Inversions	Cost
Best discovered through learning	$3, 3, 5, 3, 7, 5$	1	3.45	249
Synthesized based on ground truth	$3, 3, 3, 5, 7, 5$	3	3.60	248

Table 8.2: Configuration and performance (with respect to ground truth models) of the best human quicksort algorithm discovered through our learning experiments and the human quicksort algorithm synthesized with respect to the ground truth models. We consider random permutations of a list with 20 elements holding values 1 through 20, with costs $c_c = c_p = 1$ and budget $C = 250$.

algorithm. From the designer's perspective, algorithms synthesized at any point in time can be viewed as candidates for A/B testing against the best algorithm discovered thus far that is (presumably) currently deployed. Taking this view, we also compared the value of information elicitation strategy against the uniform elicitation strategy based on the solution quality of the best algorithm discovered thus far. Figure 8.5 shows that on average, the value of information elicitation strategy discovers efficient algorithms more quickly, and at any point in time, has already discovered a more efficient algorithm than has been discovered by the uniform strategy.

Table 8.2 shows the configuration and performance (with respect to ground truth models) of the best human quicksort algorithm discovered through our learning experiments and the human quicksort algorithm synthesized with respect to the ground truth models. We see that both algorithms apply more effort at deeper levels of recursion than at shallow levels (1 and 2). In quicksort, at deeper levels of recursion, any two tiles being compared are more likely to be closer in grayscale value. Since workers are more likely to make mistakes when tiles are close in grayscale value, the additional effort being applied at deeper levels of recursion reduces the likelihood of such errors and thus effectively reduces the number of inversions.

We also observe from Table 8.2 that it is possible for an algorithm synthesized based on learned models to outperform an algorithm synthesized based on ground truth models when evaluated with respect to the ground truth models. This counter-intuitive situation may occur because the local search procedure we use to synthesize algorithms may constrain the search space differently depending on the models considered. In particular, since the local search procedure fixes the number of repetitions (r) used for pivot selection tasks based on the best constant sequence algorithm, depending on the models considered, some values for r are not explored. While the search space with respect to the ground truth models can only consider human quicksort algorithms for which $r = 3$, the search space with respect to learned models may consider different values of r and thus include better algorithms.

In comparing the best algorithm discovered through our learning experiments with the algorithm synthesized based on ground truth models, we observe that with $r = 1$, the best algorithm discovered allocates more repetitions to k_3 and fewer repetitions to k_4, which is infeasible for $r = 3$. This helps to reduce the number of inversions because the nominal number of calls to pairwise comparison tasks (not counting how many repetitions are requested for each task) for which the tiles' grayscale values are 1 apart is highest at level 3. This is due to the recursive structure of quicksort. Since there are roughly twice as many lists at level 4 than at level 3, there are roughly twice as many pivots selected at level 4. Since tiles being compared against the pivot are those that have yet to be selected as a pivot, the number of pairwise comparison tasks decreases rapidly as we move to deeper levels of recursion. While the fraction of pairwise comparison tasks for which grayscale values are 1 apart is higher at level

4, at level 3 there is significantly more pairwise comparison tasks. The effort shifted from repetitions assigned to pivot selection tasks and to k_4 is thus put to good use through k_3 to reduce the potential for error in more pairwise comparisons tasks for which the likelihood of error is highest.

8.6 Discussion

We applied the active, indirect elicitation framework of automated environment design to the problem of automated workflow synthesis and demonstrated how learning about human performance on tasks and synthesizing workflows based on learned models can enable more efficient human computation. To discover more efficient algorithms more quickly, we introduced an elicitation strategy that reasons about the value of information that can be derived from conducting different experiments and focuses the learning on where this value is greatest. Results from experiments on human sorting tasks showed that the elicitation strategy is effective for quickly discovering efficient algorithms that are tailored to the crowd's performance on tasks.

Our framework and methods are quite general, and can be extended in a number of ways. In the context of sorting, we can for example consider a larger set of possible tasks beyond pairwise comparison and pivot selection, and include in the design space other classes of sorting algorithms beyond quicksort. As some tasks may be used in multiple algorithms, any knowledge of the crowd's performance on such a task will inform the design of all algorithms that use it. In addition to deciding how to allocate effort within each class of algorithms, we can also consider optimizing over *hybrid sort algorithms*. For example, we can consider using one algorithm to first order items roughly and another algorithm to then refine the sort, which may be more efficient in some settings [62].

We saw from observing Turkers' performance when comparing grayscale tiles that task performance can depend on not only the task, but on specifics of the problem instance. In general, for accurately predicting the crowd's performance in order to effectively synthesize algorithms, models of task performance may need to be quite rich. This suggests that a model may require significant effort and domain knowledge to construct and a significant amount of data from experiments to learn. Given that the same tasks may be used in not only different algorithms for solving a particular problem, but also in different contexts for solving completely different problems, we would like to be able to reuse designer and crowd effort by building extendable libraries of task performance models that can be reused in other automated workflow synthesis problems. Such libraries would allow for a "warm start," where one can begin reasoning about algorithms using already learned information about some tasks and focus learning efforts on other tasks. Learned models can similarly be incorporated into libraries for future reuse.

While we have focused primarily on the learning problem, considering a more complex design space brings into focus computational challenges in optimizing and synthesizing workflows. Since synthesizing an algorithm may involve choosing tasks and allocating effort to tasks, both of which are combinatorial in general, the problem can be arbitrarily hard computationally. Having tractable procedures that can effectively search over the design space and discover efficient algorithms quickly is crucial, both for the purpose of quickly deploying designs based on learned information and for making decisions based on elicitation strategies that synthesize workflows under different knowledge conditions as subroutines in value of information calculations. As in active, indirect elicitation approaches more generally, synthesis procedures can use current models of participant behavior to help constrain the search for a well-tailored design.

Human computation algorithms may include tasks for machine computation. The performance and efficiency of such tasks can be similarly measured, modeled, and reasoned about when synthesizing workflows. Our framework extends straightforwardly to include machine tasks, and allows for learning and optimizing over human-machine algorithm design spaces. The decision-making over whether to use human or machine computation components may consider particular tradeoffs in efficiency, cost, and performance [84].

As mentioned in our discussion of related work, we make a conceptual distinction between automated workflow synthesis and decision-theoretic control [16, 17]. Workflow synthesis is about *algorithm design*, and focuses on reasoning about the structure of an algorithm before it is deployed. Decision theoretic control is about *execution control*, and focuses on reasoning about the state and progression of problem solving in the midst of solving a problem. As design and control both influence eventual performance and complement one another, future work should explore considering both aspects in unison, which may lead to discovering new techniques and approaches.