AskSheet: Efficient Human Computation for Decision Making with Spreadsheets

Alexander J. Quinn[1,2], Benjamin B. Bederson[1,2,3]
University of Maryland, College Park
Human-Computer Interaction Lab[1] :: Computer Science[2] :: Institute for Advanced Computer Studies[3]
College Park, Maryland 20740
{aq, bederson}@cs.umd.edu

ABSTRACT
The wealth of resources online has empowered individuals and businesses with an unprecedented volume of information to aid in decision making processes. However, finding the many details needed for a non-trivial decision can be very labor-intensive. We present AskSheet, a general system that leverages human computation to acquire the inputs to an arbitrary decision spreadsheet provided by the user. The key innovation is the ability to prioritize the inputs by analyzing the user's spreadsheet formulas to calculate value of information for each of the blanks. By directing workers to find the details that impact the end result most, it results in a conclusive decision without gathering all of the inputs.

Author keywords
Human computation; crowdsourcing; value of information

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
The explosive growth in availability of information online has opened a bounty of resources for people making plans and strategic decisions. They can now turn to web pages and online databases for questions requiring objective information. Subjective questions can often be answered by a post on a social network or discussion forum, or a simple email or chat message to a trusted colleague.

The challenge comes when trying to leverage these mediums for more complex problems that require more thorough analysis. While it may be easy to find an answer to a specific question, a solution to the overarching problem may be more elusive. Consider the following questions:

Where should I buy groceries this week to save money?
Where/when can I vacation to balance cost vs. preference?
What graduate programs should I apply to?

CSCW'14, February 15 - 19 2014, Baltimore, MD, USA

http://dx.doi.org/10.1145/2531602.2531728

Each person will base their decision on their own unique set of factors. Of course, there can never be a database with up-to-date details on *every* possible aspect of such decisions. The details may be locale-specific or time-sensitive.

When the outcome is worth some effort, a tactic used by many decision-makers is to create a spreadsheet with tables containing the relevant dimensions of the decision problem. If the user is fluent with the use of spreadsheet formulas—as many in the business world are—and already has a clear idea *how* she wants to decide (i.e., if she had all information available), then she can even add formulas to display the decision result. At that point, it is only a matter of gathering the details to fill in the blank cells in the spreadsheet.

In reality, time to gather the information is limited, so we compromise. If the value of the outcome does not justify such an effort, we often accept a suboptimal decision based on partial information or take the first option that meets some baseline standard. This behavior by individual decision-makers, termed "satisficing" by Simon [23], results from the inherent tradeoff between the cost of deciding and the benefit of a well-reasoned decision that utilizes as much information as possible. *This project aims to help people do better.*

The rapid development of online task markets, such as Amazon Mechanical Turk, has opened new options. Now, one can easily hire web workers to gather information, paying a small reward for a small amount of work.

A commonly overlooked limitation of most such systems is that they assume every question is mandatory. Requests are sent and then the requester waits until all results have been received. In contrast, individual information foraging tends to follow a dynamic workflow [17]. Especially with decision-making, users seek out the information that will have the greatest impact on the final result, constantly adjusting the strategy based on the information seen so far.

AskSheet is a research prototype we built to demonstrate a new method of coordinating workers to support decision making tasks. To reduce human effort wasted gathering unnecessary information, it leverages a model of the decision provided by the user in the form of a spreadsheet. By efficiently offloading the information task, a decision-maker may be able to avert the all too common tendency to compromise, potentially leading to a more optimal decision.

To use AskSheet, the user (someone with a decision to make) starts with an ordinary spreadsheet with formulas that will calculate the end result. Then, in cells where information is needed, the user enters a special `=ASK(…)` formula to indicate to the server that some information is needed in that cell. The parameters specify such details as the type and range of values that are expected. AskSheet analyzes the formulas and manages the sending of requests. To reduce human effort, it calculates the *value of information* [13,21] of each `=ASK(…)` formula and eliminates those that cannot affect the final result. With this analysis, it aggressively prioritizes the inputs to minimize unnecessary requests.

The key contribution of this paper is a fully operational system for efficiently coordinating crowd workers to gather the input data for spreadsheet decision models.

Although our efforts so far have targeted workflows with Mechanical Turk, our longer-term goals for AskSheet extend to other types of group decision workflows, such as screening job applicants. That vision is reflected in the design of the system, and will be discussed later in the paper.

NEED FOR RESEARCH

AskSheet can be viewed as efficient human computation driven by a declarative notation based on spreadsheets. We examine the related work as such.

Efficient human computation

Human computation is a paradigm for computation that delegates to humans parts of the problem that computers cannot solve adequately [18,25]. Often, it starts with the assumption that you have a set of questions for which you would like to receive answers. Many such systems aim to reduce the number of tasks that workers are asked to perform by adjusting the number of judgments per question [10,22], the number of iterations on incremental improvement jobs [6,8,15,22], the strategy for decomposing tasks into subtasks [2,3], or by delegating some tasks to machines [19].

AskSheet is different in that it optimizes the number of *questions*—akin to telling the user, "No, actually you don't need these ones." A user shopping for a house might ask for several facts about each of many homes for sale, but if AskSheet can determine that some will not affect the user's final choice, those questions will be culled, and not asked. This notion of "efficiency" has been relatively unexplored.

AskSheet is built on the premise that human time is more precious and plentiful than CPU time, and thus any opportunity to use machine cycles to reduce human burden is a positive value proposition. Active learning [24], a machine learning technique, does this to some extent, but its focus is on efficient training of predictive models, as opposed to solving a single, monolithic problem, such as a decision.

Declarative crowdsourcing

Several recent projects have demonstrated a new, declarative approach to describing human computation processes in SQL [7,16]. We see such work (and our own) as exploring a big idea in crowdsourcing: By firmly separating the computational logic from the details of posting tasks and managing responses, the workflow can be optimized using computational methods that are independent of those details.

Spreadsheets

The system with the perhaps most resemblance to AskSheet is SmartSheet, a commercial web-based platform for many kinds of collaboration, including crowdsourcing via Mechanical Turk [9]. Using the site, a user creates a spreadsheet-like document with the row and column headers, and then issues a request for workers to fill in columns with needed data. However, once the process has begun, it has no automatic mechanism to stop once enough information has been gathered to solve the user's overarching problem.

Although not related to crowdsourcing or collaboration, the implementation of AskSheet has parallels with an early spreadsheet system by Lewis that operated on uncertain values, represented as inequality constraints [14]. They were propagated throughout the spreadsheet with formulas. As we will explain, AskSheet treats every cell as a random variable, which is similarly propagated through the formulas.

ASKSHEET

We describe the use of AskSheet with a simplified illustration. Later, we will show how the system supports applications with more realistic requirements.

Example: Grocery stores

Andrew operates a catering business that needs to buy a substantial amount of food each week—typically about 50 items totaling about $1,000. Depending on the week, any of the 5 grocery stores nearby might have some of the items on sale. All of the grocery stores make their weekly flyers available online in PDF format, but there is no API or central database of current grocery prices. Any such database would have to contain prices for every locality, and update them weekly as prices change. Therefore, Andrew will hire web workers to search through the flyers for sale prices on the items so he can get the best deal possible at a single store.

Starting with a blank spreadsheet, he enters the needed ingredients, the names of the stores, and some formulas. (For compactness, we use only 9 items and 3 stores.)

	A	B	C	D	E	F
1				Store A	Store B	Store C
2	Coffee, ground	32 oz	$10 to $15			
3	Heavy cream	1 gal	$10 to $20			
4	Hormel ham slices	3 lbs	$2 to $6	=ASK(C2)		
5	Johnsonville Brats	3 lbs	$5 to $16			
6	Ribeye steak	1 lb	$8 to $16			
7	Roasted chicken	1	$5 to $8			
8	Round steak	1 lb	$2 to $4			
9	Salmon filets	1 lb	$5 to $10	=SUM(F2:F10)		
10	Tide laundry liquid	>150 oz	$10 to $20			
11				0	0	0
12	Lowest price	0	=MIN(D11:F11)			
13	Cheapest store	Store A				
14						

=INDEX(D1:F1,1,MATCH(B12,D11:F11,0))

Cells D11, E11, and F11 calculate the sums of the prices at each stores. Cell B12 calculates the minimum of those totals.

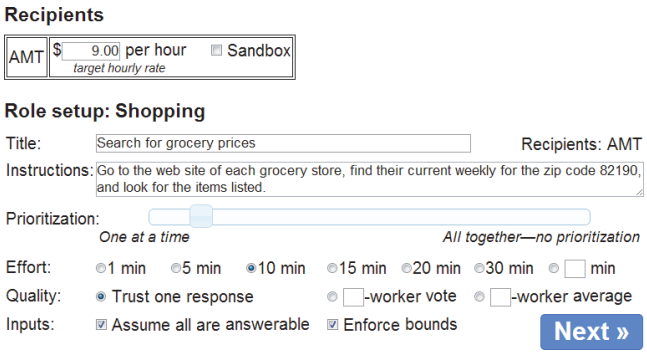
Cell B13 selects the name of the store with the lowest total. Note that `=SUM(…)`, `=MIN(…)`, `=INDEX(…)`, and `=MATCH(…)` are standard functions found in most spreadsheet applications; they are not specific to AskSheet.

In cells D2:F10 (columns D through F, rows 2 through 10), he will enter `=ASK(…)` formulas. `=ASK(…)` formulas *are* specific to AskSheet, and indicate requests for information.

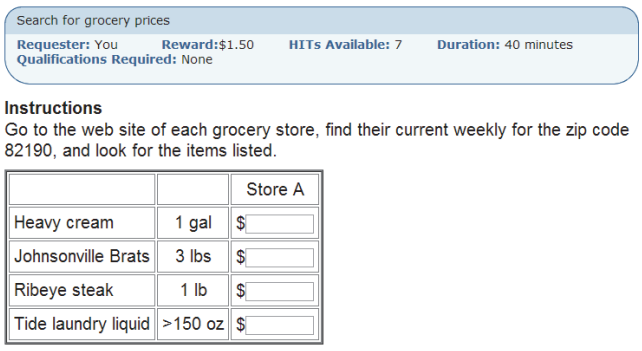
The formula in D2, equivalent to `=ASK("$10 to $15")`, means that Andrew expects the 32 ounces of ground coffee to cost between $10 and $15. (More details of the `=ASK(…)` function parameters are given in Figure 2.)

Note that if the price ranges he supplied were too wide, the prioritization might be less effective. The system might request the other value unnecessarily. On the other hand, if the ranges were too narrow, it might stop too soon.

Next, he enters some instructions for the workers and other details the system needs in order to post the tasks and manage the quality of the results. The "Prioritization" slider controls how many inputs to include in each batch.



Based on the contents and structure of the spreadsheet, AskSheet generates a form for entering data. If *Enforce bounds* was selected in the setup, then AskSheet will require that workers enter only values in the specified bounds. This may enhance data quality in some cases (e.g., 5-star rating must be between 1 and 5) but would not be appropriate for values like prices where values outside the bounds may be possible, or even desirable (e.g., lower price than expected).



AskSheet posts the tasks and manages the entire process. Andrew can monitor the progress from a dashboard. The braced expressions show the possible range of output values for each cell. The shaded colors indicate the relative priorities of the cells. Obtaining the dark green cells first would provide the greatest opportunity to eliminate other cells. The system will request those inputs first, in order to reduce the overall human effort—and hence, Andrew's cost.

			Store A	Store B	Store C
Coffee, ground	32 oz	$10 to $15	{10···15}	{10···15}	{10···15}
Heavy cream	1 gal	$10 to $20	{10···20}	{10···20}	{10···20}
Hormel ham slices	3 lbs	$2 to $6	{2···6}	{2···6}	{2···6}
Johnsonville Brats	3 lbs	$5 to $16	{5···16}	{5···16}	{5···16}
Ribeye steak	1 lb	$8 to $16	{8···16}	{8···16}	{8···16}
Roasted chicken	1	$5 to $8	{5···8}	{5···8}	{5···8}
Round steak	1 lb	$2 to $4	{2···4}	{2···4}	{2···4}
Salmon filets	1 lb	$5 to $10	{5···10}	{5···10}	{5···10}
Tide laundry liquid	>150 oz	$10 to $20	{10···20}	{10···20}	{10···20}
			{57···115	{57···115	{57···115
Lowest price	{57···115}				
Cheapest store	{3 values}				

As results are received, AskSheet recalculates the priorities based on the new information, and posts new tasks, as needed. When enough data has been entered to calculate a conclusive answer, it displays the result of that formula.

			Store A	Store B	Store C
Coffee, ground	32 oz	$10 to $15	not needed	12	10
Heavy cream	1 gal	$10 to $20	20	not needed	10
Hormel ham slices	3 lbs	$2 to $6	not needed	4	not needed
Johnsonville Brats	3 lbs	$5 to $16	5	not needed	5
Ribeye steak	1 lb	$8 to $16	16	12	10
Roasted chicken	1	$5 to $8	not needed	7	not needed
Round steak	1 lb	$2 to $4	not needed	3	not needed
Salmon filets	1 lb	$5 to $10	not needed	8	5
Tide laundry liquid	>150 oz	$10 to $20	20	not needed	10
			{85···104}	{71···102}	{59···68}
Lowest price	{59···68}				
Cheapest store	Store C				

In this fictional illustration, 16 of the 27 inputs have been entered. The range of possible totals for each of the stores has been constrained just enough that a winner can be conclusively determined. The worst case (upper bound) of Store C ($68) would still be better than the best case (lower bound) of either Store B ($71) or Store A ($85).

A primary benefit of using AskSheet is that it does not need to fulfill all of the requests. It is able to obtain a conclusive result that satisfies the user's spreadsheet model, with only partial information. This is similar to what individuals do when they have seen enough information to make a decision, even without seeing every detail of every alternative. For Andrew, this means lower cost and/or faster turnaround.

It should be emphasized that AskSheet does not depend on this particular decision model. The user starts with a blank spreadsheet and creates a model based on their needs. The example models in this paper are not part of AskSheet itself.

Usability

Like most current research in human computation, our primary focus is on new ways to coordinate the crowd to accomplish work. The interface we use to control AskSheet may someday be adapted for use by general users, but its usability is not an intended contribution of this work. Although many users are fluent with the use of spreadsheet formulas, many more are not. In the future, we plan to refine

the user experience of creating models, starting the requests, and viewing progress. To support users who are not experts in spreadsheets, we plan to build simpler interfaces for common models, effectively hiding the spreadsheet details.

AskSheet is however designed to streamline the activities of workers. To that end, it balances the goal of gathering inputs in the most efficient order with the reality that workers can be more efficient if they can gather several inputs from each web search conducted.

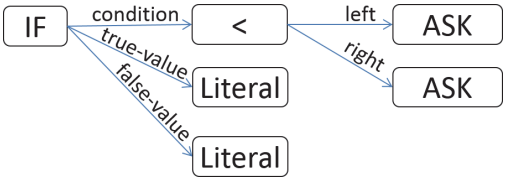
IMPLEMENTATION

AskSheet is implemented as a web application coded in Python. The user creates the model in Google Spreadsheets and imports it into AskSheet, which extracts the formulas.

To illustrate how the prioritization algorithms work, we will refer to this very trivial model.

	A	B	C
1	=ASK("1 to 5")	=ASK("1 to 10")	=IF(A1 < B1, "true", "false")

The first step is to parse each formula into an abstract syntax tree (AST). These trees are joined wherever there is a cell reference, resulting in the structure in a structure like this.



By analyzing the dependence relationships between cells, the planner can infer which cells the user is likely to care about (e.g., the `=INDEX(…)` function in cell B10 of the grocery shopping example). We call these the *roots* of the resulting graph because they depend on other cells, but other cells do not depend on them. The root of this trivial model is C1.

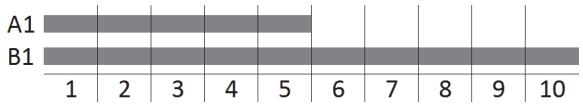
The goal is to find an ordering of the requests that will minimize the expected cost of evaluating the model—finding a conclusive result for all root cells. We think of the *cost of an ordering* as the expected sum of the costs from all requests that would be fulfilled (i.e., not eliminated) if they were gathered in that order. We define the *resulting expected model cost* of fulfilling a particular request as the expected cost of evaluating the entire model, if that request were fulfilled next. This cost should be regarded as a heuristic.

(In this section, we refer to `=ASK(…)` functions as *requests* or *r* since *i* (inputs) would be confusing as a variable name.)

An advantage of using spreadsheets for this work is that the order of evaluation for most operations is effectively arbitrary. Often, evaluating operands in one order can yield a higher probability of eliminating one of the others by short-circuit evaluation or some form of lazy evaluation. This is the fundamental advantage that AskSheet exploits.

When evaluating the comparison in the `=IF(…)` function, we could evaluate A1 and B1 in either order: (A1, B1) or (B1, A1). To prioritize, AskSheet considers the range of possible values for A1 and B1, from the `=ASK(…)` formulas.

In our current implementation, every possible value is treated as equally probable, i.e., discrete uniform distribution.



Suppose we choose the latter and evaluate B1 first. If the request in B1 receives any of the values 6, 7, 8, 9, or 10, then the comparison A1 < B1 must be true, no matter which value A1 receives. Also, if B1 is 1, then A1 < B1 must be false. Therefore, we would only need A1 if B1 turns out to be 2, 3, 4, or 5. In other words, Pr(need A1 | have B1) = 0.4.

On the other hand, if we evaluate A1 first, no matter what the value, we will need B1. Thus, Pr(need B1 | have A1) = 1.0.

Based on the above, we can say that the resulting expected model cost of fulfilling A1 first is 2.0 because if A1 is fulfilled next, we will definitely need both B1 (cost=1) and A1 (cost=1) with probability 1.0. The resulting expected model cost of fulfilling B1 is 1.4 because we will need A1 with probability 0.4 and we will need B1 will probability 1.0.

More generally, for any model M we will calculate the expected resulting model cost of fulfilling each request that the root cells depend on. Sorting by that will give us our final ordering. The resulting expected model cost is calculated as:

$$\mathrm{E}\big(\mathrm{C}(M)\big) = \sum_{r \in reqs} \mathrm{C}(r)\Pr(M \text{ needs } r)$$

… where C(r) is the cost specified in the `ASK(…)` function parameters, *reqs* is the set of all requests in the model, C(M) is the cost of evaluating the model, and Pr(M needs r) is the probability that evaluating every root in the model will require fulfilling request r. That can in turn be expressed as the probability that any root node will need request r.

$$\Pr(M \text{ needs } r) = P\left(\bigvee_{n \in roots} n \text{ needs } r\right)$$

The probability that a particular node n needs a request r is calculated recursively:

- If n is a request and $n = r$, then P(n needs r) = 1. (Every request node needs itself.)
- If n is a request and $n \neq r$, then P(n needs r) = 0. (A request node never needs any other nodes.)
- For any other node type, the probability is:

$$\Pr(n \text{ needs } r) = \Pr\left(\bigvee_{op \in Ops_n} op \text{ needs } r \ \wedge\ n \text{ needs } op\right)$$

… where Ops_n is the set of operands (i.e., child nodes) to n.

For simple arithmetic operations, P(n needs op)=1 for all of the operands, since you cannot calculate an arithmetic operation without all of its operands (except for the case of multiplication by zero). For example, to calculate $a + b$, you need both a and b, regardless of the bounds or distributions.

AskSheet prioritizes the requests by the conditional expected model cost E(C(M | r next)), conditioned on acquiring each request r next. It is equivalent to the this utility function:

$$\mathrm{U}(r) = \mathrm{E}\big(\mathrm{C}(M)\big) - \mathrm{E}(\mathrm{C}(M) \mid r \text{ next})$$

In essence, this is the expected overall savings if we acquired *r* next, versus choosing parameters randomly at every step.

Note that the cost is based on marginal probabilities over all possible values of other inputs in the model. The calculations assume parameters are evaluated in random order by default. (For `=IF(…)`, the condition is always evaluated first since it would not make sense to evaluate the value parameters first.)

For operations that can potentially be optimized to eliminate requests—including the comparison operators, `=MIN(…)`, `=MAX(…)`, `=IF(…)`, and many others—calculating the probability that the node needs each operand generally entails calculating the output distribution (probability mass function) of the node—every possible output value and its probability relative to the decision inputs (random variables).

Operations

The current system supports a small but important subset of core spreadsheet formulas: `=IF(…)`, `=MAX(…)`, `=MIN(…)`, `=SUM(…)`, `=AND(…)`, `=OR(…)`, `=NOT(…)`, `=INDEX(…)`, `=MATCH(…)`, and operators (+, -, /, *, =, <, <=, !=, >=, >). Efficient algorithms for calculating the output distribution and need probabilities are specific to each these. Below, we show how these are calculated for one example.

Example: `=MAX(…)`

The output distribution for `=MAX(…)` is the joint distribution of possible values the formula could take, for all possible values of its parameters.

Since `=MAX(…)` is synonymous with the k^{th} order statistic for a discrete random variable with *k* possible values, we can use the probability mass function for discrete order statistics:

$$\Pr(\max = k) = \Pr(\max \le k) - \Pr(\max < k)$$
$$= \Pr\left(\bigwedge_{op \in Ops_n} op \le k\right) - \Pr\left(\bigwedge_{op \in Ops_n} op < k\right)$$

The need probability Pr(max needs *op*) for each operand of `=MAX(…)` is calculated on the premise that an operand is needed only when there was some chance that it could be greater than the maximum. In other words, for each possible max value *k* in the output distribution calculated above, we temporarily assume that was the max value, and then set the cost for that case as the sum of expected costs for every operand *op* for which $\Pr(op > k) > 0$. With that, we calculate an expected value over every possible value of *k*.

Scope

AskSheet offers a unified process for efficient application of crowdsourcing to a wide range of decision making tasks. It is especially suited to decisions that entail gathering a lot of details and then selecting a subset of them, typically based on the relation to the rest. This includes the examples above and, more generally, any problem that can be modeled using operations such as minimum, maximum, if/else, conditionals, and so forth. Since expected bounds of the inputs are specified by the user a priori, such operations can often be precisely computed with only partial information.

AskSheet is not intended for needle-in-a-haystack search problems (e.g., find Mayor X's salary). Also excluded are problems where a large set of information is gathered and all of it is to be used (e.g., find salaries of all mayors in the US).

LIMITATIONS OF CURRENT IMPLEMENTATION

The calculations described above have enabled us to test the core concept embodied by AskSheet. Below, we summarize some known limitations and outline some potential solutions.

Performance

In our current implementation, models with `=MAX(…)` and `=MIN(…)` formulas are currently practical only in modest sizes (i.e., up to around 30-40 parameters). This affects weighted sums models, a common type of decision model.

There is no closed form expression for the running time, since it depends on specifics of the model, such as how closely spaced the weights are. For a weighted sums model with *a* alternatives each having *b* attributes with *c* levels per attribute, and all weights being the same, running time is $O(a^2b^2c)$. The theoretical worst case, $O(a^2b^2c^2)$, could only occur if the weights were spaced in an extremely unlikely pattern. These were determined by analyzing the code.

The algorithm can compute prioritization in polynomial time because instead of considering every possible permutation, it simply ranks requests by the expected cost savings that would result if each one were fulfilled next. That can be calculated in polynomial time for a given request, and so calculating it for every request is also polynomial.

To understand this practically, we measured the time to prioritize inputs in several WSM models. All weights were set equal to 1 (the best case) and we assumed 5 levels for every attribute. This represents a decision where all attributes are scored from 1 to 5 and are equally important. As a spreadsheet, it would occupy $a \times b$ cells, plus the labels.

These were run on a desktop computer with an Intel model i7-3770K 3.50 GHz CPU and 32 GB RAM using the Python 2.7 interpreter. The results are shown in Figure 1.

This time is important because when a worker submits a form, the prioritization must be recalculated in real-time, before the answers are accepted. AskSheet only posts a small number of tasks (HITs) at a given time, to avoid collecting responses that will not be needed. To make this possible, as a worker submits a form, it recalculates the prioritization and, if more inputs from that role (HIT Type) are still needed, it posts another task—all before accepting the worker's data.

number of alternatives \ number of attributes	***5***	***10***	***15***	***20***	***25***
10	0.1	0.5	1.1	2.0	2.9
20	0.6	2.2	4.7	8.2	12.0
30	1.5	5.3	11.8	20.9	30.9
40	2.9	10.6	23.4	42.8	61.9
50	5.0	18.3	41.6	73.8	108.8

Figure 1. Time (secs.) to prioritize a benchmark model based on the sum of 5-star ratings for each alternative.

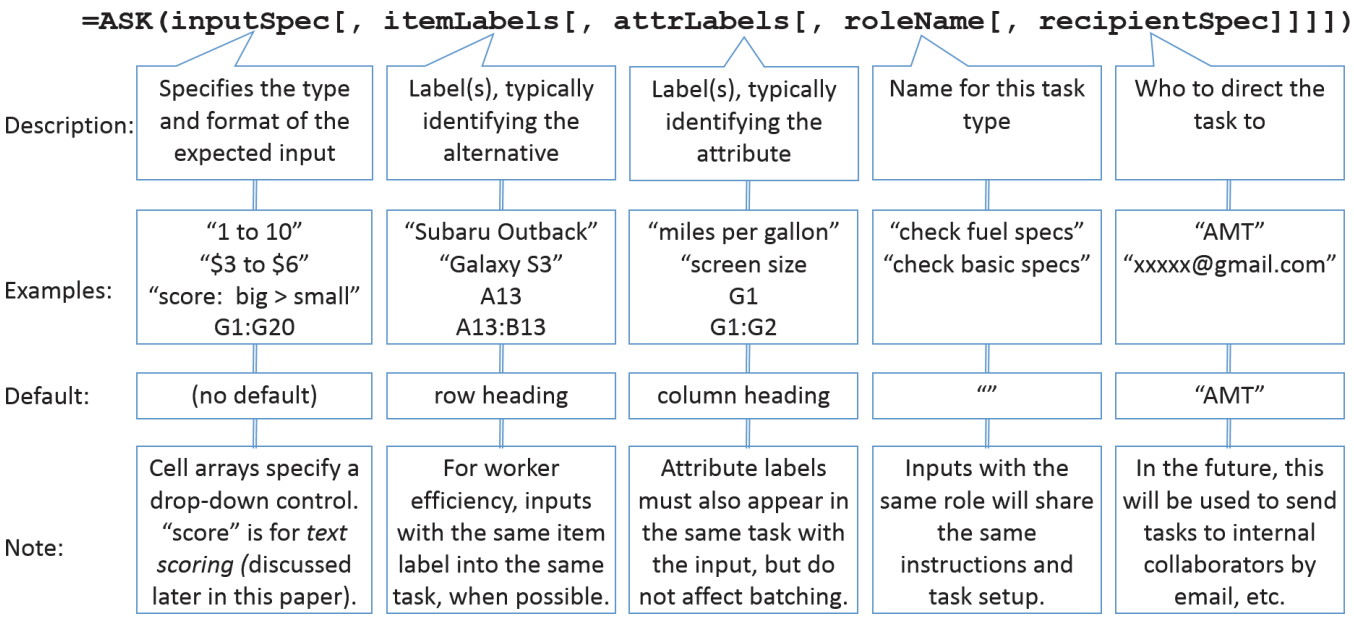


Figure 2. `=ASK(...)` formula parameters are at the core of the strategy for partitioning inputs into efficient tasks.

To reduce the recalculation time, we could keep the state from the last calculation, and then update the output distributions and need probabilities incrementally. Recalculation time is most important since it affects the delay that workers experience when submitting the form.

To improve the initial calculation time, we are aware of several untapped opportunities for optimizing the code. Using dynamic programming, we previously improved performance of calculations for the `=SUM(...)` operation. We believe a similar approach could be used for other operations.

For some operations, we have not found computationally feasible algorithms for calculating output distributions and need probabilities. These include `=RANK(...)` and `=LARGE(...)` (pick n^{th} largest value). To support those, and also increase the number of parameters that can be supported with the current set of operations, we are exploring options for approximating the probabilities by random sampling. This is complicated by the large number of parameters.

Finally, since AskSheet is implemented in pure Python, these times could be improved by a constant—but substantial—factor by porting critical modules to C and splitting the calculations acrosss multiple CPU cores. The algorithms are extremely parellelizable because they compute the probabilities for a large number of scenarios.

Dependence between formulas

The algorithms that calculate output distribution and need probabilities use only local information (i.e., at the level of a particular function or operator), so they cannot detect dependence between cells. For example, in this simple model, AskSheet *should* detect that A1 and B1 refer to the same quantity, and thus elimincate A1, i.e., Pr(need A1)=0.0.

	A	B	C
1	=ASK("1 to 10")	=A1*1	=IF(A1=B1, "same", "different")

The problem is that when analyzing the condition in C1, it has no knowledge of the relationshiop between A1 and B1. Although it calculates that both A1 and A2 have the same distribution (discrete uniform, 1 to 10), it cannot detect the dependence between the two, and so it fails to eliminate A1.

In our experience, the most common area where this issue is apparent is with comparison operators, as well as spreadsheet functions that depend on comparisons internally, such as table lookups. For example, in the grocery shopping example above, the system will continue to calculate the entire total cost for a store, even though the actual number is not needed to determine which store is the winner.

These issues do not arise with tree-structured models (i.e., each cell referenced by at most one formula). They only arise in DAG-structured models, because the local information used by the probability calculations does not provide information on these relationships.

A potential solution we intend to explore is to simplify the formulas algebraically before calculating need probabilities for comparison operators. Software libraries exist for performing such algebraic manipulation [11].

When algebraic simplification determines two nodes to be the same quantity, then the output distribution of the equality operator will be Pr(node=True)=1.0. This could be applied to functions that inherently involve comparison operations (e.g., table lookups), following a strategy similar to symbolic evaluation, a technique used in software testing [5].

These solutions might not fully eliminate the issue of dependence, which affects any calculation of conjunctions ($A \wedge B$) or disjunctions ($A \vee B$). However, it would reduce the effect on the prioritizations.

BATCHING INPUTS FOR WORKER EFFICIENCY

Up to now, we have assumed that that inputs are acquired one at a time. To make the task more efficient for workers, AskSheet can batch inputs that involve the same type of activity (e.g. searching for fuel economy specifications) and, where possible, the same source information (e.g., pages about a single car model). The user sets the batch size.

In addition, the parameters to `=ASK(…)` allow it to split the job into multiple roles (HIT Types) which are run in parallel. (See Figure 2.) The example below illustrates how the job of gathering information to find an acceptable pediatrician is split into four separate roles which run in parallel.

Field trial #1: Find a pediatrician

Alan needs to find a pediatrician for his daughter. He values information from doctor rating sites but does not trust one site alone. Recognizing that these sources alone would not be sufficient to find the *best* pediatrician, he will be satisfied to find one that matches his basic requirements:

- Average rating at RateMDs.com is at least 4 stars.
- Average rating at HealthGrades.com is least 80% positive.
- The doctor accepts Alan's insurance, CareFirst.
- Driving to the office takes no more than 20 minutes.

We tested this scenario using the model in Figure 3. The root formula in F53 simply evaluates whether *any* of the doctors conforms. Recall that Alan will be satisfied with any doctor who meets his minimum criteria. Thus, AskSheet's task is ostensibly just to determine whether any doctor satisfies the criteria. Once any doctor has been confirmed to fit the criteria, the value of F53 will be `TRUE`, and no more inputs are needed. Columns B, C, D, and E each becomes a separate role (HIT Type) which runs in parallel.

Note that the current implementation of AskSheet requires the user to fill in the row labels in the model. For this example, we found a list of pediatricians online. In the future, we plan to extend AskSheet so that workers can gather the row labels and extend the spreadsheet.

We received a conclusive answer in 47 minutes. With the prioritization, AskSheet required only 36 of the 200 inputs. The total cost was $5.67 with one judgment per input. We paid $0.63 per task. The 6 workers spent a combined total of 58 minutes, for an average hourly rate of $5.87 per hour.

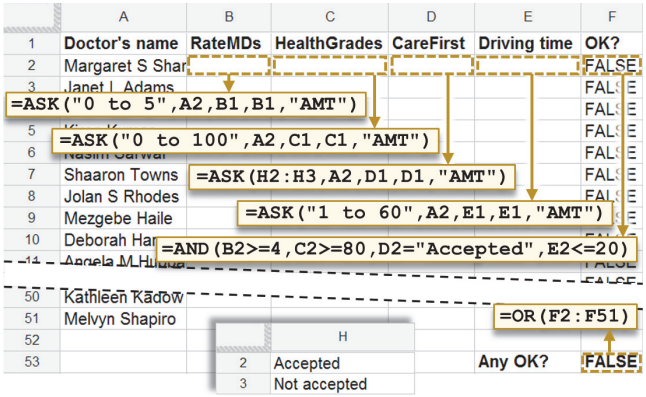


Figure 3. Model for field trial #1 (pediatrician)

Had all of the 200 inputs been fulfilled (i.e., the approach used by tools such as SmartSheet) the total cost would have been $31.50. The optimizations saved $25.83 (82%).

OPTIMALITY

Full optimality would be achieved only if a few simplifying assumptions were satisfied. They are listed below. To the extent these are violated, optimality may be diminished.

- *Input values are presumed to be uniformly distributed and mutually independent.* This affects the calculated output distributions, which are used to calculate need probabilities. For `=MAX(…)`, AskSheet effectively prioritizes inputs for the top contenders in order to rapidly differentiate which is the maximum. This assumption may lead to inaccurate selection of the top contenders.
- *Bounds given in `=ASK(…)` formulas are presumed accurate.* If there are actual values outside the bounds (e.g., lower price than anticipated), it may stop too soon.
- *Each cell is referenced by at most one node.* This is the problem of dependence between formulas described above. It can lead to gathering more inputs than necessary.
- *At each step, only one input is acquired from one worker.* The batching mechanism described above violates this. However, making tasks more efficient for workers may reduce overall cost and completion time. The user controls this tradeoff via the `=ASK(…)` parameters and the setup.

Single-step assumption

AskSheet considers only the effects of the next input acquired, and greedily acquires whichever maximizes overall expected cost savings. In many value of information applications, this can diminish optimality, especially for those that optimize cost and another utility function [13,21].

Within the above assumptions, the single-step assumption does not affect AskSheet. We will explain with this example.

	A	B	C
1	=ASK("0 to 1")	=ASK("0 to 100")	=AND(A1=0, B1=0)
2	=ASK("0 to 1")	=ASK("0 to 100")	=AND(A2=0, B2=0)
3	=ASK("0 to 1")	=ASK("0 to 100")	=AND(A3=0, B3=0)
4			=OR(C1:C3)

If either A1, A2, or A3 were chosen first, then the expected model cost would be 3.88. However, choosing B1, B2, or B3 first would reduce that to 3.61, for a utility of 0.27. (These were calculated by AskSheet, and will not be derived here.) Below, the dashboard is annotated with all of the utilities.

	A		B		C
1	{0…1}	U=0.00	{0…100}	U=0.27	{false…true}
2	{0…1}	U=0.00	{0…100}	U=0.27	{false…true}
3	{0…1}	U=0.00	{0…100}	U=0.27	{false…true}
4					{false…true}

Although the *pair* of (A1, B1)—or (A2, B2) or (A3, B3)—could eliminate the other four cells, AskSheet does not know this. Nevertheless, optimality is not affected, as we will see.

Suppose we enter 0 in cell B1. Then, A1 gets the highest utility since it could eliminate the remaining cells. Taking

A1 next, the expected model cost would be 2.28, versus 3.01 if B2 or B3 were chosen next, or 3.22 if A2 or A3 were next.

	A		B		C
1	{0…1}	U=0.93	0		{false…true}
2	{0…1}	U=0.00	{0…100}	U=0.21	{false…true}
3	{0…1}	U=0.00	{0…100}	U=0.21	{false…true}
4					{false…true}

The result is the same as if the pair had been selected together. More generally, because utility is based on marginal probabilities over all other values, there is no extra value to selecting pairs (or *n*-tuples) together, as long as the input with single highest utility is acquired at each step.

In practice, it is often more efficient to gather a few related inputs at once. This *can* affect optimality, although the user can control that tradeoff in the setup panel. If the user had set batch size to 2, B1 and B2 would have been acquired first.

QUANTIFYING UNSTRUCTURED INPUTS

Up to this point, we have discussed only models where all inputs are either numbers or discrete choices. AskSheet handles unstructured, non-numeric inputs using a method we call *text scoring*. It leverages the "score" input type mentioned in the `=ASK(…)` parameters (Figure 2).

Field trial #2: Car shopping

The use of text scoring is illustrated by the following scenario. Jay is shopping for a new car and has these criteria:

- Hatchback with 4-wheel-drive
- Good fuel efficiency
- Wheel base about 100 inches (similar to Jay's former car)
- Abundance of modern safety features
- Seats 5 adults comfortably (for long car trips)

As before, Jay creates a spreadsheet model. The last two criteria are difficult to quantify. Although they are somewhat subjective, Jay will write his criteria and ask workers to simply specify how well each car model fits what he wants.

Workers are given two car models to research at a time. For safety features, they are asked to write a textual description of how spacious they think the inside of the car would be, as it would relate to traveling with 5 adults. They are instructed to look for photographs of the interior or other metrics that would help. The first worker is asked to compare the two models based on which seems to be most spacious. A similar course was used to compare by safety features (Figure 4).

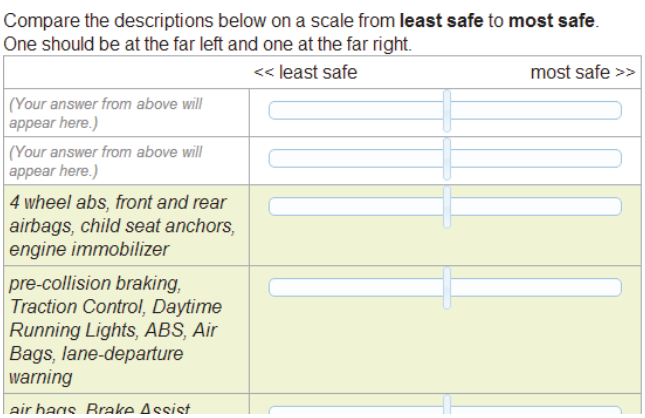


Figure 4. Text scoring interface

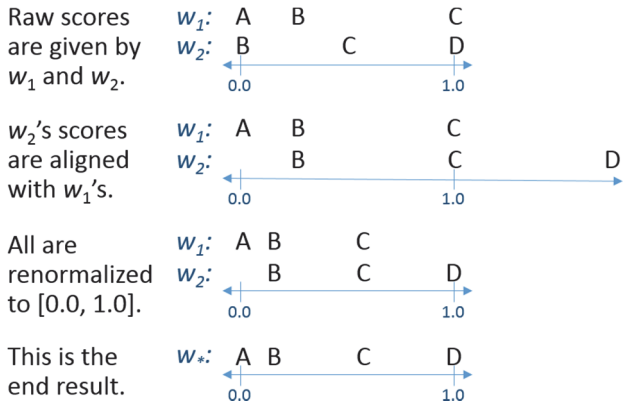


Figure 5. In this minimal example, text scores from two workers who each scored three items are aligned.

To enable a global view across tasks and/or workers, AskSheet asks subsequent workers to compare the features on the car model they have researched together with the text descriptions entered in up to four prior tasks. Figure 4 shows this comparison with two prior values.

Each subsequent worker who does the task adds some new descriptions and is shown some prior descriptions for reference. To reconcile all of the scores, AskSheet scales and translates all of the scores for a particular field onto a common coordinate system (i.e., offset and scale). This is analagous to if all of the scores were in different unknown units, and there were just a few measurements of the same quantity with which to convert the units of one to another.

Two common scores would be sufficient if the information were perfect. A challenge is that workers may disagree on the relative score of items, and the values on the slider may not be precise, in the absence of more reference points. Therefore, we collect up to four comparative scores with each task in order to accumulate redundant information and use the method of least squares to solve for the set of scale factors and offsets that gives the best fit (Figure 5).

When we ran the model to choose between 20 current model cars, the results for the text scoring agreed with our own assessment of the cars (e.g., Volkswagon CC = 0.14, Volvo C30 = 0.83). Although this method is not designed to be statistically rigorous, in the context of a model with various kinds of information, it allows us to add these unstructured inputs. In this case, because these items were mixed with numeric and choice answers, only 8 of the 20 cars were compared for safety features and spaciousness.

We received a conclusive answer (Subaru Outback) in 2 hours 4 minutes. With the prioritization, AskSheet required only 62 of the 120 inputs. The total cost was $7.50 with one judgment per input. We paid $0.50 per task. The 12 workers spent a combined total of 1 hour 34 minutes, for an average hourly rate of $4.75 per hour.

Had all of the 120 inputs been fulfilled, the total cost would have been $12.50. The optimizations saved $5.00 (40%).

This strategy was adopted to allow a richer set of models, but is not intended to be primary basis of a model. A more thorough study of text scoring is needed to fully understand its ability to reconcile scores from different contributors.

Prioritizing without known bounds

Another challenge is that there is no way to specify bounds a priori for such unstructured quantities. To solve this, at each step, after reconciling the scores received so far, AskSheet estimates how much of the total space has been seen so far.

This is accomplished using maximum spacing estimation (MSE) [20], a method of estimating properties of a distribution based on a limited sample. The MSE formula for uniform distributions gives us an estimate of the global maximum and minimum (e.g., Relative to cars seen so far, what is the most spacious car on the consumer market?).

$$max_{estimate} = \frac{n \cdot max_{samples} - min_{samples}}{n-1}$$

$$min_{estimate} = \frac{n \cdot min_{samples} - max_{samples}}{n-1}$$

Although this relies on our presumption of uniform distribution, which is very unlikely to be accurate for these fields, the effect is what we need: The bounds narrow as more inputs are received. From that, we calculate a scale factor and offset that are applied to all scores in a given field.

$$scalefactor = \frac{1.0}{max_{estimate} - min_{estimate}}$$

$$offset = -min_{estimate} \cdot scalefactor$$

That pushes all values toward the middle of the space from 0.0 to 1.0. The prioritization assumes a minimum and maximum of 0.0 and 1.0, respectively, so this effectively causes the prioritization to assume greater and/or lesser values may still exist. As more values are received, this margin is decreased gradually.

This approach while straightforward, is necessary to enable a richer set of models using a variety of types of data. It is not necessary to discover the absolute maximum or minimum for any one attribute. Eventually, with enough data, one decision alternative in the decision will prevail.

Field trial #3: Smartphone shopping

Another example we have used throughout the project development is comparison shopping. Using a spreadsheet, a user can model their exact personal preferences, and include details that might not be important to all shoppers. The simplest way to model this is with a weighted sums model, which depends on the `=MAX(…)` and `=SUM(…)` formulas. Web workers can be employed to search the web for details about the alternatives.

We created a model to compare seven leading smartphones by the following criteria:

- Weight=1: screen size, material, appearance, CPU cores
- Weight=2: physical shutter button, camera megapixels
- Weight=4: battery talk time, storage potential, RAM

For appearance, it was set up to ask workers to look at a photo of the phone and compare based on the scale of "solid" to "cheap-looking". As in the example of car shopping, we separated this unstructured input into a separate role. Because it had a relatively low weight, it was prioritized low and ultimately not utilized for this field trial. Since this is a more time-intensive task for workers (and we set it to pay a higher reward), this demonstrates how prioritizing inputs can save worker effort. Which cells will be needed cannot be predicted in advance. The result is shown in Figure 6.

We received a conclusive answer (Samsung Galaxy S4) in 1 hour 29 minutes. With the prioritization, AskSheet required only 40 of the 63 inputs. The total cost was $8.00 with one judgment per input. We paid $0.80 per task. The 4 workers spent a combined total of 1 hour 5 minutes for an average rate of $7.38 per hour.

Had all of the 63 inputs been fulfilled, the total cost would have been $12.83. The optimizations saved $4.83 (38%).

QUALITY CONTROL

AskSheet includes a few mechanisms for controlling quality. First, in the setup screen, the user can request multiple judgments. The judgments can be aggregated by averaging (for subjective ratings) or voting (for objective information).

	Weights	Nexus 4	Samsung Galaxy S3	Samsung Galaxy S4	Sony Xperia T7	Kyocera Rise	HTC One	iPhone 5
Camera quality	*2*	8	8	13	8	3	4	8
Battery, talk time, hours	*4*	12	11	12	7	9	12	8
Screen size	*1*	not needed	3	4	not needed	not needed	3	not needed
Max storage (built-in + card)	*4*	0	2	4	2	2	4	4
Material	*1*	not needed	2	2	not needed	not needed	2	not needed
Appearance	*1*	not needed	not needed	not needed	not needed	not needed	not needed	not needed
CPU, # of cores	*1*	not needed	4	4	not needed	not needed	4	not needed
RAM	*4*	4	2	4	2	0	4	2
Shutter button	*2*	not needed	0	0	not needed	not needed	0	not needed
Score		{68.3…93.3}	{76.3…77.3}	{101…102}	{48.3…73.3}	{45.6…70.6}	{94.1…95.1}	{60.3…85.3}
Best model		**Best score**						
Samsung Galaxy S4		{101…102}						

Figure 6. AskSheet's dashboard shows how 20 of the 63 inputs in this smartphone shopping model were eliminated. Note that many of the values shown are the values associated with labels in a drop-down selection control specified in the model.

AskSheet optimizes this, as well, requesting judgments incrementally. For example, if the user requests five judgments and the first three agree, it will not request any more. In fact, it only requests one at a time because there is always a chance that the input might be eliminated by the optimizations. This approach was inspired by Get Another Label [22], although it is far simpler than that system.

In addition to multiple judgments, AskSheet uses the input specification to validate the format and range of the inputs. In the setup screen, the user may specify that blanks not be allowed (i.e., all inputs required) and/or that ranges must be enforced (or not). With these options, the form does not allow submission until the inputs conform.

DISCUSSION

The models we used for these field trials allowed us to demonstrate AskSheet in the context of familiar personal decision problems within the technical boundaries of our current implementation. However, these examples do not capture the full expressive power of this framework.

In creating those models, we were confined by the limitations of our current implementation. We had to limit `=SUM(...)`, `=MIN(...)`, and `=MAX(...)` functions to around 30-40 parameters each. Moreover, the system does not yet support functions related to relative ordering, such as `=RANK(...)`, `=PERCENTILE(...)`, `=MEDIAN(...)`, `=LARGE(...)`, and `=SMALL(...)`. If it did, far richer models would be supported.

Spreadsheets are used in business contexts to model a wide range of decision problems [12]. In fact, the language of spreadsheets formulas can be considered first-order functional programming [1] and, as such, can be used to solve problems as permutations, combinations, and even the Towers of Hanoi problem [4]. The most fundamental requirement imposed by AskSheet—ignoring computational boundaries—is that the decision answer(s) be expressed as a formula(s) somewhere in the spreadsheet. However, some types of problems are more amenable than others.

From our experiences so far, we have learned that the method works best where the decision-maker knows in advance what aspects are important and something about the choices. In addition, there is a trade-off between the time the decision-maker must spend specifying the model and the instructions versus the time that is saved by delegating the work to others.

As we mentioned in the introduction, our vision for AskSheet extends beyond Mechanical Turk to internal organizational decisions that require inputs that are not readily on hand. Besides gathering facts from the web, this could include trusted judgments, such as evaluating materials submitted by applicants to a job or academic program. Spreadsheets have the advantage of creating a record of the decision rationale.

Instead of working through Mechanical Turk, internal users would enter inputs via a private web site. Since they would already be familiar with the context, we expect the burden of writing instructions would be less. However, since the "cost" of asking different people is rarely equal—in terms of hourly rate, social capital, etc.—the decision coordinator who creates the model would need an interface for specifying the relative cost of asking each collaborator. For example, asking the boss could be considered equivalent to asking an administrator five times. If the same model contained tasks for Mechanical Turk *and* internal collaborators, then it would need to relate social capital to monetary cost, as well.

MORE POTENTIAL APPLICATIONS

In this section we outline two use cases that illustrate our vision for the future, and how they can be modeled.

Example: Vacation

Vicky wants to take a vacation sometime in March for a few days. She has three destinations in mind, with a preference for Puerto Rico. Cost is important, too.

She makes a table of the travel dates that are compatible with her work schedule, and her preference (1 to 10) for each. She does the same for the candidate destinations.

Date	Preference
3/1/2013	4
3/2/2013	5
3/3/2013	5

Destination	Preference
Puerto Rico	10
Florida	5
Lake Michigan	5

Elsewhere, she enters weights that describe what aspects are important to here. Finally, she uses formulas to make a table of every combination of destination, departure date, and duration along with a calculated score for each combination.

Itineraries						
To location	*Depart*	*# of Days*	*Return*	*Fare*	*Hotel*	*Score*
Puerto Rico	3/1/2013	4	3/5/2013	970	480	48
Puerto Rico	3/1/2013	5	3/6/2013	530	600	72
Puerto Rico	3/1/2013	6	3/7/2013	710	720	27

At the top, a result formula displays the best option of all.

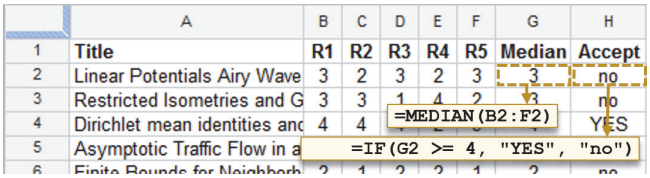
BEST CHOICE			
Destination	**Depart**	**Duration**	**Return**
Florida	3/22/2013	4	3/26/2013

The large number of combinations makes this infeasible with our current implementation. With a future version, Vicky could direct workers to check various travel web sites to find the best airfares, hotels, and other features for each location, date and duration. If the hotels at a destination are expensive, it would skip checking airfares for any of her listed dates.

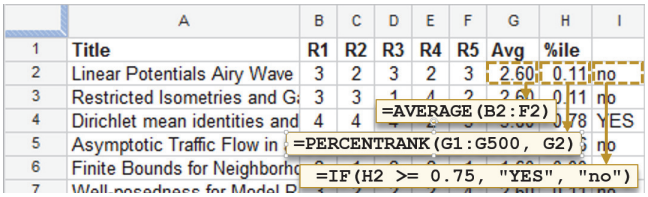
Example: Reviewing conference paper submissions

The XYZ conference has a very simple review process. It initially assigns three reviewers per paper. For those that are borderline, it adds more until a consensus forms.

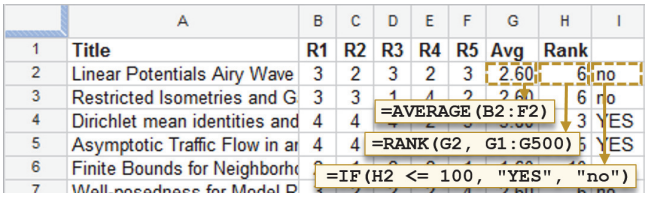
This could be modelled as accepting any paper where the median of five scores is at least 4.0 out of 5.0. If the first three are more (or less) than 4.0, then no more are needed.



They could have structured the process in other ways. For example, to target a 25% acceptance rate, it could accept every paper in the top 25 percentile, as this model illustrates:



To accept a specific number of papers (e.g., to fit the number of rooms), the `RANK(...)` function could be used as follows:



In each case, once there was enough information to be sure the process was complete, it would stop requesting reviews.

Note that since the submissions are confidential, reviewers would of course be trusted experts, not web workers. Also, the scores would be entered on a private, controlled web site.

CONCLUSION

We have presented AskSheet, a system that leverages the structure of a decision spreadsheet to coordinate an efficient human workflow to solve the user's overarching decision problem without necessarily gathering all of the inputs. It is fully operational with a basic set of spreadsheet operations.

The three field studies each generated decisions in about 1-2 hours for between $5.67 and $8.00, while paying workers between $4.75 and $7.38 per hour. This shows what we believe to be a financially and temporally viable process.

Extending this to larger, more complex decision models will require solving many technical hurdles. Adding support for spreadsheet functions related to ranking and relative ordering will enable new model types important for decision-making.

The potential benefits extend well beyond saving money or time. The ability to offload a complex task such as gathering information for a decision would give users new flexibility for delegating work. Also, a user who might otherwise choose to satisfice—accept a subpar option to avoid the tedium of researching others—would gain a viable alternative. Ultimately, we see this as an opportunity to effectively connect such users with the people who can help.

ACKNOWLEDGMENTS

We gratefully acknowledge the advice of Nicholas Chen, Chang Hu, Lise Getoor, Piotr Mardziel, Atif Memon, Jay Pujara, Ben Shneiderman, Keith Walker, Tom Yeh, and Ben Zou, as well as thoughtful feedback from the reviewers.

REFERENCES

1. Abraham, R., Burnett, M., & Erwig, M.. 2009. Spreadsheet Programming. *Wiley Encyclopedia of Comp Sci and Eng.*
2. Bernstein, M. S., Brandt, J., Miller, R. C., & Karger, D. R. 2011. Crowds in two seconds: Enabling realtime crowd-powered interfaces. *UIST '10.*
3. Bernstein, M.S., Little, G., Miller, R.C., et al. 2010. Soylent: a word processor with a crowd inside. *UIST '10.*
4. Casimir, R. J. 1992. Real programmers don't use spreadsheets. *ACM Sigplan Notices*, *27*, 6, 10-16.
5. Cheatham, Jr, T.E., Holloway, G.H. & Townley, J.A. 1979. Symbolic evaluation and the analysis of programs. *TOSE.*
6. Dai, P., Mausam, & Weld, D. S. 2010. Decision-theoretic control of crowd-sourced workflows. *AAAI '10.*
7. Franklin,M., Kossmann,D., Kraska, et al. 2011. CrowdDB: answering queries with crowdsourcing. *SIGMOD '11.*
8. Franklin, M. J., Trushkowsky, B., Sarkar, P., & Kraska, T. 2013. Crowdsourced enumeration queries. *ICDE '13.*
9. Frei, B. 2009. Paid crowdsourcing: Current state & progress toward mainstream business use. Smartsheet.com.
10. Ipeirotis, P.G., Provost, F., & Wang, J. 2010. Quality management on Amazon Mechanical Turk. *HComp '10.*
11. Joyner, D., Čertík, O., Meurer, A., & Granger, B.E.. 2012. Open source computer algebra systems: SymPy. *ACM Communications in Computer Algebra,* 45, 3/4, 225-234.
12. Kirkwood, C.W. 1997. *Strategic Decision Making: Multiobjective Decision Anal. w/ Spreadsheets.* Duxbury.
13. Koller, D. & Friedman, N. 2009. *Probabilistic graphical models: principles and techniques.* MIT Press. 1121-1125.
14. Lewis, C. 1985. Extending the spreadsheet interface to handle approximate quantities and relationships. *ACM SIGCHI Bulletin,* 55-59.
15. Little, G., Chilton, L.B., Goldman, M., & Miller, R. C. 2010. Exploring iterative and parallel human computation processes. *HComp '10.*
16. Marcus, A., Wu, E., Karger, D.R., Madden, S.R., & Miller, R.C. 2011. Crowdsourced databases: query processing with people. *CIDR '11.*
17. Pirolli, P. & Card, S.K. 1999. Information foraging. *Psychological Review,* 106, 4, 643-675.
18. Quinn, A.J. & Bederson, B.B. 2011. Human computation: a survey and taxonomy of a growing field. *CHI '11.*
19. Quinn, A.J., Bederson, B.B., Yeh, T., & Lin, J. 2010. CrowdFlow: integrating machine learning with Mechanical Turk for speed-cost-quality flexibility. Technical Report HCIL-2010-09, University of Maryland.
20. Ranneby, B. 1984. The maximum spacing method. An estimation method related to the maximum likelihood method". *Scandinavian J. of Statistics*, 11, 2, 93–112.
21. Shachter, R.D, & Peot, M.A. 1992. Decision Making Using Probabilistic Inference Methods. *UAI '92.*
22. Sheng, V., Provost, F., & Ipeirotis, P.. 2008. Get Another Label? Improving Data Quality and Data Mining using multiple noisy labelers. *KDD '08.*
23. Simon, H.A. 1972. Theories of bounded rationality. *Decision and organization 1*, 161-176.
24. Tong, S. & Koller, D. 2002. Support vector machine active learning with applications to text classification. *JMLR.*
25. von Ahn, L. 2005. *Human Computation.* Doctoral Thesis.