ABSTRACT

Title of dissertation: CROWDSOURCING DECISION SUPPORT: FRUGAL HUMAN COMPUTATION FOR EFFICIENT DECISION INPUT ACQUISITION

Alexander J. Quinn
Doctor of Philosophy, 2014

Dissertation directed by: Professor Benjamin B. Bederson
Computer Science

When faced with data-intensive decision problems, individuals, businesses, and governmental decision-makers must balance trade-offs between optimality and the high cost of conducting a thorough decision process. The unprecedented availability of information online has created opportunities to make well-informed, near-optimal decisions more efficiently. A key challenge that remains is the difficulty of efficiently gathering the requisite details in a form suitable for making the decision.

Human computation and social media have opened new avenues for gathering relevant information or opinions in support of a decision-making process. It is now possible to coordinate paid web workers from online labor markets such as Amazon Mechanical Turk and others in a distributed search party for the needed information. However, the strategies that individuals employ when confronted with too much

information—satisficing, information foraging, etc.—are more difficult to apply with a large, distributed group. Consequently, current distributed approaches are inherently wasteful of human time and effort.

This dissertation offers a method for coordinating workers to efficiently enter the inputs for spreadsheet decision models. As a basis for developing and understanding the idea, I developed AskSheet, a system that uses decision models represented as spreadsheets. The user provides a spreadsheet model of a decision, the formulas of which are analyzed to calculate the value of information for each of the decision inputs. With that, it is able to prioritize the inputs and make the decision input acquisition process more frugal. In doing so, it trades machine capacity for analyzing the model for a reduction in the cost and burden to the humans providing the needed information.

CROWDSOURCING DECISION SUPPORT: FRUGAL HUMAN COMPUTATION FOR EFFICIENT DECISION INPUT ACQUISITION

By

Alexander J. Quinn

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2014

Advisory Committee:

Professor Benjamin B. Bederson, Chair
Associate Professor Hal Daumé III
Professor Jeffrey S. Foster
Associate Professor Atif Memon
Professor Ben Shneiderman
Professor Philip S. Resnik, Dean's Representative

Substantial portions of this document have been adapted from the following published work in accordance with the ACM copyright policy.

Quinn, A. J. & Bederson, B. B.. (2014). AskSheet: Efficient Human Computation for Decision Making with Spreadsheets. In *Proceedings of the 2014 ACM Conference on Computer Supported Cooperative Work - CSCW '14*. New York: ACM Press.

Dedication

To my surrogate father, Will Shopes, whose decision 28 years ago to offer tutoring in programming to an 8-year-old he had never met would set the course of my life.

Acknowledgements

This work would not have been possible without the enduring support and generosity of a number of people around me. I simply could not have finished this without them.

My mentor, Prof. Ben Bederson, has granted me the latitude to run with ideas—even some crazy ones—while always giving me his encouragement, frank advice, flexibility, and his time so that I could have the greatest possible chance of success. Prof. Bederson has been a patient teacher, a staunch ally, and a good friend.

I am grateful to my other committee members—Prof. Atif Memon, Prof. Ben Shneiderman, Prof. Hal Daumé III, Prof. Jeff Foster—for donating their time to help make this document as strong as it can be. Each has supported me in a variety of ways beyond this dissertation, as well.

All of the people who comprise the Human-Computer Interaction Lab (HCIL) have been an inextricable part of everything I love about life and work throughout my time in graduate school.

This dissertation is a direct reflection of many in-depth conversations with current and former faculty and colleagues in the HCIL, the Department of Computer Science, and beyond, including Dr. Nicholas Chen, Dr. Chang Hu, Prof. Lise Getoor, Prof. Phillip Leslie, Piotr Mardziel, Jay Pujara, Dr. Yaron Shlomi, Keith Walker, Prof. Tom Yeh, Prof. Ginger Zhe, and Ben Zou.

None of my work could be accomplished if not for the efforts of more than 300 workers on Mechanical Turk who have done my tasks in earnest and sent me feedback over the years, as well as the twenty-four anonymous participants in my field studies.

Finally, my wife (Akemi Quinn) and daughter (Alice Quinn) have both given me constant comfort, love, and logistical support to allow me to follow my aspirations.

Table of Contents

Chapter 1. Introduction

When faced with important decisions—where the difference between good and great has significant consequences—businesses, governments, and even some consumers use models and systematic decision processes. These tools ground their choices in facts and reason, and set the stage for positive outcomes. For example, in my former job with Nordstrom, a major fashion retailer, I built spreadsheet-based decision models as part of a large team tasked with deciding how many items to carry in each department (to keep customers satisfied without inflicting choice overload), how many units of each item to buy at a given time (to avoid running out of any item while minimizing loss from over-buying), and so on. The quality of our decisions had a substantial impact on the company's profits. On a smaller scale, everyday purchasing decisions can sometimes require countless hours researching options and details online. The reader has undoubtedly had to make complex decisions. This might have involved elaborate data-driven models and a large committee, or just a "gut" feeling after looking over the facts.

Regardless of the method or how many people are involved in the process, unless we already know everything about the situation, deciding can entail considerable effort to understand the problem, gather relevant information, and synthesize it to infer which course of action would maximize the value of the end result—or at least maximize the probability of an *acceptable* result. As Simon described in *Theories of Bounded Rationality* (Simon, 1972), our cognitive deficiencies and limited view of the world eventually push us to accept compromises leading to suboptimal outcomes, sacrificing a measure of rationality as a pragmatic coping mechanism. This dilemma gives rise to

satisficing, the strategy of optimizing the outcome offset by the cost of deciding (Simon, 1956). To be clear, satisficing is a compromise—accepting a suboptimal result in order to mitigate the burden of making an ideal decision.

Solving this dilemma will require devising better ways for decision-makers to manage the process so that they can make good decisions with less compromise. For some tasks, decision support tools can help by guiding decision-makers through a method that suits the task. Often, this involves creating a model describing the requirements and rationale of the decision.

An entire industry is devoted to producing decision support tools, mainly for business and government use, as well as some for consumers. However, spreadsheets are a common choice, especially for ad hoc decision analysis and prototyping of new methods. Many courses on decision support systems are taught based on the use of spreadsheets (Power & Sharda, 2007). Spreadsheets can be used for a wide variety of decisions (Ragsdale, 2014). In fact, even sophisticated programmers commonly use spreadsheets for mission-critical applications (Grossman, Mehrota, & Özluk, 2007). Due to their versatility and relatively wide use, I have adopted spreadsheets as the basis for the work presented in this dissertation.

A common pattern is for an individual analyst to formulate the problem, create a model, fill in the requisite information, develop a solution, and test the solution using the spreadsheet (Figure 1). The cells in the spreadsheet generally contain a mixture of inputs (raw data values) and formulas that calculate some analysis of the inputs. When the inputs are not readily available, the analyst may solicit help from others within the

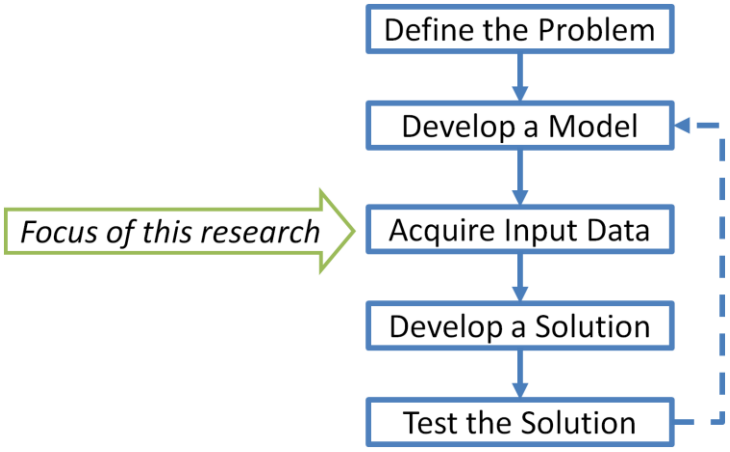


Figure 1. This dissertation aims to improve the input data acquisition step of the decision modeling process. *This diagram was adapted from a textbook on Managerial Decision Modeling (Balakrishnan, Render, & Stair, 2006).* Latter steps relating to analysis and implementation of the results were omitted.

organization or beyond, to search the web for specific information or enter detailed subjective judgments, as needed to support the decision.

Part of the process of decision modeling entails acquiring the input data for the model. This is the focus of my dissertation. For example, selecting a location for a company headquarters would likely entail gathering data about each of the candidate cities, possibly including details specific to the organization (e.g., presence of suppliers of specialized materials needed by the company). For a complex scenario with many choices and many attributes of interest, the amount of requisite data can be quite large, requiring considerable human effort to gather.

If the amount of data required exceeds what an individual is willing or able to gather alone, then the task can be split among a group of people working in different locations. This has the advantage of distributing the possibly tedious task of searching for information. In addition, some models may depend on specialized knowledge held by

only certain people. Distributing the process will enable participation by people with different perspectives or areas of knowledge.

1.1. Problem

The problem addressed by this dissertation is that acquisition of input data by distributed groups of humans is inherently inefficient because (a) distributed groups lack a mechanism for judging which information is still needed; and (b) overhead is incurred in order to integrate the contributions in the decision model.

Unlike an individual gathering information for a smaller decision, groups have no mechanism for judging which information is still needed. An individual gathering information for a smaller decision can use what has been learned so far to identify which further information is most likely to affect the final result (i.e., features of interest) and which is most likely inconsequential to the decision (i.e., minor features of an option that appears unlikely to become the final choice). In contrast, groups have no central control (i.e., a single brain), and thus cannot optimize the process in this way. Thus, a group would waste a lot of effort looking at information that is inconsequential to the final outcome.

Consider the aforementioned problem of selecting a location for a company headquarters. Suppose each of the twenty-five members of a search committee is responsible for researching one of the twenty-five cities under consideration. Each member is asked to find forty attributes of interest to the company for their assigned city. Collectively, the committee would need to look at all information about all of the options—1000 data points in all. This is an inefficient use of the committee members' time. In

contrast, an individual doing the same task at a smaller scale could likely eliminate many of the options quickly based on partial information.

The other part of the problem is integrating the contributions in the decision model. Once the 1000 data points have been collected, they must be analyzed. The information will be entered in a spreadsheet model that will score each city according to some agreed-upon criteria and select the city that most closely matches these criteria. The spreadsheet also leaves a transparent rationale that can be easily communicated to other stakeholders and audited later, if necessary. The model was created primarily by one of the committee members, but now the 1000 data points must be entered. The other members do not have direct access to the spreadsheet file. Even if the committee had used a web-accessible spreadsheet (i.e., Google Sheets[1], Microsoft SharePoint[2], etc.), entering such a complex data set would be vulnerable to data entry errors. Therefore, an additional process will need to be devised to integrate the data points in the decision model, incurring overhead.

1.2. Objective

The objective of this dissertation is to develop an efficient method of acquiring decision input data that can be directly integrated in a decision model. My focus is on offloading the burden of input acquisition to paid workers hired through online labor markets, such as Amazon Mechanical Turk.

---

1 Google Sheets – http://spreadsheets.google.com

2 Microsoft SharePoint – http://sharepoint.microsoft.com

For purposes of this dissertation, I define decisions as not only conventional decisions such as choosing a graduate school to attend or a city to locate a new company headquarters in, but also group processes such as deciding which conference submissions to accept for publication, finding a mutually convenient meeting time, evaluating graduate admission applications, planning a multi-family vacation, or selecting a job applicant to hire.

In the context of the broader decision-making process (Figure 1), this relates exclusively to the input acquisition step. Decision science and decision support tools comprise a mature field of work spanning disciplines as diverse as business, cognitive psychology, behavioral economics, computer science, and civil engineering. A wide range of tools, methods, and theories have been developed and studied over the past century and beyond (Buchanan & O Connell, 2006). These are described in *Related Work* (page 24). This dissertation does not aim to develop any new theory or method for decision making, or advance the fields of decision theory or decision support in any way, other than by developing a more effective method of input acquisition.

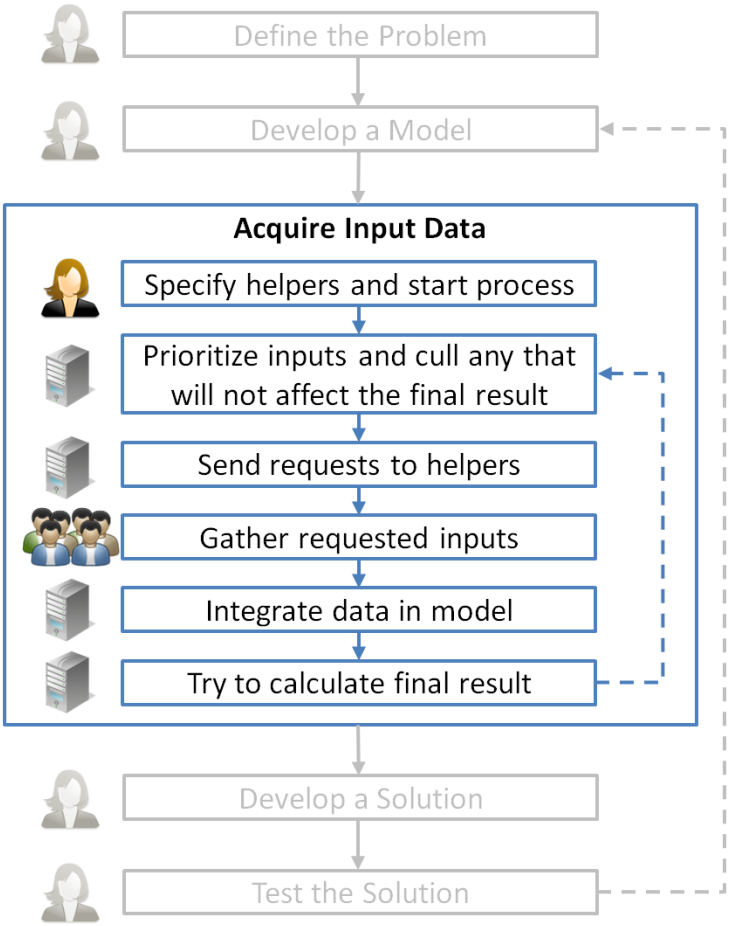


Figure 2. This method is a means of accomplishing the input data acquisition step from the process in Figure 1. *This figure is adapted from* (Balakrishnan et al., 2006).

1.3. Solution

The solution I have developed integrates the decision model into a system for collecting decision inputs from paid crowd workers in a way that leverages the formulas in the user's model to prioritize inputs by the likelihood that they will be needed to compute the final result, and cull those that are not. The system is primarily designed for users who are fluent with using spreadsheet formulas to model decision problems, though I will also demonstrate how it can be used by users without knowledge of spreadsheets using an alternate interface. (That interface is discussed in section 3.6 on page 67.)

This section will give a high-level view of the process, which is summarized in Figure 2. The next section will give a concrete example, based on the system I developed.

To begin the process, the user (decision-maker) starts by defining the problem and developing a model. In the model, the user will enter special *request* formulas in cells where input data is needed. The parameters to the formulas include the range of values that is expected for each input and a cost measure indicating the relative burden of collecting each. Somewhere in the model, there must be a formula (or formulas) that calculates a final result (e.g., a formula that returns the name of the city to which the company should locate its headquarters, according to the model). I call this a *root* cell because generally, it is a root of the dependency hierarchy within the spreadsheet. It depends on other cells, including request formulas, but no other cell in the spreadsheet depends on it. The user will also specify who should help, such as web workers, specific co-workers, vendors, etc.

The system will prioritize the inputs so that the ones that are most likely to affect the final result are requested first. The system will then send requests for input data to the designated helpers, starting with the highest priority inputs first. As input data is received, it is used to populate the inputs in the decision model, and the prioritization is updated.

1.3.1. Frugal human computation

A key motivating idea is that when computers make requests to humans as a part of a computation—a strategy called *human computation*—the requests should be minimized. (Human computation is discussed in Section 2.1 on page 25.) Since CPU time is almost always less expensive and more plentiful than human time and attention, it

makes sense to optimize the overarching computation to use human capacity more frugally. We assign a cost to each request and then try to minimize the total cost incurred in the course of performing the computation.

In this case, the cost is the relative burden of gathering a given input, as estimated by the user who created the model. The overarching computation is the calculation of the root cells. This project uses frugal human computation in two ways: culling inputs and prioritizing inputs.

1.4. AskSheet (preview)

In this section, I will give a brief overview of AskSheet, a system that I built to study this method. Although this is logically a part of latter sections of this dissertation, I am describing it here to give context to the sections that follow.

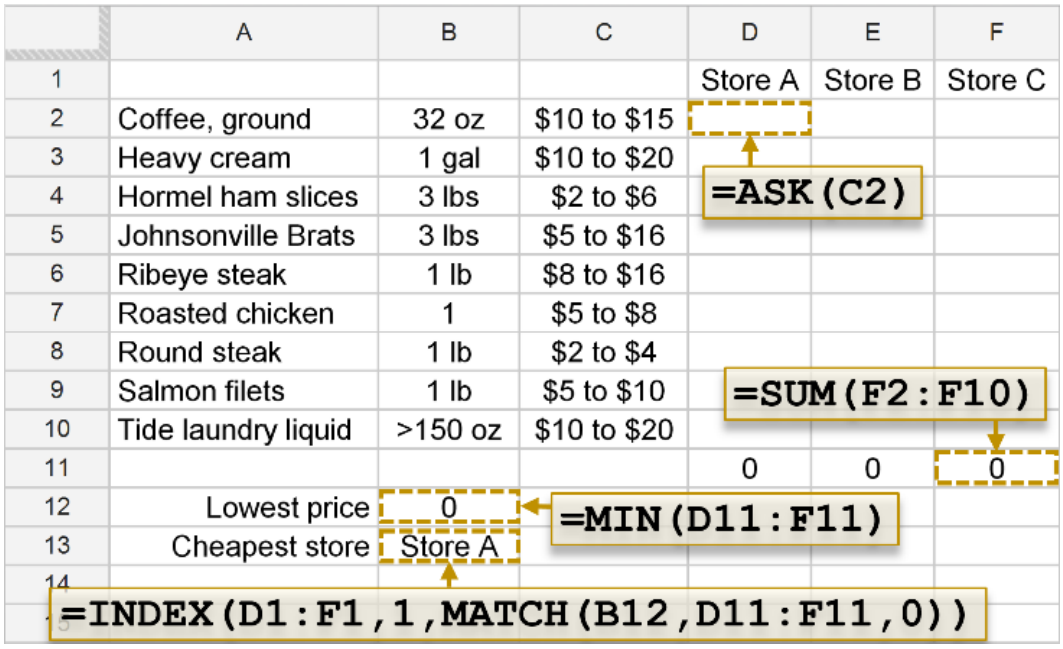
I will explain the use of AskSheet by way of a simplified illustration.

1.4.1. 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 grocery items, the names of the stores, and some formulas. (For compactness, I will use only 9 items and 3 stores.)

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. The resulting model is shown in Figure 3.



	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			
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			
10	Tide laundry liquid	>150 oz	$10 to $20			
11				0	0	0
12	Lowest price	0				
13	Cheapest store	Store A				
14						

Figure 3. In this example, a fictional user needs to determine which of three grocery stores will have the lowest total price for each of the 9 grocery items listed.

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 20 (page 63).

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 (Figure 4). The "Prioritization" slider controls how many inputs to include in each batch.

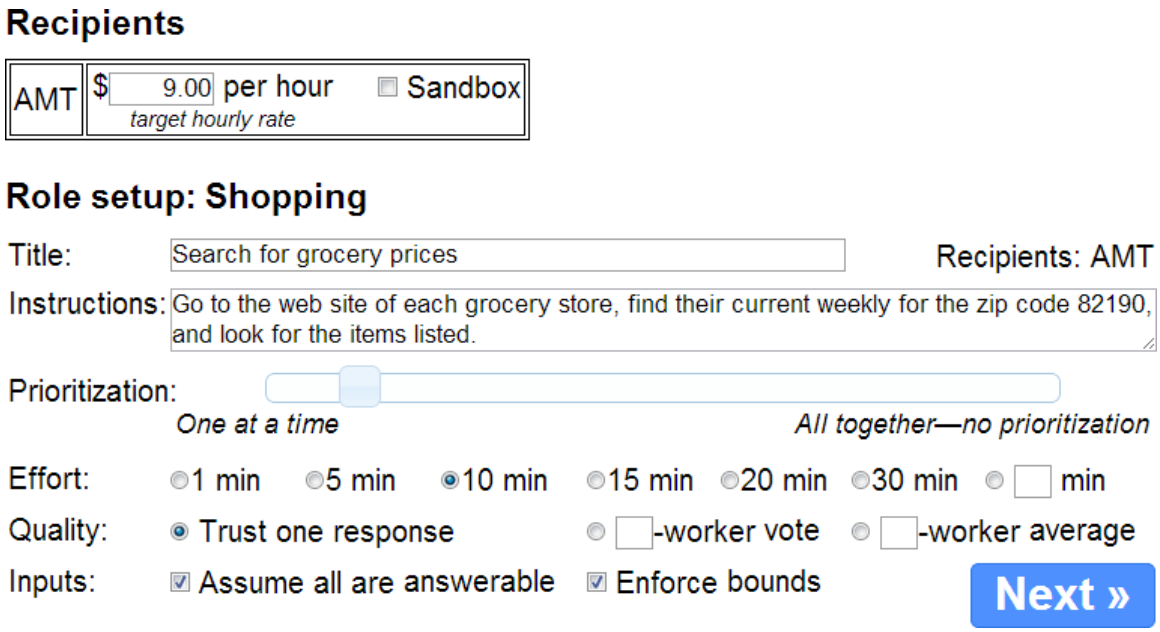


Figure 4. The setup screen is where users enter instructions to the workers, as well as settings that affect the price paid and the structure of the tasks.

Based on the contents and structure of the spreadsheet, AskSheet automatically generates a form for entering data (Figure 5). 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).

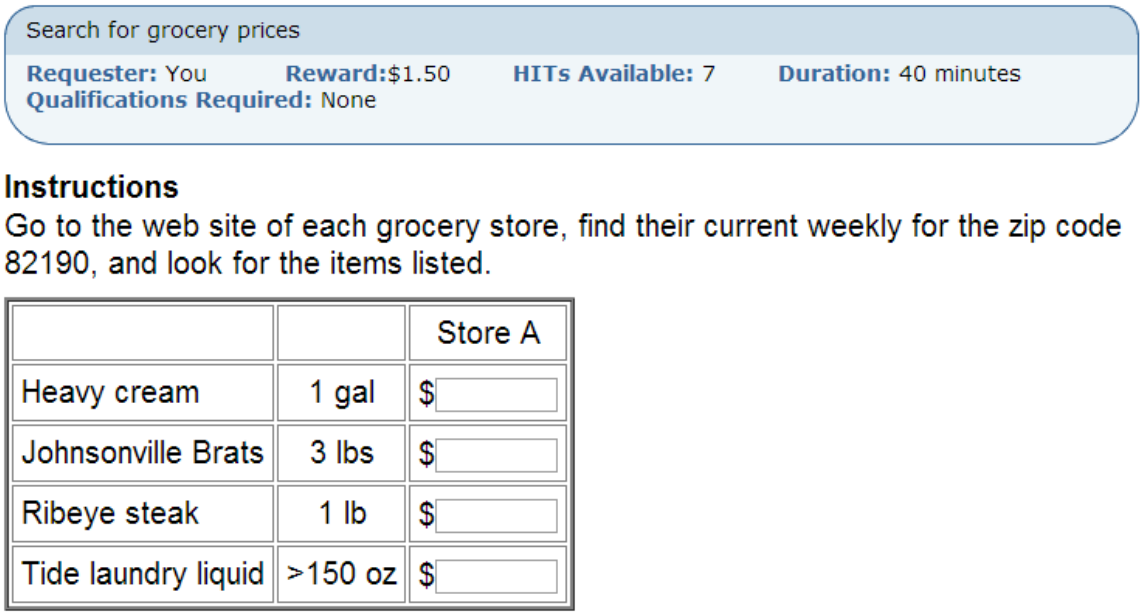


Figure 5. This form was automatically generated by AskSheet from the spreadsheet model in Figure 3.

AskSheet posts the tasks and manages the entire process. Andrew can monitor the progress from a dashboard (Figure 6). 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}				

Figure 6. This is the state of the grocery model before any inputs have been acquired. The dark green cells have been deemed the highest priority. For visual contrast, this screen only distinguishes between substantial differences in utility. Internally, utilities are continuous.

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, it displays the result of that formula calculated with the inputs given by the workers.

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 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). The final state is shown in Figure 7.

			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				

Figure 7. This is the state of the grocery model after enough inputs have been received to calculate a conclusive result (Store C).

A primary benefit of using AskSheet is that it does not need to fulfill all of the requests. It is able to obtain a 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 is possible that the benefit of the information gathered by the crowd workers will not be enough to justify the cost of hiring them. As with any time-saving tool, the user must make that judgment before deciding to use it. Also, since tasks typically ask for more than one input at a time, AskSheet might sometimes gather slightly more inputs than necessary.

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.

1.5. Motivating example

Early inspiration for AskSheet came from a decision spreadsheet that I used when shopping for a smartphone (Figure 8). I include it here because it illustrates both the value of such modeling, even for small decisions, and the intuition behind the optimizations used in AskSheet.

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2			**Atrix**			**Thunderbolt**			**iPhone 4**			Weights
4	Battery, web browsing		5.25 hrs	7		prob bad (Evo got 3.68 hrs)	0		6.65 hrs	10		25
5	Camera quality		bad, 5MP	0		good, HTC, 8MP	5		excellent, 5MP	10		20
6	GPS quality		Motorola	10		HTC	0		Apple	10		10
7	Speed, network, 3G		1410 kbps, AT&T	5		877 kbps, Verizon	0		1410 kbps, AT&T	10		10
8	Calling plan		Rollover, AT&T	10		Expensive, Verizon	0		Rollover, AT&T	10		10
9	Metro tunnel service		no	0		Service in tunnel	10		no	0		10
10	Speed, device, web		excellent, R=53576	10		good? (myTouch R=35503)	0		good, R=31375	0		10
11	Speed, device, general		Tegra, 1GB, Q=2600	10		MSM8655, 768MB, Q=1950	0		A4	0		10
12	Screen size		4.0"	5		4.3"			3.5"	0		10
13	Speed, network, 4G		HSPA+, 1.5~7 Mbps	0		LTE, 5~12 Mbps			n/a	0		10
14	Screen durability		Gorilla glass	10		base			high breakage rates	0		5
15	Battery, talk time		9 hrs	10		6 hrs, extendable			7 hrs	5		5
16	Screen quality		PenTile, TFT LCD	0		TFT, prob good (like Evo)			IPS	10		5
17	Kickstand		no	0		yes, new sturdy design			no	0		5
18	Max storage		48 GB	10		40 GB			32 GB	0		5
19	UI niceties		Motoblur	0		Sense 2			IOS 4	0		5
20	Concurrent talk+data		Concurrent T+D	10		experimental			Concurrent T+D	10		3
21	USB flash drive capable		USB drive via adapter	10		no			no	0		3
22	Boot time		50 secs	0		8 secs, "fast boot" feature			28 secs	4		2
23	Service provider trust		AT&T	10		Verizon			AT&T	0		2
24	Compact		118 x 64 x 11 mm	10		122 x 66 x 13 mm			114 x 59 x 9 mm	10		2
25	National affilliation		USA, Motorola			Taiwan, HTC			USA, Apple			1
26	Screen resolution		960x540, 0.52MP, 275ppi			800x480, 0.38MP 215ppi			960x640, 0.61MP, 326 ppi			1
27	Works in Japan		International			no			International			1
28	Lightweight		4.76 oz			5.78 oz			4.83 oz			1
30		Default		**925**			**200**			**883**		
31	Worst case	0		925			200			883		
32	Average case	5		945			530			903		
33	Best case	10		965			860			923		
34			WINNER			LOSER			LOSER			
35				curr			curr			curr		
36												
37	Margin of winning			2			-765			-82		
38	Margin of loss			-82			65			2		
39												
40												
41	**Best choice:**											
42	Atrix											

Figure 8. Manual simulation of minimum amount of data. Empty green cells illustrate values that were not needed. No matter what score is chosen for these cells (within the expected range of 0-10), the end result is guaranteed to be the same. All empty shaded cells and "WINNER" are in green. "LOSER" is in red.

After some preliminary research to narrow the field to three top contenders, I created a model comparing them on 25 attributes. Populating the 75 data points was a time-consuming process.

Some of the attributes were readily available technical specifications, such as the battery life (hours of talk time) and screen size (inches). Others were more obscure attributes of interest to the individual buyer, such as boot time (seconds) and screen glass durability. All attributes were converted to a desirability score: 0 for least desirable, 10 for most desirable (among these choices).

Starting with the original weights and scores, I substituted 0 in place of some scores on the winner (Motorola Atrix) to simulate the worst case, if I did not already have that data. I substituted 10 in place of several scores on the two losers (Apple iPhone 4 and HTC Thunderbolt) to simulate the best case. If the best case of the losers cannot beat the worst case for the current winner, the decision is final and data acquisition can be terminated.

Using manual trial and error, I found that I only needed **50** of the **75** data points to guarantee the same end result. In other words, AskSheet would save having to gather **33%** of the data points.

The actual savings depends heavily on the specific weights. If all weights were the same, the savings would be less. To test one other set of weights, I repeated the exercise substituting uniformly distributed random numbers for the weights. They were assigned to attributes in an order to guarantee the same relative ordering of the attributes. I found that I could guarantee the same result with only 45 of the 75 data points. Thus, in

this scenario with uniformly distributed random weights, AskSheet would save having to gather 40% of the data points.

As a side effect of this exercise, one can see exactly which attribute made the difference: compactness. This may initially seem unexpected, since the final winner and the runner up both got 10 for that attribute. One might expect the two 10s to cancel each other out. The important point is that this is a comparison between the worst case of the winner versus the best case of the runner up. Adding in the rating for compactness improved the Atrix's worst case, without improving the iPhone's best case. If they had both gotten 0 for compactness, then more information would be required to guarantee a consistent end result.

Although these reductions would be attractive in themselves, I expect the savings to be much greater given a much larger, initially unfiltered field of choices. This would save the user the effort of initial filtering and also reduce opportunities for suboptimal decisions. For the above smartphone decision model, I had already identified three alternatives that were comparable. That required a non-trivial effort to survey the roughly 25 options currently available in the US. Had I (or some user) started with an unfiltered list, the model would have 625 inputs. Many could be "eliminated" quickly, since their scores would be far lower than the top contenders.

1.6. Listing the alternatives

The smartphone example highlights the problem of gathering the list of alternatives, an important aspect of input acquisition for decision making. This is a complementary problem, but requires a different type of system design from AskSheet.

AskSheet assumes a skeleton spreadsheet with a fixed set of row and column headers. Based on the headers, the intended content of any given cell in the spreadsheet is determined (e.g., gas mileage of Subaru Forester in cell G13). In contrast, when making a list, the exact size of the list is not known in advance. Furthermore, the items are likely to be found in arbitrary order. For example, it would be meaningless to ask crowd workers to enter the 17th car available for sale.

In an important, but separate effort from AskSheet, I developed a system for delegating the task of creating a list of alternatives—or any other list generated from internet research—to paid crowd workers. Instead of integrating Relay with AskSheet, I chose to develop it as a standalone application to demonstrate how this aspect of decision input acquisition could be addressed, as a guide to future researchers who may want to connect the ideas.

The system, called Relay, is described in Chapter 5 (page 92). It designed to encourage a diverse set of list items while setting the conditions to minimize duplication. The user of Relay enters a description of the type of items to gather. The system posts tasks to Mechanical Turk that ask workers to think of search queries or other sources for finding list items, and then enter some list items into a web form.

If Relay merely asked workers to enter list items, it is quite possible that many workers would do the same web search and enter many of the same items. Therefore, Relay shows workers the list of sources that have been checked so far, and allows them to pick up where previous workers left off.

Since some duplication is inevitable, Relay employs an aggressive autosuggest feature, such that when a worker begins to enter a list item, the system searches other items

entered so far and offers to fill in details from existing items. This saves the worker time while increasing the likelihood that that any duplicates will be spelled exactly the same so that they can be trivially deduped by the software.

Relay does not work directly with spreadsheets. Instead, the results are returned in the form of a web page. However, a user could easily copy and paste the list items from the web page into a spreadsheet for use with AskSheet.

List-making is an important component of decision-making, so developing Relay filled an important gap with respect to the user-facing aspects of this dissertation. Nevertheless, the primary focus is on coordinating crowd workers to gather the inputs for spreadsheet decision models.

1.7. Notation

AskSheet is based on spreadsheets, but the core idea is not actually dependent on that particular notation. Spreadsheets were chosen as the initial implementation medium just as one would choose a programming language for any other project. In fact, the same decision models could be expressed in other programming languages or notations, and the methods and algorithms could be adapted accordingly. Nevertheless, spreadsheets have some features that make them especially convenient for applying this method:

- **guaranteed to halt:** Because spreadsheets do not support recursion or unbounded loops, every spreadsheet program is guaranteed to halt (Sipser, 1997). In other words, the calculation of the spreadsheet is guaranteed not to go into an infinite loop. This allows the system to calculate, a priori, the set of possible outputs for

every formula, which in turn is an important part of the optimizations that make this strategy powerful.

- **no side effects:** Evaluation of a cell in a spreadsheet does not have any side effects. Thus, the results will be the same no matter what order the cells are evaluated in, and lazy evaluation can be employed without affecting the results.
- **arbitrary parameter evaluation order:** There is no stated guarantee as to the order in which parameters to functions will be evaluated. My optimizations exploit this extensively to reduce the overall number of inputs required to calculate a final result.
- **array-oriented constructs:** Many functions used in spreadsheet formulas take arrays as parameters (i.e., `=MAX(A1:A20)`, `=COUNTIF(A1:A20, ">30")`, etc.). These functions further enable opportunities for the prioritization algorithms to manipulate evaluation order.

Spreadsheets support all of these features as part of their fundamental programming model. However, I can imagine alternative implementations of the basic idea that would use other programming models. For example, it might use a restricted subset of Python or JavaScript that did not allow recursion or unbounded loops (i.e., foo(foo(a,b,c)), while, do, etc.). The only firm requirements are that programs in the language be guaranteed to halt, and that it have some opportunities for manipulating the order in which certain operations are executed.

1.8. Contributions

The primary contribution of this dissertation is a method for *coordinating workers* to efficiently enter data to be supplied as the inputs to a spreadsheet decision model.

I developed AskSheet, a system that coordinates paid crowd workers to fill in the inputs (blank cells) in a decision spreadsheet. To prioritize the inputs, it uses a static analysis of user-provided spreadsheet formulas to calculate the value of information of input. The typical application is to acquire the attribute values for a set of alternatives. (Chapter 3)

I conducted a set of field studies to measure the effectiveness and applicability of AskSheet. These include three field trials with three replications each, a survey of decision problems, and a study of users modeling decision problems. (Chapter 4)

I developed Relay, a system that coordinates crowd workers to acquire a list of alternatives meeting some given criteria. (Chapter 5)

1.8.1. Thesis statement

> The thesis of this dissertation is that there is a class of data-driven decision problems that can modeled with spreadsheet formulas, such that the syntactic structure of the formulas can be leveraged to decompose the process of gathering the inputs into parallel workflows by paid crowd workers, and the inputs can be automatically prioritized so that a decision result can be efficiently obtained with only a subset of the decision inputs.

1.8.2. Generalizable knowledge

The generalizable knowledge created by this dissertation includes (a) the prioritization method used by AskSheet, (b) the framework for decomposing a decision spreadsheet into independent tasks that are efficient to perform, (c) the interaction method

used by Relay to decompose the curation of a list of undetermined length into independent tasks that are efficient to perform, and (d) an exposition of the compatibilities and incompatibilities of using paid crowd labor in a microtask format to acquire inputs for decision models.

1.9. Non-contributions

1.9.1. Data quality

Data quality is mostly orthogonal to the core research problem of coordinating the input acquisition process. Nevertheless, it is important to making the results useful, so I have used established methods for managing quality with Mechanical Turk, including input validation and mechanisms for aggregating multiple judgments either by consensus vote or averaging (depending on the type of input).

The aggregation features that AskSheet supports solicit an arbitrary number of responses in order to receive a single value. Research done by others uses elaborate methods to determine the right number of judgments to request, determine the reason for errors (e.g., carelessness, random clicking, ambiguity in the question, etc.), or manage which workers seem to be most reliable (Dai, Weld, & others, 2010; Ipeirotis, Provost, & Wang, 2010; Sheng & Provost, 2008). Quality control in human computation and crowdsourcing is an active research area in itself, and not an area in which I claim to have made any significant contribution. My focus is on coordinating independent workers to support a decision process, and prioritizing which questions to ask in order to avoid requesting inputs that are not needed. This challenge is orthogonal to that of ensuring that

any given input contains an accurate value. (This is discussed further in section 4.1 on page 70)

1.9.2. Usability

The focus of this work is on coordinating *crowd workers* to efficiently gather and enter the inputs to spreadsheet decision models, which are taken as input to the system. As such, the usability of the interface for creating and managing models in the system is not a primary concern. AskSheet uses spreadsheet models that were created in an unmodified commercial spreadsheet application (Google Sheets).

The interface used to control AskSheet could conceivably be adapted for use by general users. In fact, as part of a field study for this dissertation, I did create a feature within AskSheet that allows users who are not familiar with spreadsheet formulas to create decision models that can work with AskSheet. (See section 3.6 on page 67 and section 4.4 on page 86.) However, that tool was designed solely for the purpose of the study. Usability for the requester (decision maker) is not an intended contribution.

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.

Chapter 2. Related work

This project is essentially an application of *human computation and crowdsourcing* combined with some *probabilistic optimization*, to advance the problem of *decision-making*.

Over the course of my dissertation work, I have had the privilege of discussing this project with many colleagues from various areas of computer science, as well as other

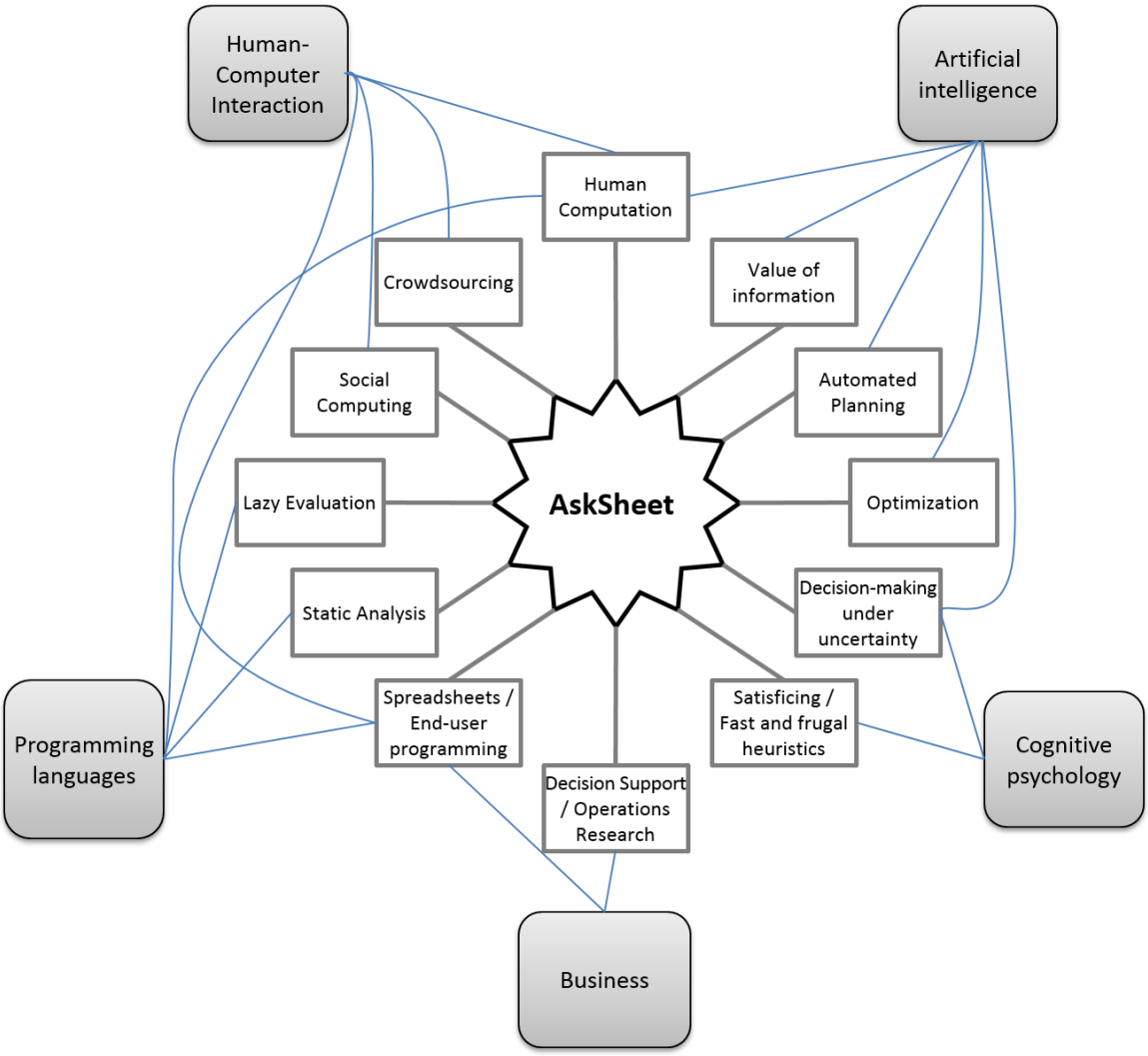


Figure 9. The topic of this dissertation intersects with an especially wide range of areas.

disciplines. They had widely varying perspectives on what topics and related work would be relevant. These are summarized in Figure 9, along with the areas of study that each topic relates to, which often overlap. Even this omits work from information retrieval (i.e., models of information foraging) and database systems (i.e., analogies with probabilistic selectivity estimation).

My literature review is focused on a few guiding questions:

- What related projects exist?
- What are the capabilities and limitations of human computation and crowdsourcing?
- What is the relationship of spreadsheets to programming languages?
- How are spreadsheets used?
- What are other methods of modeling the kinds of decisions this project pertains to?

2.1. Human computation and crowdsourcing

Human computation and crowdsourcing are closely related strategies for accomplishing work that cannot be done with computers alone. Human computation, defined as "… *a paradigm for utilizing human processing power to solve problems that computers cannot yet solve"* (von Ahn, 2005), has been used for such tasks as image tagging (Von Ahn & Dabbish, 2004), text editing (Bernstein et al., 2010), music genre classification (Law, Von Ahn, Dannenberg, & Crawford, 2007), and translation (Hu, Bederson, Resnik, & Kronrod, 2011). Essentially, it is the strategy of replacing computers with humans. Crowdsourcing is different in that it replaces human workers with members of public at large. Jeff Howe, who coined the term, defines crowdsourcing

as "*the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call.*" (Howe, 2006). The key difference is that human computation deals primarily with prescribed work processes, as opposed to letting workers solve a problem ad hoc. Also, with human computation, the tasks are generally things one would like a computer to be able to do—i.e., translating text—as opposed to fundamentally human tasks—i.e., writing a personal letter from scratch (Quinn & Bederson, 2011).

Since this project involves a prescribed process, everything here can be considered human computation. However, some aspects may involve having web workers do jobs that in-person labor would have otherwise performed. Such tasks can be considered examples of both.

PeopleCloud is a system that was tested at IBM Research for harnessing large numbers of in-house workers to do tasks in the spirit of crowdsourcing (Vukovic, Lopez, & Laredo, 2010). The system provided mechanisms for allocating tasks to available workers who possess the needed capabilities. The latter mechanisms could offer a useful starting point for integrating AskSheet's support for known individuals and groups within an enterprise.

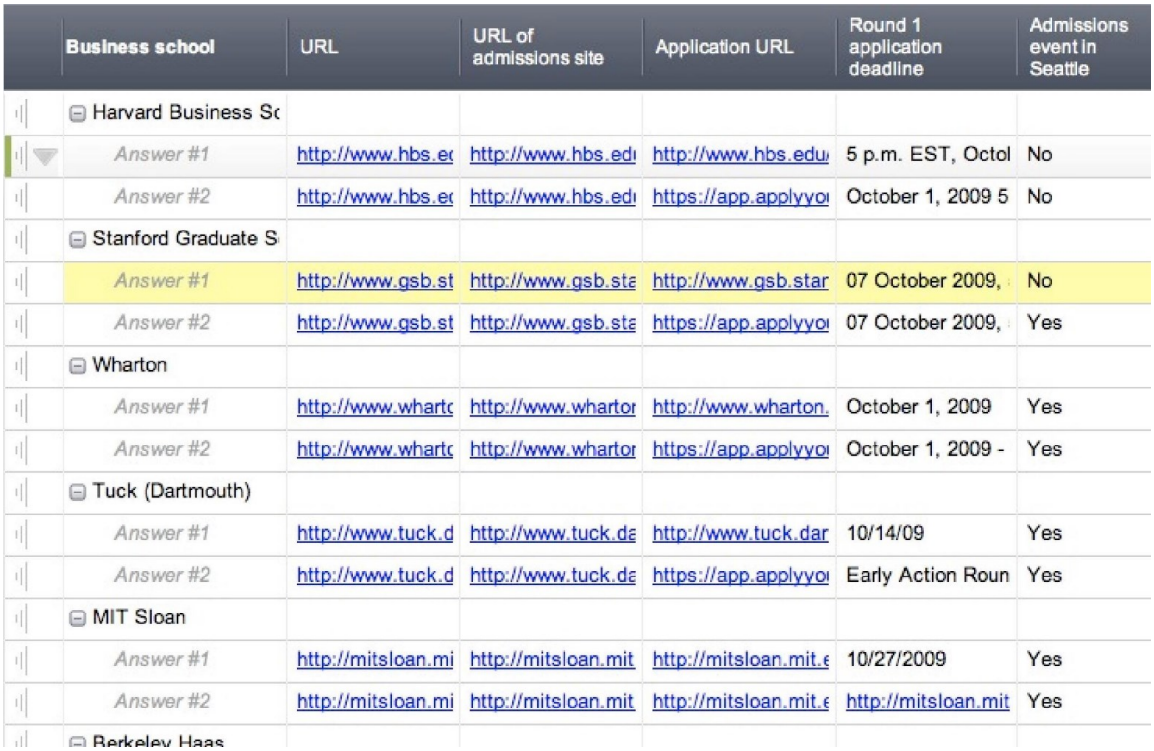
A few related projects in this area stand out as being especially relevant to AskSheet.

2.1.1. SmartSheet : Mechanical Turk from spreadsheets

Currently, the most prominent use of crowdsourcing to populate data in spreadsheets is SmartSheet, a commercial service. It provides an web-based spreadsheet-like application and offers the ability to hire paid web workers to fill in data in the sheets

(Figure 10). For example, given a spreadsheet about companies, SmartSheet works with paid crowdsourcing marketplaces (i.e., Mechanical Turk) to fill in other information about the companies, such as the mailing address, name of the company president, or other information which the web workers can find by searching the web (Frei, 2009).

The most important distinction with AskSheet is that Smartsheet does not have any facility for optimizing the use of the labor. Whereas the objective of SmartSheet is to fill the entire grid, AskSheet will collect the least amount of data that will allow it to calculate the result of the key formula(s) in the model. Also, despite the apparent similarity of the tabular formats, the capabilities are different. AskSheet is based on a general purpose spreadsheet, whereas SmartSheet is based on a spreadsheet-like application that does not provide general computation.



Business school	URL	URL of admissions site	Application URL	Round 1 application deadline	Admissions event in Seattle
Harvard Business Sc					
Answer #1	http://www.hbs.ec	http://www.hbs.edu	http://www.hbs.edu	5 p.m. EST, Octol	No
Answer #2	http://www.hbs.ec	http://www.hbs.edu	https://app.applyyo	October 1, 2009 5	No
Stanford Graduate S					
Answer #1	http://www.gsb.st	http://www.gsb.sta	http://www.gsb.star	07 October 2009,	No
Answer #2	http://www.gsb.st	http://www.gsb.sta	https://app.applyyo	07 October 2009,	Yes
Wharton					
Answer #1	http://www.wharto	http://www.whartor	http://www.wharton.	October 1, 2009	Yes
Answer #2	http://www.wharto	http://www.whartor	https://app.applyyo	October 1, 2009 -	Yes
Tuck (Dartmouth)					
Answer #1	http://www.tuck.d	http://www.tuck.da	http://www.tuck.dar	10/14/09	Yes
Answer #2	http://www.tuck.d	http://www.tuck.da	https://app.applyyo	Early Action Roun	Yes
MIT Sloan					
Answer #1	http://mitsloan.mi	http://mitsloan.mit	http://mitsloan.mit.e	10/27/2009	Yes
Answer #2	http://mitsloan.mi	http://mitsloan.mit	http://mitsloan.mit.e	http://mitsloan.mit	Yes
Berkeley Haas					

Figure 10. Smartsheet uses a spreadsheet-like interface to specify jobs to be completed using crowd labor. A requester specifies the row and column headers. Information collected by crowd workers is populated into the body cells. *This figure was taken directly from (Agarwal 2009).*

2.1.2. TurKit : Mechanical Turk from JavaScript

TurKit is a toolkit that simplifies the task of programming human computation applications using Amazon Mechanical Turk. Applications are written in JavaScript, and the entire process of posting tasks and collecting results is encapsulated in a single program. If a program is run multiple times, the toolkit ensures that the tasks are not posted multiple times. An example is shown in Figure 11.

Although both TurKit and AskSheet can be thought of as simplified programming models for interacting with web workers, there are a few important distinctions between the two projects.

```
ideas = []
for (var i = 0; i < 5; i++) {
    idea = mturk.prompt(
        "What's fun to see in New York City?
        Ideas so far: " + ideas.join(", "))
    ideas.push(idea)
}
ideas.sort(function (a, b) {
    v = mturk.vote("Which is better?", [a, b])
    return v == a ? -1 : 1
})
```

Figure 11. This TurKit example program is used as an example in Little's paper about TurKit. It shows how to generate five ideas for things to see in New York City, and then have workers sort the list by pairwise voting. *This snippet is quoted verbatim from (Little, Chilton, Goldman, & Miller, 2010b).*

First, AskSheet uses probabilistic optimizations to aggressively pursue opportunities to avoid doing work. TurKit presumably observes the same evaluation model as JavaScript, which allows for some short-circuit evaluation, but much less aggressively. For example, AskSheet will attempt to optimize evaluation of a comparison operator (i.e., `A1<B1`) to avoid having to evaluate one side of it in some cases. This is enabled by the fact that AskSheet requires inputs to be bounded. Beyond this dissertation, I hope that optimizations like the ones in AskSheet will allow different kinds of programming, such as coding an algorithm in a naïve way that would otherwise be cost-prohibitive, and relying on the optimizations to accomplish the same task with less work.

Second, TurKit is designed to work only with Mechanical Turk, whereas AskSheet is designed to work with different kinds of human helpers. This in turn requires specifying relative costs, even for helpers who are not directly paid for doing the tasks. Doing so allows AskSheet to balance the use of different resources (i.e., trade-off between 20 paid tasks versus one request to a co-worker). I believe this will also allow AskSheet

to support different kinds of workflows that require both (i.e., collecting objective data with web workers and expert subjective ratings from co-workers).

Finally, TurKit is designed to support a more general programming model: anything that can be coded in JavaScript. AskSheet is designed around the spreadsheet programming paradigm, which does not allow for recursion or unbounded loops, for example. By supporting these things naturally and cleanly, TurKit allows more advanced workflows, such as sorting.

To summarize, TurKit supports a more general programming model with a homogeneous workforce (only Mechanical Turk), while AskSheet supports a heterogeneous workforce and a narrower programming model, which enables the optimizations.

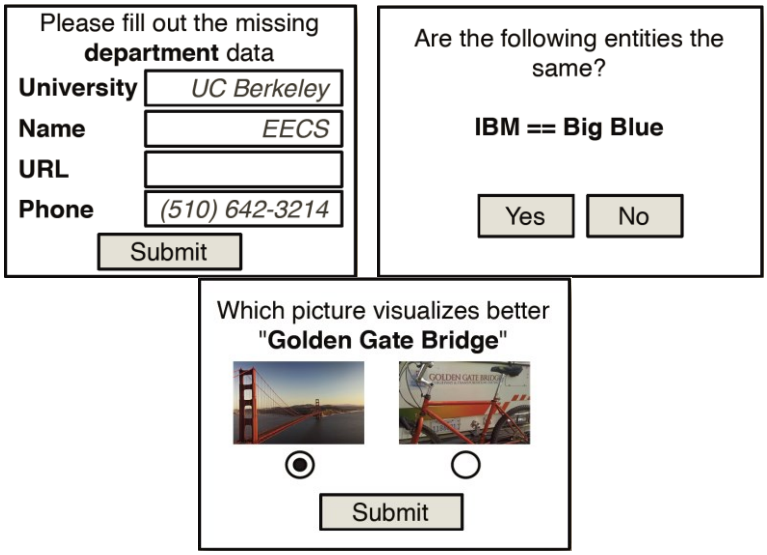
2.1.3. Turkontrol : Decision-theoretic control

Decision theoretic planning methods from artificial intelligence have been leveraged to optimize the use of crowd labor to achieve sufficient degrees of quality for tasks such as image processing (Dai et al., 2010). The system built by Dai et al, called *Turkontrol*, builds on iterative crowd workflows that aim to maximize quality for such tasks where result quality can be highly variable and difficult to measure (Little, Chilton, Goldman, & Miller, 2010a). In contrast, AskSheet's optimizations center on identifying and eliminating tasks that do not need to be done at all, rather than manipulating result quality. In the case of AskSheet, the kinds of tasks I anticipate would be sent to anonymous web workers would be information gathering tasks, where iterative improvement would be less of a factor.

2.1.4. Qurk and CrowdDB

Whereas AskSheet is essentially applying a spreadsheet interface to working with crowdsourcing channels, Qurk (Marcus, Wu, Karger, Madden, & Miller, 2011a, 2011b) and CrowdDB (Franklin, Kossman, Kraska, Ramesh, & Xin, 2011) have applied a database interface. Both projects allow a user to write SQL-like queries to be answered by web workers, even including sort and join operations. The differences lie in their handling of database features, such as foreign keys, and the range of relational operators supported.

Although the objective and mechanisms differ from AskSheet, they are similar in that both use a declarative syntax and a goal-directed process to generate requests to workers. The worker interfaces of the two projects are shown in Figures 12 and 13.

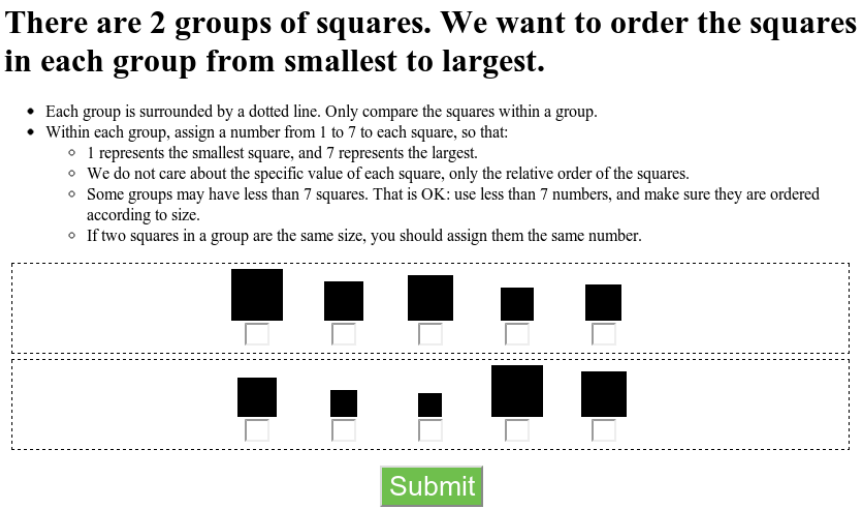


(a) `SELECT` (b) `CROWDEQUAL` (c) `CROWDORDER`

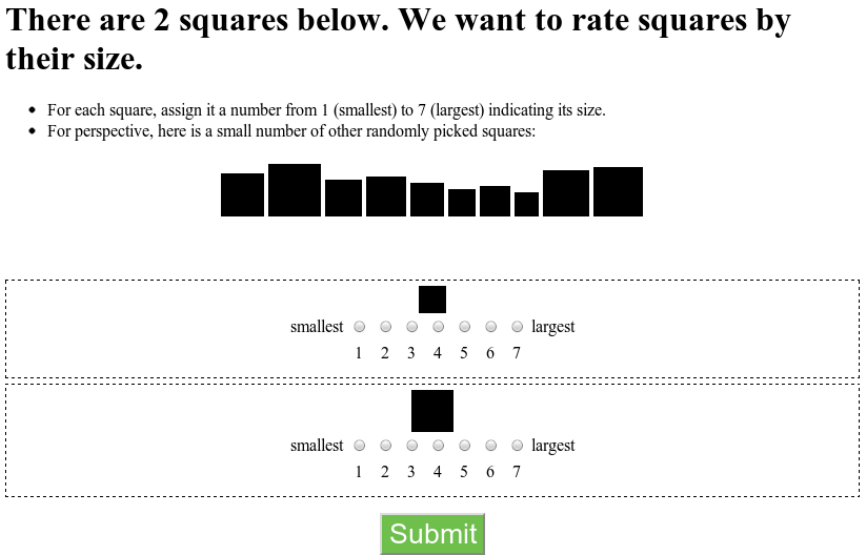
Figure 12. CrowdDB enables the use of SQL syntax to specify queries to crowds. A `SELECT` query will result in asking crowd workers to fetch a specific piece of information. The `CROWDEQUAL` AND `CROWDORDER` operators are used for joins and sorts, respectively. *This figure was taken directly from (Franklin et al., 2011).*

```
SELECT squares.label
FROM squares
ORDER BY squareSorter(img)
TASK squareSorter(field) TYPE Rank:
     SingularName: "square"
     PluralName: "squares"
     OrderDimensionName: "area"
     LeastName: "smallest"
     MostName: "largest"
     Html: "<img src='%s' class=lgImg", tuple[field]
```

(a) Code for a sort operation in Qurk



(b) Comparison Sort interface



(c) Rating Sort interface

Figure 13. Qurk also enables using SQL to specify jobs for crowd workers. It is similar in spirit to CrowdDB, but uses different approaches for managing multiple results from workers, operators and optimizations. *These figures are from (Marcus, et al 2012).*

2.1.5. Efficient human computation

Human computation is a paradigm for computation that delegates to humans parts of the problem that computers cannot solve adequately (Quinn & Bederson 2011; von Ahn 2005). Often, it starts with the assumption that the requester has a set of questions for which answers are desired. Many such systems aim to reduce the number of tasks that workers are asked to perform by adjusting the number of judgments per question (Ipeirotis et al., 2010; Sheng & Provost, 2008), the number of iterations on incremental improvement jobs (Dai et al., 2010; Little et al., 2010a; Sheng & Provost, 2008; Trushkowsky, Kraska, Franklin, & Sarkar, 2013), the strategy for decomposing tasks into subtasks (Bernstein et al., 2010), or by delegating some tasks to machines (Quinn, Bederson, Yeh, & Lin, 2010).

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 (K. T. Chan, King, & Yuen, 2009), 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.

2.2. Spreadsheets

2.2.1. Relationship to programming languages

Although often thought of as a business intelligence tool, spreadsheets can also be considered an end-user programming system—and probably the most widely used in the world. They are accessible for less technical users to get started, and offer a smooth transition from basic models to more powerful ones, such as Monte Carlo simulation and decision trees (Bodily, 1986).

Spreadsheets can be considered a declarative programming language because the *program* (network of spreadsheet formulas and literals) specifies "*what* is to be computed, but not necessarily *how* it is to be computed" (Lloyd, 1994). The "declarative" aspect is important because it characterizes the manner in which spreadsheets are constructed. More specifically, they are an instance of first-order functional programming (Abraham, Burnett, & Erwig, 2008), a subset of declarative programming (Hanus, 1997). In fact, many of the typical pedagogical examples often used to teach functional programming (i.e., Fibonacci sequences, Towers of Hanoi, generation of permutations, etc.) can actually be performed using clever spreadsheet formulas (Casimir, 1992).

2.2.2. Interface construction

In addition to functional programming, spreadsheets were used as the basis for much early work in end-user programming and interface construction tools. Gotfried and Burnett demonstrated the "graphical definitions" technique which pairs gesture and direct manipulation interactions with a spreadsheet-like application to enable the user to specify objects. This was applied to tasks such as creating visualizations of data using custom

shapes (Gottfried & Burnett, 1997). Hudson showed how the behavior of a GUI application could be specified using a spreadsheet-like end-user programming system based on cells and formulas. This allowed typical benefits of spreadsheets—visibility, accessible control constructs, etc.—to be harnessed for other kinds of programming (Hudson 1994). Even earlier, Myers used a spreadsheet-like model as the basis of a graphical tool for specifying user interface constraints, with the goal of integrating it into a larger interface builder (1991). Spreadsheets have also been cited as the inspiration for other work in end-user programming that did not directly involve spreadsheets (Burnett et al., 2001; Lai, Malone, & Yu, 1988).

Alternate representations have also been used for the spreadsheet formulas. Cox and Smedley (1994) created an extension that allows the user to construct the computation using Prograph, a visual programming language.

2.2.3. Collaboration and coordination

One of the earliest distributed spreadsheets was based on the UNIX *sc* spreadsheet, and explored the technical challenges associated with maintaining consistency, even in the face of concurrent edits by networked users (Palmer & Cormack, 1998). Even before that, Nardi and Miller studied the ways in which users were collaborating using standard single-user spreadsheet applications, either through asynchronous contributions, or by working together at a single workstation (1990).

Recently, web-based spreadsheets, such as Google Sheets (http://docs.google.com), Microsoft Excel Web App (http://office.microsoft.com/en-us/web-apps/), and EditGrid (http://www.editgrid.com), have brought spreadsheet-based collaboration into mainstream use. The Forms feature of Google Sheets goes further by integrating an online survey

mechanism with the spreadsheet. Users define a table in the spreadsheet, create a survey form using the provided interface, and send the form to their contacts. Each response to the survey becomes a row of the spreadsheet.

Gilige et al are preparing a design of an enhanced environment for collaboration using spreadsheets, starting with a study of user interface requirements (Ginige, Paolino, Sebillo, Shrodkar, & Vitiello, 2010). Their system design deals with issues such as conflict resolution, visibility, and update notifications (Ginige, Paolino, Sebillo, Tortora, et al., 2010).

Whereas AskSheet is about coordinating complex human workflows using spreadsheets, Fujima (2007) proposed how spreadsheets could be used to define coordination among complex web applications.

2.2.4. Prevalence

Several studies of spreadsheet use and spreadsheet prevalence have been conducted (Brown & Gould, 1987; Y. E. Chan & Storey, 1996; Hendry & Green, 1994; Lawson, Baker, Powell, & Foster-Johnson, 2009; McGill & Klobas, 2005; Nardi & Miller, 1990). However, none of these directly measured the prevalence of the specific style of modeling that my research is based on. There is only indirect evidence, based on studies taken from different populations and at different times.

One such study, conducted by the Spreadsheet Engineering Research Project (SERP) at Dartmouth, surveyed 1,597 spreadsheet users at several organizations between 2005 and 2006 (*SERP (Spreadsheet Engineering Research Project)*, 2006). The vast majority of the respondents self-identify as spreadsheet experts ("extensive experience; some expertise" or "very experienced; high expertise"). They reported creating

spreadsheets using optimization (46.8%), simulation (30.6%), statistical analysis (60.3%), and other techniques (25.9%). The percentages do not sum to 100% because some respondents used multiple techniques.

The SERP data deals with expert use, but gives no insight as to the frequency of spreadsheet use by the general population. In 2003, Microsoft claimed in a press release that there were 400 million users of Excel worldwide (Microsoft Corporation, 2003). However, the statement did not specify whether it was counting active users of Excel or all installations of Microsoft Office, which includes Excel.

US Government data gives some additional insight. Between 1987 and 2003, the US Bureau of Labor Statistics tracked use of spreadsheets and databases by American workers as part of the Current Population Survey (Scaffidi, Shaw, & Myers, 2005; US Bureau of Labor Statistics, 2001, 2003). The results (below) give some insight to the historical use of spreadsheets in the workplace in the United States.

1989	"Do you use spreadsheets?"	10%
1997	"Do you use spreadsheets?"	30%
2001	"Do you use spreadsheets or databases?"	60%
2003	"Do you use spreadsheets or databases?"	64%

These data do not directly address the prevalence of spreadsheet *modeling* (i.e., with formulas), but they show that spreadsheets have been widely used in the past and are probably widely used today. Unfortunately, the question phrasing was changed in 2001 to also encompass databases. Therefore, the most recent data do not directly reflect the use of spreadsheets only.

2.2.5. Optimization models

Spreadsheets, in their most elemental form, are merely a cell-flow oriented programming system. However, many models require more advanced analysis techniques, such as optimization or constraints satisfaction (Kirkwood, 1997; Ragsdale, 2014). These allow users with limited mathematical background to perform sophisticated optimization calculations. Typically, the user specifies data in the worksheet and then uses add-ons or advanced features in the spreadsheet application to specify the targets or other parameters for the optimization.

The most common instance is the solver. The solver is a feature built-in to the application that allows the user to minimize or maximize the value of one cell by changing other cells specified by the user in a dialog box (Fylstra, Lasdon, Watson, & Waren, 1998). Solvers are available in most if not all modern desktop spreadsheet applications, including Microsoft Excel, OpenOffice Calc, Google Sheets.

A research prototype by Stadelmann demonstrated how constraints can be specified for a spreadsheet. For example, one could specify that two cells must always be equal, or that a cell must be greater than zero, simplifying some semantics that would otherwise be awkward to represent with spreadsheet formulas (Stadelmann, 1993).

While the current system design for AskSheet does not involve a solver or any constraint satisfaction functionality, both would be natural additions to the system in the future. For example, when modeling a vacation plan, it would be natural to specify a constraint that the cost and total time not exceed some values specified. Several other examples I have considered would involve some sort of optimization.

2.2.6. Decision modeling

This dissertation takes a broad definition of what constitutes a decision. However, it includes some problems from traditional decision modeling, most often applied to management science and operations research.

Numerous business textbooks teach methods of decision modeling using spreadsheets as the primary tool (Baker, 2012; Kirkwood, 1997; Ragsdale, 2014; Seref, Ahuja, & Winston, 2007; Tennent & Friend, 2011). Focusing more on the spreadsheet development itself, a paper by Mather gives a framework and general method for the development of spreadsheet-based decision models (Mather, 1999). It includes best practices, such as separating out the inputs, outputs, constants, calculation internals, and the critical logic of the model.

Some papers have explored specific applications. Ipsilandis detailed a spreadsheet model for the problem of "multi-item vendor selection (MIVS)". The objective is to find the most cost-effective way to purchase *m* items from *n* different vendors, including a fixed per-vendor handling cost. The combinatorial complexity makes this a non-trivial integer linear programming problem. Nevertheless, the author shows how it can be accomplished using spreadsheet formulas alone (Ipsilandis, 2008).

Nehzati addressed the problem of warehouse layout, another integer programming problem from operations research. Their solution used Excel as a basis, but also used Visual Basic for Applications (VBA) to handle graphical aspects of the user interaction (Nehzati, Ismail, & Rashidi-Bajgan, 2010).

Some applications use a combination of formulas and the Excel solver to solve difficult optimization problems. Troutt shows how this can be applied to multi-criteria optimization problems, such as budgeting (1991).

2.2.7. Uncertainty

AskSheet handles two types of uncertainty: missing information (not collected yet) and untrusted information (from workers who might not provide good information). The emphasis (and contribution) are entirely focused on the former, though the latter is a feature of the implementation. Thus, literature about calculations and representations of uncertainty in spreadsheets is especially relevant.

In one of the earliest uses of spreadsheets to express uncertainty, Lewis (1985) built a prototype spreadsheet application that defines cells as constraints instead of values. A constraint could be a single bound (i.e., >5), an interval (i.e., >3 and < 5), or a certain value (i.e., =6). Instead of calculating the spreadsheet as normal, it uses Prolog to evaluate the resulting system of constraints. Formulas in cells would result in a new set of constraints.

More recently, intervals have been used in conjunction with testing spreadsheets to prevent errors (Ayalew, 2001). That work explicitly provided for interval-sets as well as intervals. Interval-sets are needed because spreadsheets often contain conditionals (`=IF(c,a,b)`) and other non-continuous functions.

Representing uncertainty in spreadsheets is a challenge because the interface model is based on the assumption that each cell represents a single, concrete value. Streit (2008) showed how to augment the model so that users could add (and also remove) uncertainty information to individual cells, and have it propagated through formulas.

Although most of the above work uses intervals and/or interval-sets to represent uncertainty, AskSheet uses discrete sets of possible values instead. Actually, an early iteration used intervals and interval-sets. However, that made it difficult to reason about the probability of various outputs because the interval sets do not contain information about the relative probabilities of intervening values. Also, they are not well-suited to Boolean values or integers with a small range of possible values. With a discrete set, it is possible to assign a probability to each possible value. This approach is easier to implement, and is acceptable because the algorithms used in AskSheet assume that inputs will be either multiple choice or numbers of limited precision (i.e., integers, currency, etc.).

An alternative approach proposed by Lenz (2009) is to use multivariate Gaussian distributions. This was described only briefly in theoretical terms and I am aware of no further work on it.

2.3. Decision-making

The history of human decision-making is long and varied. It begins with prehistoric Chinese and Greek use of mystic traditions to guide their decisions; continuing with decisions on the fate of accused criminals in Greece; the development of mathematics, probability, and scientific inquiry; the development of psychology and the awareness of human cognitive processes; and finally modern computational tools for synthesizing all available information to calculate a course of action, which a principled rationale indicates will maximize the probability of an acceptable outcome (Buchanan & O Connell, 2006).

2.3.1. Benefit of information

It would be nice to be able to prove a causal relationship between the use of modern decision-making tools and the attainment of positive outcomes. Even the seemingly obvious notion that the availability of information leads to better outcomes requires thorough examination, and is not without exceptions.

A number of theories have modeled the benefit of information in specific instances of decision-making in widely different academic contexts. Stigler's *Economics of Information* theory relates the value of information about different suppliers of goods, their prices, the cost of search for them, and the benefit to a consumer, in terms of better pricing (Stigler & Stigler, 1961). In artificial intelligence, *Value of Information* theory (originally called *information value* theory) is used to model the value of individual inputs to an information-gathering agent in order to achieve some goal with the minimum cost (Howard, 1966; S. J. Russell & Norvig, 1995). Essentially, my prioritization algorithms can be considered an instance of value of information theory, although I only learned of this after they had been developed and integrated into AskSheet. Human-computer interaction researchers have modeled information foraging behavior in terms of the cost structure of accessing a given resource its initial perceived value (Chi, Pirolli, Chen, & Pitkow, 2001; D. M. Russell, Stefik, Pirolli, & Card, 1993). These models have yielded insights which have informed the design of information access interfaces to improve efficiency, primarily for individual information seekers.

More recently, some popular books have touted counter-intuitive examples in which people make better decisions when they have less information and worse decisions with more (Gladwell, 2007; Schwartz, 2003). Such examples tend to be due to human

cognitive deficiencies, such as an inability to quantify and compare by subjective properties, or the tendency to be disproportionately distracted by familiar or emotionally engaging attributes. Such cognitive deficiencies are not nearly as significant when decisions are made using tools which aid in the quantification of information and force the decision-maker to keep the various facets of the options in perspective with one another.

2.3.2. Decision support systems

Eom et al surveyed publications about 210 decision-support system applications from 1995 to 2001 (S. Eom & Kim, 2005) The survey, which followed two previous surveys covering 1971 to 1994(H. B. Eom & Lee, 1990; S. B. Eom, Lee, Kim, & Somarajan, 1998) noted movement toward using decision support technologies to support group processes, including institutional decisions, negotiation, web-based processes, and decisions that span multiple organizations. It also noted increases in the integration of knowledge management systems in decision support systems.

2.3.3. Group decision support systems

Group decision support systems (GDSS) are a class of decision support systems that facilitate group consensus building, problem refinement, brainstorming, deliberation, communication and other aspects of group problem solving processes (J. A. Y. F. Nunamaker, Briggs, Mittleman, Vogel, & Balthazard, 1997; J. F. Nunamaker, Applegate, & Konsynski, 1988). The aim of such systems differs from this dissertation because they are inherently focused on the shared responsibility of the participants. Also, they assume that all participants are involved not only in providing information, but also in shaping the strategy for arriving at the final decision—and frequently even the initial definition of the

problem. In contrast, my research assumes that an individual either formulates the problem definition and decision strategy, or receives them as the output of some external process. This dissertation is focused primarily on the process of acquiring the requisite information to follow the strategy. It is conceivable that it could someday be integrated into a GDSS, but that is outside of the scope of this dissertation.

2.3.4. Model-driven decision support systems

Model-driven decision support systems are a class of DSS that allow decision-makers to use algebraic formulas, optimization models, and simulations to reach their decision results (Power & Sharda, 2007). This includes a wide variety of specialized applications. Spreadsheet applications would mostly accurately be characterized as "DSS generators" because they essentially serve as a platform with which users create all kinds of models for different types of applications. This dissertation is focused specifically on decision support systems that are both model-driven and data-driven since the acquisition of input data is a primary focus, and it is accomplished by leveraging the model.

2.3.5. Multiple criteria decision making

A large class of decision problems share a common high-level challenge: balancing a set of criteria, often conflicting. These problems, collectively known as *multiple criteria decision making (MCDM)*, can be broken into *multiple attribute decision making (MADM)* and *multiple objective decision making (MODM)* (Zanakis, Solomon, Wishart, & Dublish, 1998).

MODM problems involve choosing from a continuous solution space, such as deciding on a number or broader solution to a problem. It is generally more theoretical

than MADM, with methods generally involving mathematical programming on continuous spaces (Triantaphyllou, 2000).

MADM concerns choosing among a finite set of alternatives. Many such problems can be represented by a decision matrix (Figure 14) at some point in the solution process. There are many competing methods of solution. A few notable examples include the weighted sums model (WSM), weighted products model (WPM), simple additive weighting (SAW), analytic hierarchy process (AHP), ELECTRE, and TOPSIS (Zanakis et al., 1998)

Because AskSheet's optimizations essentially exploit the entropy differential between the inputs and the outputs, I expect it to find more compelling applications for MADM (finite, countable solution space) than for MODM (infinite, uncountable solution space). This remains to be explored further. In any case, AskSheet is agnostic to the particular method. A spreadsheet may be used to create models expressing many different methods. The functionality added by AskSheet is not specific to any particular modeling method within MCDM or otherwise. I am discussing MCDM here because it encompasses some of the solutions one might want to implement with AskSheet.

2.3.6. Analytic hierarchy process and analytic network process

Key to solving any MCDM problem is eliciting the weights for each attribute and scoring the alternatives according to each attribute. The analytical hierarchy process (AHP) is a popular method that does both in a way that was designed to promote consistent, well-reasoned decision results (Saaty, 1980).

The analyst starts by stating an objective and breaking it into a hierarchy of criteria and sub-criteria. Then, she makes a series of pairwise comparisons among the criteria,

judging the relative importance of each. Next, she makes similar comparisons among all relevant values of for each criterion. Finally, a tool uses this information to rank the alternatives by the expected benefit from each. The calculation involves solving an eigenvector problem.

AHP has been used with a wide variety of decision problems, including not only the canonical choice and prioritization problems, but also budgeting, benchmarking and strategic planning (Forman, Gass, & Smith, 2001; Vaidya & Kumar, 2006). A generalization of AHP, the analytic network process (ANP), allows for more flexible representations of the decision, accounting for dependency and feedback between criteria and alternatives (Saaty & Hall, 1999). For example, one might want to assign more weight to criteria that better differentiate the alternatives.

	Criterion					
Alternative	c_1	c_2	...	c_j	...	c_N
1	r_{11}	r_{12}	...	r_{1j}	...	r_{1N}
2	r_{21}	r_{22}	...	r_{2j}	...	r_{2N}
⋮	⋮	⋮		⋮		⋮
i	r_{i1}	r_{i2}	...	r_{ij}	...	r_{iN}
⋮	⋮	⋮		⋮		⋮
L	r_{L1}	r_{L2}	...	r_{Lj}	...	r_{LN}

Figure 14. This prototypical decision matrix illustrates a pattern used for many models that I have used with AskSheet. *This figure was taken directly from (Zanakis et al., 1998).*

AskSheet does not support AHP or ANP. They are normally performed using specialized tools, rather than spreadsheets. Moreover, it would likely be difficult to devise the needed optimization algorithms for the eigenvector calculations. I am

discussing AHP and ANP here because they are well-known methods of solving many of the same types of decision problems that AskSheet aims to solve. Furthermore, they have seen some success at generating stable consensus (Vaidya & Kumar, 2006), making them a method of interest for structured decision making in groups. However, group use is not the primary impetus of AHP and ANP, as it is for AskSheet, and there is no assumed support for coordination or optimization of the input acquisition process.

Chapter 3. AskSheet

This chapter will explain the implementation of AskSheet. I will begin with an overview of the architecture (Figure 15) before covering the prioritization itself.

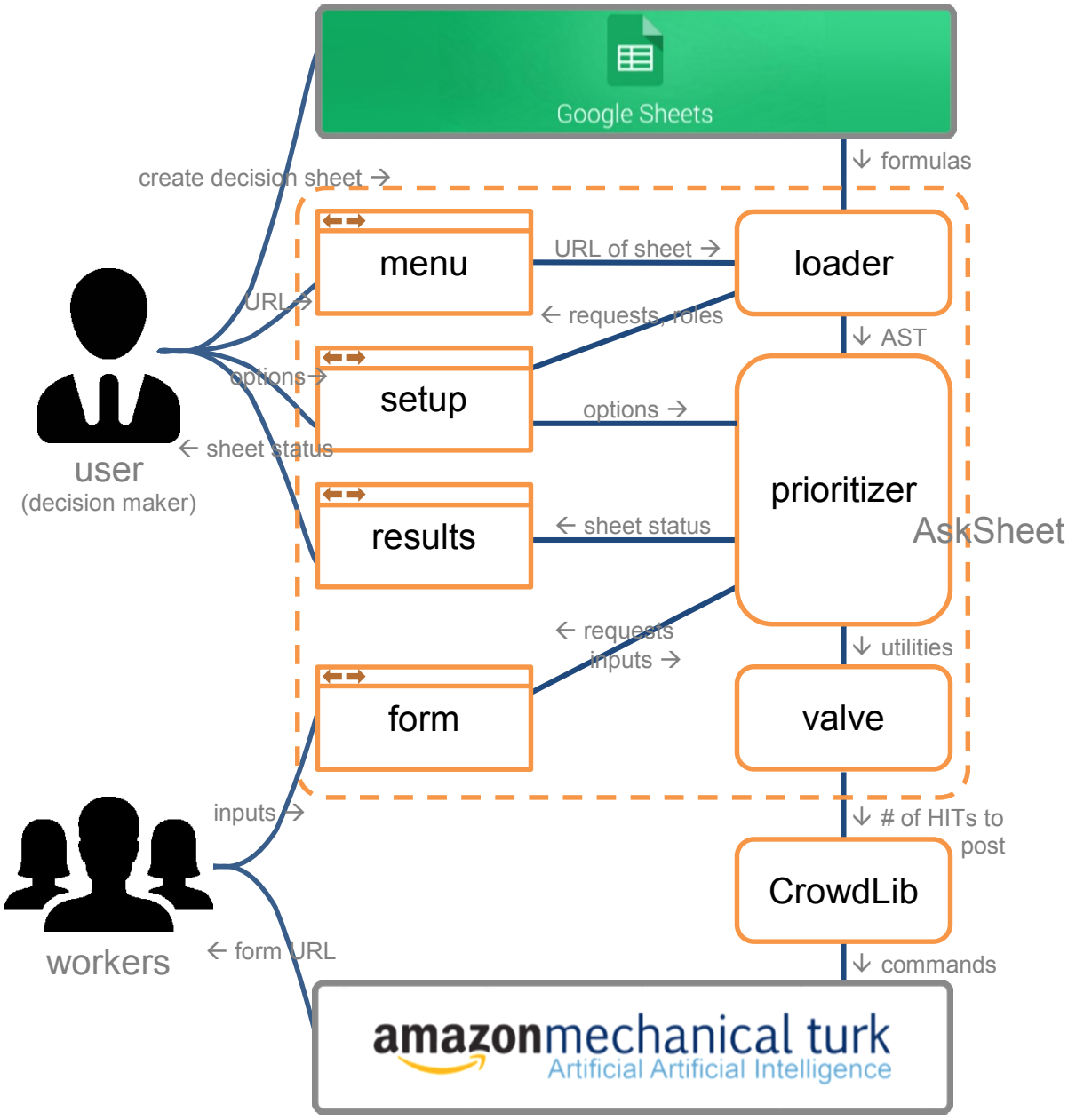


Figure 15. AskSheet sits between Amazon Mechanical Turk and Google Sheets. The boxes on the left are web pages that are accessed by the user (decision maker) or workers. The ones on the right are backend modules.

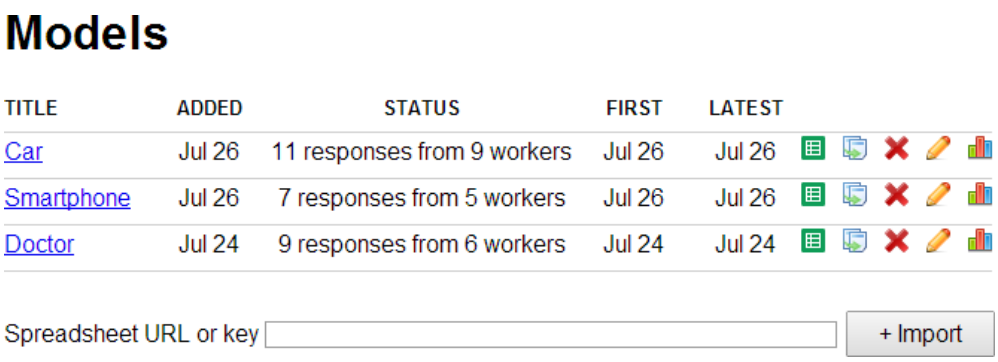


Figure 16. The *menu* page shows all of the user's decision models.

The first step to creating a new decision project is to create a spreadsheet in *Google Sheets*, an online spreadsheet and collaboration service. AskSheet presumes that the user (someone with a decision to make) is competent enough with spreadsheet formulas to create a model of their decision.

Next, the user enters the URL of the spreadsheet document she created into the AskSheet *menu* page (Figure 16). This page also shows other models that have been created, and provides typical tools for copying and deleting models, viewing the results, or returning to the source spreadsheet.

The *loader* module in the backend retrieves the formulas of the spreadsheet from Google Sheets using a public API. Those formulas are parsed into ASTs and then joined on cell references to create a structure that covers the entire spreadsheet model. (An example is given later in this chapter. The full grammar is given in Appendix A.)

A limitation of my implementation is that once the model has been imported, it is fixed. The user is not able to make changes to the weights or labels after that point.

In addition to the formulas, the loader also retrieves an HTML-formatted version of the spreadsheet. It will later modify this to create the input forms used by workers.

The loader analyzes the spreadsheet formulas using parameters to the `=ASK(…)` formulas (Figure 20, page 63) to determine how many separate types of tasks there are (i.e., gathering details from separate web sites in parallel). Using this information, the *setup* page (Figure 17) elicits information about the tasks, such as the instructions, estimated time to complete each task, target price per hour, batch size, and so on. This is the basic information needed by the prioritizer and other aspects of the system.

Prioritization will be covered in depth in the next section. Its primary role is to calculate the utility of each of the `=ASK(…)` formulas (decision inputs). The *valve* module of the backend is responsible for forming coherent batches of inputs and deciding when to post more jobs to Mechanical Turk.

Although at any given time, there is usually only one cell that is the top priority, AskSheet allows multiple roles to be active at a time for the sake of enhancing turnaround time. A role is a separate branch of workflow with its own instructions. Normally, it entails gathering a particular type of information and/or from a particular source. At any given time, a role can be marked as active or inactive. At each step, the valve makes a role active if its highest utility batch of inputs has a combined utility in the top 25% of all batches of inputs. Once active, a role becomes inactive only when its highest utility batch of inputs has a combined utility in the bottom 50%. In other words, it allows a worker to keep working even if the task they are working on is no longer the absolute top priority, but it stops them if it becomes a below-average priority. If the user has elected only one judgment per input, then at each step, the valve ensures that there is exactly one HIT available or in progress for each active role.

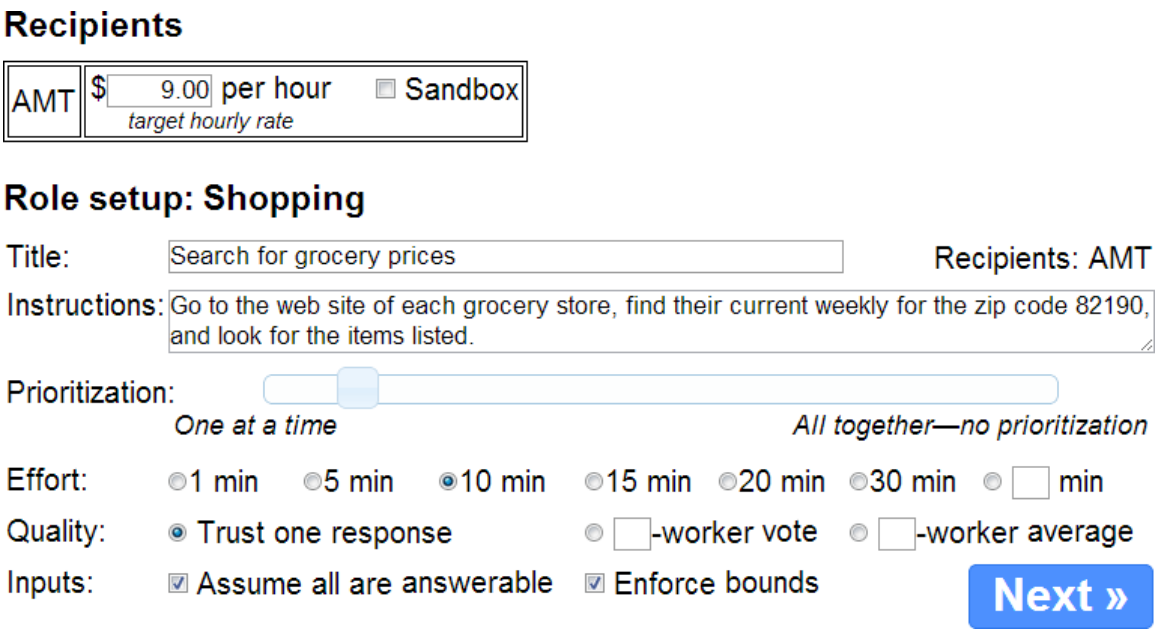


Figure 17. The *setup* page is where the user (decision maker) enters instructions to workers, as well as details that affect the amount workers will be paid and how their task will be structured.

The CrowdLib module is a general purpose module that I created for working with Mechanical Turk. It wraps the public API of the service in a clean API, and keeps a local database to track the state of all jobs in Mechanical Turk's systems.

As the results are entered by workers, AskSheet receives real-time updates. The user can view the status using the results page (see section 1.4 on page 9).

Algorithms for the above processes can be found in Appendix B (page 123).

3.1. Parsing spreadsheet formulas

When a decision model is imported, AskSheet retrieves the formulas from Google Sheets using a public API. The formulas are then parsed using a spreadsheet formula parser that I developed as part of the implementation. The parser accepts a subset of the grammars accepted by Google Sheets and Microsoft Excel, with the addition of the `=ASK(...)` formula. Among features not supported are array formulas (e.g.,

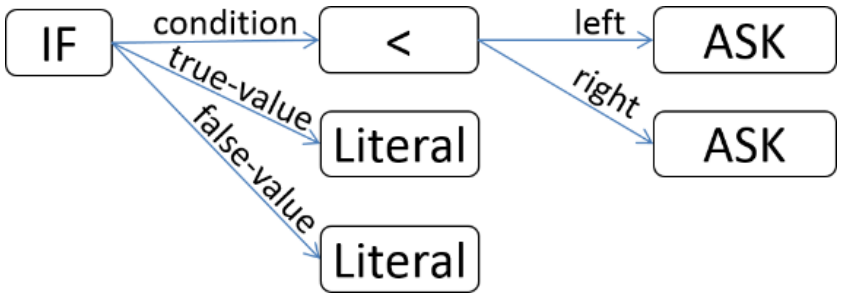
`=SUM(IF(A1:A100=B1:B100, 1, 0))`), array literals (e.g., `{"good", "great", "excellent"}`), and cross-sheet cell references (e.g., `Sheet1!A1`). The full grammar accepted by AskSheet is given in Appendix A (page 122).

3.2. Prioritization

The most important component of AskSheet is the prioritizer. To illustrate how it works, I will refer to this 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 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). I 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.

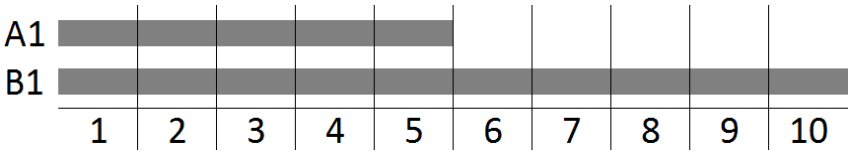
The goal is to find an ordering of the requests that will minimize the expected cost of evaluating the model—finding a result for all root cells. I think of the *cost of an ordering* as the expected sum of the costs from all requests that would be fulfilled (i.e., not

eliminated) if they were gathered in that order. I 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, I will 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, one 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. My implementation treats every possible value as equally probable. In other words, each cell—and in fact each node of the AST of each cell—is represented as a random variable with a discrete uniform distribution [3].



[3] The choice of the uniform distribution is imperfect, but inevitable. For the common types of data used in these decision models, there is no well-established default. I considered using a normal distribution, but data on product ratings from commercial web sites indicates that they are not normally distributed, and it depends on the subject matter. For prices, I discussed with Prof. Phillip Leslie, a professor of economics at UCLA who is an expert on pricing. He said that there is no well-established default distribution for product prices. The logistic distribution is commonly found in the literature for convenience of those calculations, but he said the uniform would work just as well. For AskSheet, I felt that uniform was the most neutral.

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 the 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* ≠ *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 an arithmetic operation cannot be calculated without all of its operands (except for a few special cases, such as 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 | acquire *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). Some examples are shown in the table below.

3.2.1. Operations

AskSheet 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.

In addition, Figure 18 illustrates the output distributions with a series of simple examples. The calculation for the output distribution of =MAX(…) is shown below. All of the algorithms are given in detail in Appendix C (page 129).

3.2.2. 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:

$$\Pr(\max = k) = \Pr(\max \leq k) - \Pr(\max < k)$$

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

	A	B
1	1	*Pr(A1=1) = 1*
2	“Chocolate”	*Pr(A1 = “Chocolate”) = 1*
3	=ASK(“1 to 3”)	*Pr(A3=1) = 1/3*
4	=ASK(“1 to 3”)	*Pr(A4=1) = 1/3*
5	=ASK(“1 to 3”)	*Pr(A5=1) = 1/3*
6	=A3 * 2	*Pr(A3=2) = 1/3* *Pr(A3=4) = 1/3* *Pr(A3=6) = 1/3*
7	=A3 + 2	*Pr(A3=3) = 1/3* *Pr(A3=4) = 1/3* *Pr(A3=5) = 1/3*
8	=A6 + A7	*Pr(A8=5) = 1/9* *Pr(A8=6) = 1/9* *Pr(A8=7) = 2/9* *Pr(A8=8) = 1/9* *Pr(A8=9) = 2/9* *Pr(A8=10) = 1/9* *Pr(A8=11) = 1/9*
9	=MAX(A3:A5)	*Pr(A9=1) = 1/9* *Pr(A9=2) = 7/9* *Pr(A9=3) = 19/9*
10	=SUM(A3:A5)	*Pr(A10=3) = 1/9* *Pr(A10=4) = 3/9* *Pr(A10=5) = 6/9* *Pr(A10=6) = 7/9* *Pr(A10=7) = 6/9* *Pr(A10=8) = 3/9* *Pr(A10=9) = 1/9*
11	=A4=A5	*Pr(A11=TRUE) = 1/3* *Pr(A11=FALSE) = 1/3*
12	=A5 > 1	*Pr(A12=TRUE) = 2/3* *Pr(A12=FALSE) = 1/3*
13	=AND(A11, A12)	*Pr(A13=TRUE) = 2/9* *Pr(A13=FALSE) = 7/9*
14	=NOT(A13)	*Pr(A14=TRUE) = 7/9* *Pr(A14=FALSE) = 2/9*
15	=IF(A4=1, “one”, “two”)	*Pr(A15=”one”) = 1/3* *Pr(A15=”more”) = 2/3*
16	1	“una”
17	2	“dos”
18	3	“tres”

Figure 18. This table illustrates the output distributions of various functions by example. The algorithms used to calculate each of these are described in Appendix C (page 129).

3.2.3. 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 .

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.

3.3. Batching inputs for worker efficiency

Up to now, this discussion has assumed that that inputs are acquired one at a time. However, in most cases, that would be inefficient for a worker because loading a new HIT and checking the instructions has overhead. It would be more efficient for workers to have more than one input in each task. To support this, AskSheet actually batches 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).

A batch is a set of inputs—typically no more than a few—that are shown to a worker within a single HIT. The user sets the batch size in the settings panel. (An example of

a batch and the corresponding settings panel was shown in the grocery example on in section 1.4.1 page 12.)

To create a batch of length *n*, AskSheet follows the following process:

1. Sort inputs by the following parameters:
    a. `itemLabel` (parameter to `=ASK(…)`; typically a decision alternative)
    b. utility
    c. row number of cell
    d. column number of cell
2. Form as many *n*-length batches as possible from the sorted inputs.
3. Sort the batches by the sum of the utilities of the inputs in each batch.

The result is a set of batches which minimize the number of decision alternatives that a worker has to focus on in a given task, while secondarily maximizing the utility of the inputs from each batch. It balances the inherent trade-off between optimizing the worker's focus in any given task and optimizing the order in which the inputs are acquired. The benefit of this scheme is illustrated in Figure 19.

		Safeway
Ribeye steak	1 lb	$

(a) With just one input per task would mean that a worker would waste effort going back to the site for each task. (BAD)

		Safeway	Giant	Shoppers
Ribeye steak	1 lb	$	$	

(b) If inputs were strictly prioritized by utility, workers would waste effort going back and forth between different web sites. (BAD)

		Safeway
Coffee, ground	32 oz	$
Heavy cream	1 gal	$
Hormel ham slices	3 lbs	$

(c) Because the `=ASK(…)` formulas in this model specify the store name as the `itemLabel` parameter, AskSheet is able to group inputs by store, to balance worker efficiency with the prioritization of the inputs. (GOOD)

Figure 19. AskSheet's batching uses the `=ASK(…)` parameters to enhance a worker's efficiency within a given task. In the example shown, the grocery stores (columns) are the alternatives, and the prices of items are the attributes.

In addition to grouping inputs by `itemLabel`, the parameters to `=ASK(…)` also allow it to split the job into multiple roles (HIT Types) which are run in parallel [4]. (See Figure 20, page 63.) 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.

[4] The original vision for AskSheet also included the idea of coordinating crowd workers and trusted co-workers working in parallel in support of the same decision process. Although I ultimately narrowed the scope of my dissertation work to include only crowd workers on Mechanical Turk, this broader vision is still an integral part of the AskSheet architecture, as evidenced by the `recipientSpec` parameter to the `=ASK(…)` function. In the future, AskSheet could be extended such that `recipientSpec` parameter would accept other contact details (e.g., email addresses, instant messaging handles, SMS text message numbers, social network connections, etc.). Requests for assistance would be automatically sent via those channels, just as they are posted to Mechanical Turk now.

3.4. 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 20).

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 21 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

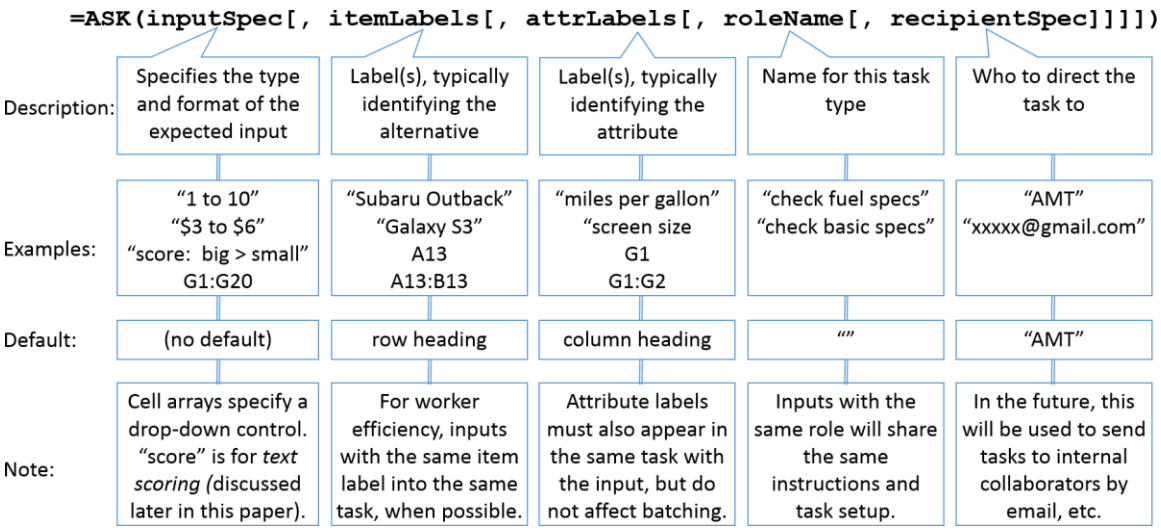


Figure 20. The `=ASK(…)` formula parameters are at the core of the strategy for partitioning inputs into efficient tasks. The `recipientSpec` parameter reflects an earlier vision for AskSheet, but is not currently used [4].

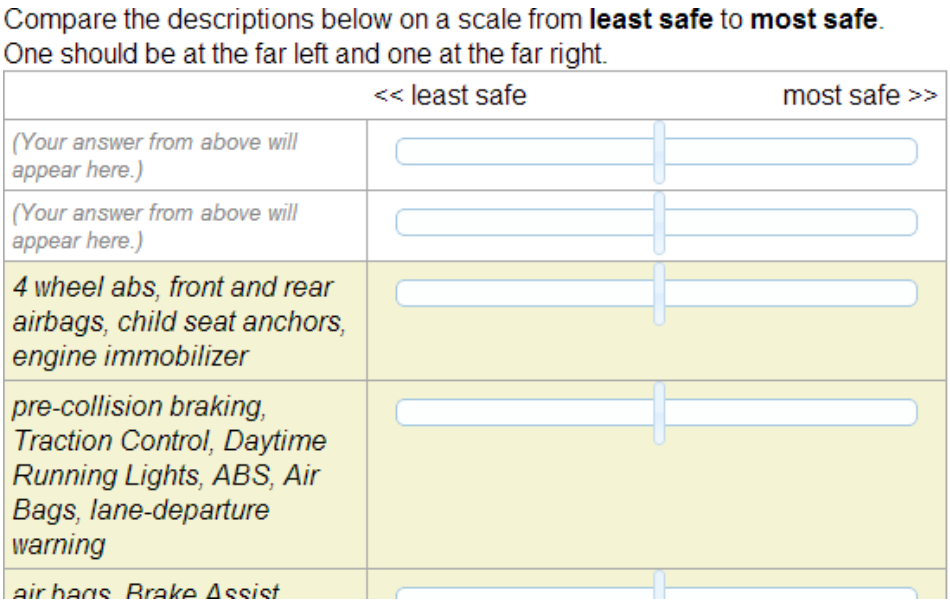


Figure 21. Text scoring interface

(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, it collects 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 22).

This strategy was adopted to allow a richer set of models including inputs that would otherwise be difficult to quantify. However, it is not intended to be primary basis of a model. A more thorough study of text scoring and its statistical properties would needed to fully understand its ability to reconcile scores from different contributors.

3.4.1. 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) (Ranneby, 2013), 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?).

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

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

These formulas are derived in (Cheng & Amin, 1983, p. 397). Although this relies on our presumption of uniform distribution—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.

3.5. 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).

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

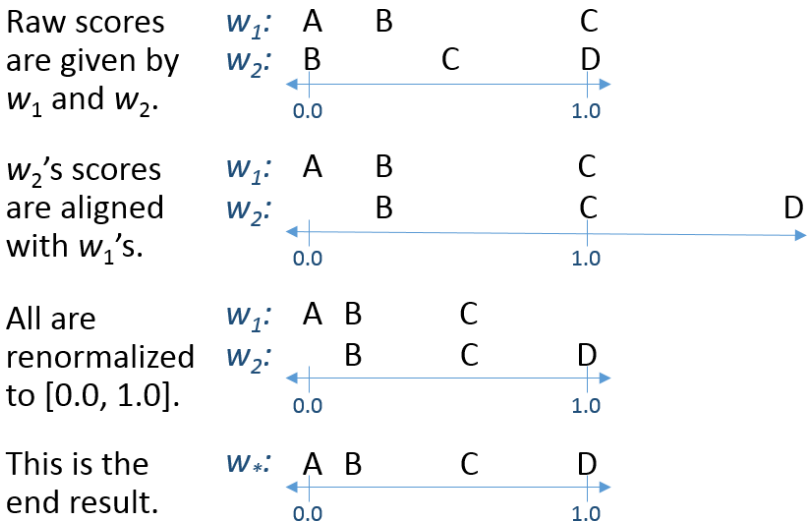


Figure 22. In this minimal example, text scores from two workers who each scored three items are aligned.

input might be eliminated by the optimizations. This approach was inspired by Get Another Label (Sheng & Provost, 2008), 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.

3.6. AskSheet Assistant

Since users' experience with spreadsheets may vary widely, I constructed a form-based interface that allows users to create spreadsheet models by simply entering their criteria and a list of alternatives. This was an early goal for my dissertation, and became the basis of a field study that will be described in section 4.4 (page 86).

The interface is simple, yet powerful enough to express a wide range of models based on a weighted sum, constraints, or both. To create a model, the user follows the following steps:

1. Enter a title for the decision
2. Decide on the basic model type:
    a. *first match* will find any alternative that meets the user's baseline requirements
    b. *best match* will maximize a weighted sum
3. Enter the decision criteria. A criterion may be any of the following:
    a. integer (with specified bounds)
    b. currency (with specified bounds)
    c. yes/no

d. true/false

e. 5-star rating

f. choice (translates to drop-down control)

g. relative (using method described in section 3.4 on page 63)

4. Enter a list of the alternatives.

Using this information, AskSheet Assistant constructs a spreadsheet (not shown to the user), which is automatically loaded into AskSheet. A screenshot of the interface is shown in Figure 23.

The alternative labels will become row headers. Criteria labels will become column headers. For the *first match* option, it will use an `=AND(…)` formula to determine if all of the constraints for a row have been met and then use an `=OR(…)` to determine if any of the rows meets all of the constraints. Since AskSheet uses short-circuit evaluation, it will stop as soon as any row meets all of the criteria. For the best match case, it uses a simple weight sum using `=MAX(…)` and `=SUM(…)`. If constraints are given along with weights, then it uses an `=IF(…)` to force the sum to zero whenever the constraints are violated.

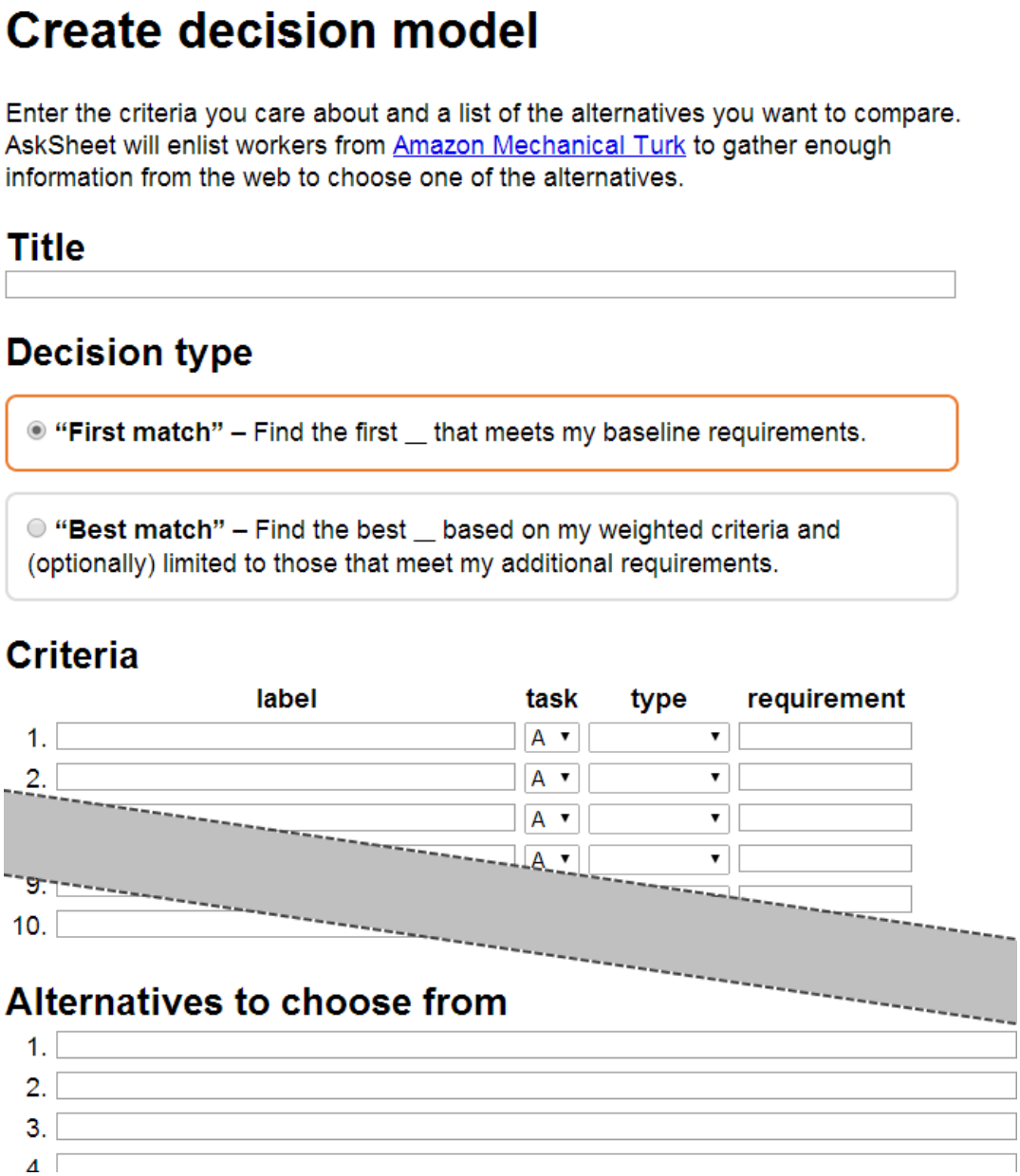


Figure 23. AskSheet Assistant allows a user with no spreadsheet experience to create a spreadsheet decision model.

Chapter 4. Field studies

This chapter will detail several field studies that I conducted to understand the efficiency and applicability of AskSheet. As with other complex systems, there are a number of ways AskSheet could be measured or evaluated. Therefore, I preface the discussion of the studies and results with a discussion of scope and how I decided which questions to address.

4.1. Scope

There are six factors that I believe would affect the success of AskSheet. The studies in this chapter will focus solely on the first two; the third will be covered in section 6.2.2 (page 106).

1. **efficiency** – Does the prioritization method result in a substantial reduction in the number of inputs that must be acquired? How much time and money are required to reach a decision result?

2. **applicability** – Can the system process models of real decisions that users care about?

3. **performance** – Can the request priorities be computed within time constraints that are compatible with the needs of the workflow? (Since the prioritization must be calculated in real-time whenever a worker submits new inputs, I aim for a maximum of 30 seconds.) Performance is covered in in section 6.2.2 (page 106).

The remaining three factors are not evaluated because they are orthogonal to the scope of my intended contribution: a method for *coordinating workers* to efficiently enter data that are then used as input to a spreadsheet model.

Each of these factors would clearly affect the value of any implementation as a product. I believe the most natural way to evaluate any of these three factors would be in isolation from the rest of AskSheet.

4. **worker usability** – Is the worker interface quickly learnable so that the barrier to begin working is minimal? Does it support efficient input? For a worker, the time to do a task consists of (a) understanding the instructions, (b) finding and interpreting the requested data from some external source, and (c) entering the requested data into the form. An ideal interface would minimize (a) and (c).

   The tasks posted by AskSheet are among many on Mechanical Turk that ask workers to find information online and enter it into a form. This is one of the most common applications of Mechanical Turk. The form fields and task structure used by AskSheet are not significantly different from any other such tasks.

   Worker usability is also impacted by the quality of instructions that the requester (decision maker) enters after importing a new model. However, those instructions are part of the input to AskSheet. Thus, any evaluation that measures the quality of the instructions would be a reflection of the system itself.

5. **requester usability** – Is the interface for requesters (decision makers) sufficiently learnable so that this barrier to using the system is minimized?

AskSheet includes web interfaces for importing decision models, entering the setup, and monitoring the results. However, these interfaces were designed primarily for my use, and for communicating the capabilities and functional semantics of the system by way of screenshots in this dissertation and related publications. The usability of this interface does not impact the ability of the system to coordinate workers to acquire decision inputs.

6. **robustness** – Are the decision results returned sound and dependable?

   The correctness of a decision result depends on two factors: (a) the suitability of the requester's decision model to the problem and (b) the accuracy of the inputs. To calculate the decision result, AskSheet simply substitutes the acquired inputs for the corresponding `=ASK(…)` formulas and then calculates the values of the other formulas that depend on them.

   (a) The suitability of the model depends on the user's ability to express their decision problem in spreadsheet formulas.

   (b) The accuracy of the inputs is generally a function of many factors: the clarity of instructions, task difficulty, accuracy of the information at the web sites workers visit, strenuousness of the qualification requirements[5] I impose on the tasks, random chance (which workers choose to do the task at a given time), and the number of judgments and aggregation method selected by the requester when the user imported the model.

[5] Qualification requirements are a feature of Mechanical Turk that lets requesters limit access to their tasks by the reputation and experience of the workers. For example, it is possible to limit tasks to only workers who have done at least *n* tasks, or whose prior submissions were accepted by the requesters at least *m*% of the time.

Any evaluation that measured—or was confounded by—the clarity of the instructions, the difficulty of finding and interpreting the requested information, or the accuracy of the web sites they visit would not be an accurate reflection of how well AskSheet coordinates workers to gather the inputs. All of these factors are active targets of current research in crowdsourcing and human computation, and are most effectively studied in isolation from a complex system such as AskSheet.

While my implementation includes features for aggregating multiple judgments—by consensus or averaging—those features were not used in these field studies. Instead, I chose to run the trials with only one judgment per input. Although multiple judgments can be expected to improve the accuracy of inputs, it also leads to results that are more difficult to explain clearly. For example, when aggregating by consensus of 3 workers, AskSheet initially requests two judgments, and then requests the third only if the first two do not agree. If after only one has been received, an input from another worker causes the first input to be culled, the first judgment is wasted. Then, expressing the proportion of inputs that were eliminated (the percent saved) becomes unnecessarily complicated. Thus, since the accuracy of the inputs is not a focus of my dissertation, I elected to work with only one judgment.

For all of these reasons, I have chosen to explicitly exclude robustness of the result from the scope of my contribution and, in turn, this evaluation.

4.2. Field trials with AskSheet

The field trials in this section were based retrospectively on actual decision problems in my own life that had entailed a substantial effort (e.g., a few hours) to gather

the details. In each case, I created a spreadsheet model in Google Sheets, loaded it into AskSheet, and used workers on Mechanical Turk to gather the decision inputs. Since minor details were changed for simplicity or privacy (e.g., my health insurance provider), I describe each in the form of an anecdote with a fictional name.

For each of the models, I ran the system three separate times in order to observe any variations. Some variation in cost, uptake, and efficiency (reduction of inputs) is normal and expected due to the availability of workers at a given time, the competence of the particular workers who accept my task, or random issues with timing.

4.2.1. Field trial #1: 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:

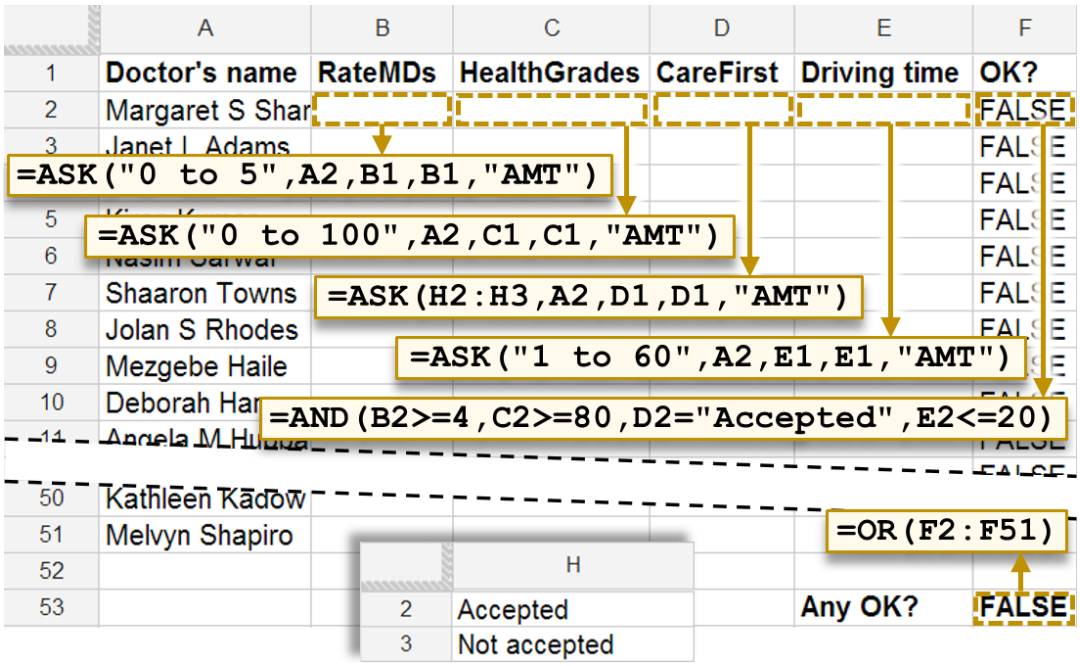


Figure 24. Model for field trial #1 (pediatrician)

- 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.

I tested this scenario using the model in Figure 24. 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 my implementation of AskSheet requires the user to fill in the row labels in the model. For this example, I found a list of pediatricians online. Chapter 4 details a method that I developed later for enumerating the alternatives for decision problems such as this.

The results were as follows. More detail can be found in Appendix F (page 178).

	#1	***#2***	***#3***
inputs required	36 of 200	108 of 200	64 of 200
completion time	47 minutes	55 minutes	42 minutes
cost	$5.67	$19.49	$11.94
# of workers	6	13	10
worker time, total	58 minutes	160 minutes	108 minutes
hourly rate	$5.89	$7.33	$6.61
decision result	Sharon Towns	Sharon Towns	Sharon Towns

Although efficiency—eliminating inputs that will not affect the final result—is an important goal of AskSheet, it would be a mistake to put too much emphasis on the percent of inputs that were needed. Because this model selects the *first match*, it will stop

acquiring inputs as soon as a match is found. Thus, if I even if I had added 100,000 doctors' names instead of just 50, it would still need the same number of inputs to find a match.

All tasks paid $0.63 for 4 inputs, except that in #2 and #3, the RateMDs task offered $1.25 for 4 inputs. That was due to my error when setting up the tasks. AskSheet asks for a time estimate for each task, and I mistakenly entered 10 minutes instead of 5 minutes for RateMDs. From that estimate, it calculates the price per task. This does not appear

to have had a significant impact on uptake, as measured by the number of workers who did each type of task.

	#1	#2	#3
RateMDs	2 workers	4 workers	3 workers
HealthGrades	3 workers	5 workers	3 workers
CareFirst	1 workers	3 workers	2 workers
driving time	2 workers	6 workers	5 workers

4.2.2. 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:

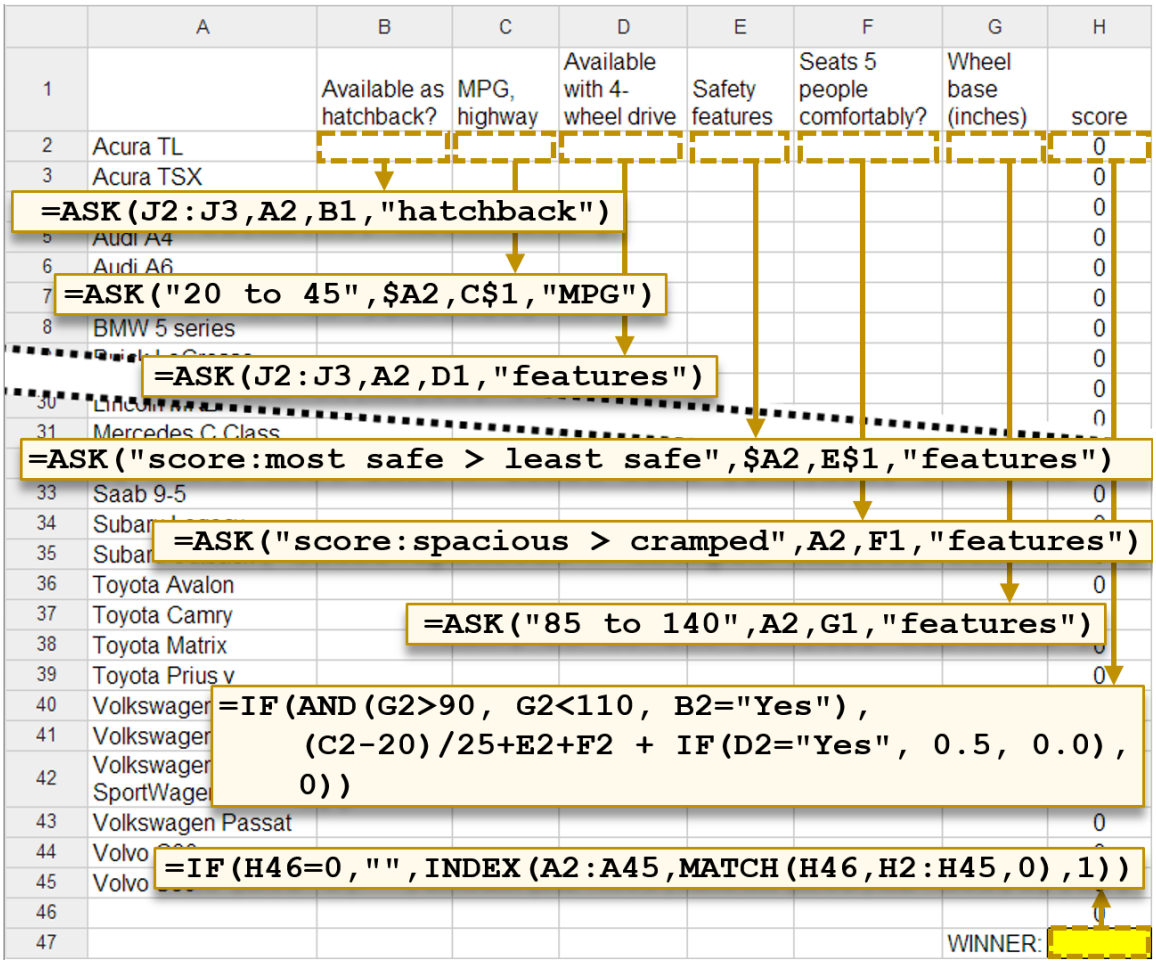


Figure 25. Model for field trial #2 (car)

- 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. The model is shown in Figure 25 (page 77).

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 21).

The results were as follows. The features task paid $0.50 for 6 inputs while the others paid $0.50 for 4 inputs.

	#1	***#2***	***#3***
inputs required	62 of 120	60 of 120	72 of 120
completion time	114 minutes	193 minutes	139 minutes
cost	$7.50	$7.50	$8.00
# of workers	12	7	6
worker time, total	95 minutes	122 minutes	130 minutes
hourly rate	$4.75	$3.70	$3.69
decision result	Subaru Outback	Toyota Camry	Hyundai Tuscon

The inconsistency of the decision result is due to differences in the way workers scored the safety and spaciousness of models, as well as some inputs that were not consistent between the repetitions. All of the inputs are reproduced in Appendix F on page 185. Differences in individual inputs may be due to a variety of factors, including misunderstanding the instructions, careless mistakes, or even incorrect information at the web sites the workers used to find information.

In general, a standard strategy for obtaining accurate inputs from crowd workers is to request multiple judgments and aggregate them using either consensus (voting), averaging, or other methods. Although AskSheet supports that feature, I chose not to enable it for these field trials because accuracy is outside the scope of my contribution, and enabling that feature would significantly complicate the other metrics that I wanted to report. (See the discussion about "robustness" on page 72.)

In examining the inputs, I found a case where multiple judgments would not have helped. In trial #1, the Toyota Camry was eliminated because the worker entered that the model does not have a hatchback option. In trials #2 and #3, the workers indicated that a hatchback option *is* available. Since AskSheet asks workers to enter the URL where they found the information, I was able to check the sources that the workers used. The page cited by the worker in trial #1 contains an abundance of detailed information about the 2013 Toyota Camry, including photos and descriptions of six variations ("trims") of the Toyota Camry, none of which is a hatchback [6]. The worker in trial #2 cited a page from the Kelley Blue Book web site (a popular resource for car information) that refers to a

[6] https://autos.yahoo.com/toyota/camry/2013/

"2013 Toyota Camry Hatchback" next to a photo of a sedan [7]. The worker in trial #3 entered sources for the other three models included in the task, but the sources did not directly relate to the Toyota Camry.

From the manufacturer's web site, I learned that the worker in trial #1 is correct. While a history of that model shows that a hatchback was available in earlier model years [8], none of the web pages cited by the workers mention that.

The problem of how to elicit *correct* answers in crowdsourced data collection would best be studied in isolation from AskSheet. Possible approaches could include effective ways of aggregating multiple judgments, guidelines for communicating instructions, or algorithms for estimating the reliability of individual workers. All of these are orthogonal to the central research problem that this dissertation addresses: *how to coordinate workers to efficiently enter data for use as input to a spreadsheet model*. (See the section on "robustness" in section 4.1 on page 70.)

4.2.3. Field trial #3: Smartphone shopping

Another example I 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.

---

[7] http://www.kbb.com/car-pictures/2013-toyota-camry-hatchback-pictures/?id=382994

[8] http://www.edmunds.com/toyota/camry/history.html

I 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, I 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 I 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 features task paid $0.50 for 6 inputs while the others paid $0.50 for 4 inputs. The results were as follows.

	#1	***#2***	***#3***
inputs required	40 of 63	44 of 63	62 of 63
completion time	80 minutes	111 minutes	75 minutes
cost	$8.00	$8.80	$13.30
# of workers	4	9	5
worker time, total	65 minutes	82.7 minutes	125 minutes
hourly rate	$7.37	$6.38	$6.39
decision result	Samsung Galaxy S4	Samsung Galaxy S4	*(inconclusive)*

In the third iteration, workers left 10 fields blank. In the AskSheet setup screen, the user (decision-maker) can decide whether all inputs are required or not. That effectively specifies whether it will allow a worker to enter blanks or not. In some cases, blanks are normal since sometimes an item cannot be found or the information is not available. At that point, the worker will have already spent time trying so as a rule, I pay for the task unless there is evidence of a significant effort to cheat. The effect in this case is that AskSheet was unable to reach a conclusive decision result. However, even in such a case, AskSheet displays to the user the relative probability that each alternative would be the decision result if all inputs were available. A screenshot of this is shown in Figure 40 (page 184).

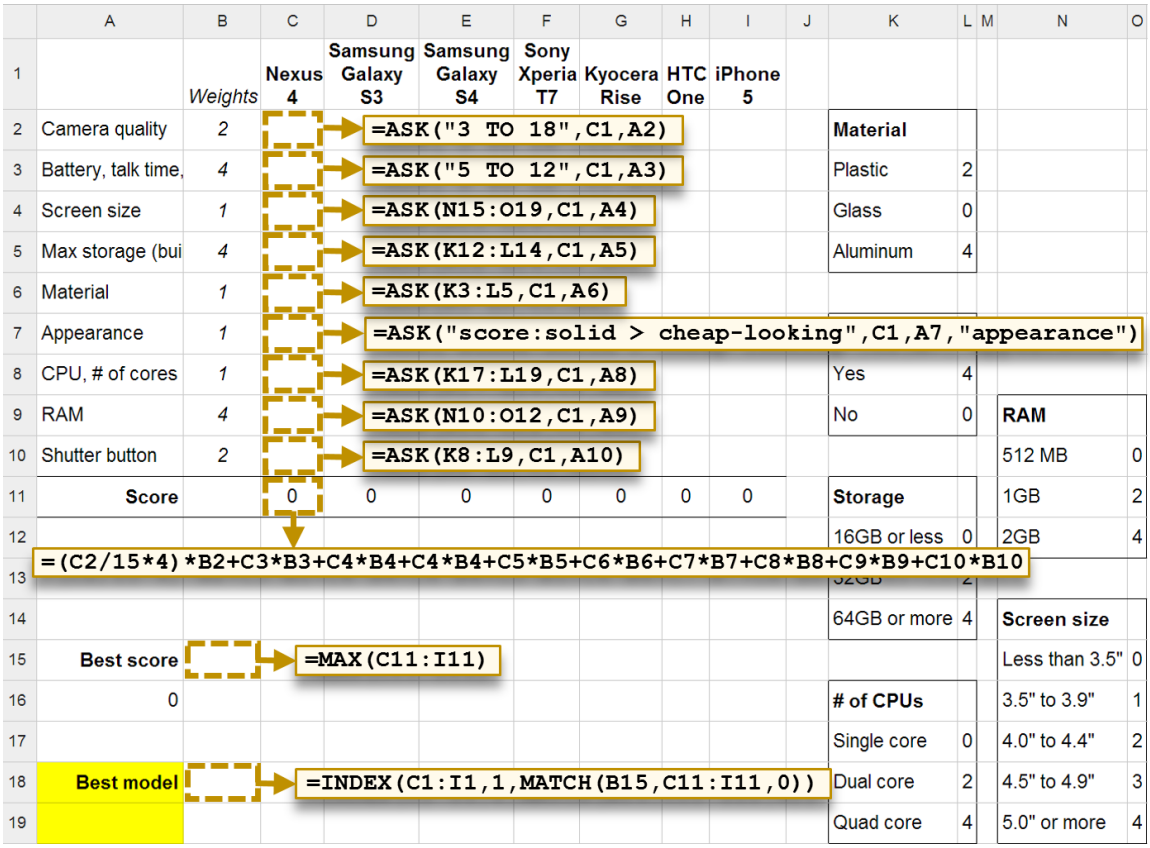


Figure 26. Model for field trial #3 (smartphone). Note that C11 could be shorter if AskSheet supported `=SUMPRODUCT(…)`. It would become:
`=C2/15*4 * SUMPRODUCT(B3:B10, C3:C10)`

4.2.4. Summary of AskSheet field trials

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 I believe to be a financially and temporally viable process.

All of these were consumer-oriented decisions based on models that I created. In the next chapter, I will discuss some of the factors that determine the applicability of this method to decisions with more inputs and/or institutional complexity.

4.3. Survey of decision problems

The field trials demonstrated that AskSheet works as it is designed to, but they did not establish whether there are any potential users with decision problems that might be amenable to AskSheet—or potentially some other crowd-driven decision input acquisition system. Therefore, I conducted a web-based survey to validate whether gathering decision inputs is actually a prevalent problem for any substantial set of users [9].

Respondents were asked to list 3-5 types of decisions they find especially time-consuming. As results were received, I adjusted the wording of the question slightly to convey the idea of data-driven, time-consuming tasks—without giving any concrete examples that would bias the responses. The exact wording of the three versions of the question are reproduced in Appendix D (page 161) [10].

---

[9] The survey was approved by the University of Maryland Institutional Review Board (IRB) under application #346307-1. All respondents agreed to an informed consent form, which explicitly gave permission for their responses to be used verbatim in future communications.

[10] Respondents were also asked questions about spreadsheet competence and demographics, but those details are omitted here because they do not yield any insights that would value this dissertation. They were originally intended for my use in recruiting future study participants, a goal that I later abandoned.

It should be emphasized that this was not designed to yield statistically significant insights about any population. The objective was to produce a *list* of examples of decision problems people find time-consuming.

It was advertised only by social network web sites, once in April 2013 and again in May 2014. Respondents were offered a 1 in 20 chance of winning a $20 gift card.

A total of fifteen (15) valid survey responses were received (excluding blank, mostly blank, and fraudulent responses). The responses are reproduced verbatim in appendix . Among the 15 survey responses were 53 entries marked as decision problems.

I removed 12 that were clearly not decision problems (e.g., "writing lessons", "gathering information", "doing taxes", etc.). I then manually edited the remaining 41 items for clarity and grouped them into twelve topics (e.g., finances, car, etc.), as shown in Figure 27 (page 85).

Finally, I classified them according to whether they meet a baseline set of criteria for applicability to input acquisition by crowd workers:

- might take over 1 hour if done normally (i.e., without crowdsourcing)
- could be expressed in terms of spreadsheet formulas
- primary obstacle to deciding is the acquisition of publicly available information

	potentially applicable	*inapplicable examples*
travel	• which hotel to stay in	• ways to get together with family
finances	• financial decisions (purchases, account management, tax forms) • living arrangements as I age • investments, e.g. stocks, other paper assets, real estate, etc. • which insurance company for home, car, and life insurance	*(all are potentially applicable)*
food	• where or what to eat can take an hour for me • choosing a place to eat lunch	• what to make for dinner • which vegetables to buy at the super market
car	• which car to buy • buying a new car	• whether we can afford a new car
other	• which new name for my business • which breed of a dog to adopt	• deciding what to say/not say in emails
reading material	*(none are potentially applicable)*	• what papers to read takes time finding them/reading abstracts • what book to read next • gathering papers to read
time management	*(none are potentially applicable)*	• whether or not some project • ordering tasks in my day • the order of my routine when I wake up or arrive home
coordinating people	*(none are potentially applicable)*	• scheduling a meeting for others • decisions in a committee
home	• which place to move to requires a lot of asking around	• which house to buy *(x2)* • buying a new home
purchasing	• which vendor to order from • which system to purchase	*(all are potentially applicable)*
setting goals	• requirements for a project	• choosing a research topic
technology	• which expensive items to buy (electronics, furniture) takes a lot of time reading and comparing items • what software to use requires a lot of reading and asking friends • which new technology to buy • which cell phone provider to use • online shopping for electronics.	*(all are potentially applicable)*

Figure 27. Examples of time-intensive decision problems were gathered in a survey. I manually edited them for clarity, categorized by topic, and classified by potential amenability to some form of crowd-supported input acquisition.

Based on the examples classified as "potentially applicable" above, we can see that many would depend heavily on publicly available information (i.e., details that could be gathered by crowd workers), but may also depend on personal preferences or social agreement. A weakness of my approach is that it does not integrate the latter factors. However, these data also indicate a number of specific application areas in which AskSheet or some future system could someday be deployed.

4.4. Modeling study

Another question that arises in relation to the viability of AskSheet is whether users are actually able to model their problems in a way that the system could work with. Conversely, one could ask this question: *Is AskSheet applicable to real decision problems of ordinary users?*

To address this question, I recruited nine adults from existing contacts (friends, acquaintances, and extended family) to create decision models that could be used with AskSheet [11]. Since a given participant's experience and competence with spreadsheet formulas may be hard to predict, I had the participants use AskSheet Assistant (see section 3.6 on page 67) to create their models. Even so, that interface requires some explanation, so I met with each participant by in person or by phone and worked with them throughout the entire process, which lasted about one hour each. None of the participants is currently a programmer, although three have had some prior programming experience.

[11] The modelling study was approved by the University of Maryland Institutional Review Board (IRB) under application #601000-1. All respondents agreed to an informed consent form, which explicitly gave permission for their responses to be used verbatim in future communications.

The interface of AskSheet Assistant ensured that all models met the technical requirements of AskSheet and some practical requirements of working with crowd workers. All models were able to be imported into AskSheet. However, they were not run with Mechanical Turk because none of the participants had time in the 1-hour study period to write instructions sufficiently clear for crowd workers to follow reliably.

A total of ten models were created, with one participant voluntarily creating a second model after the conclusion of the session. All of the models are reproduced in Appendix E on page 168.

The topics of the models were as follows: [12]

- **"Options for child birth"** – Eight (9) child birth methods (e.g., "Hospital Scheduled Induction without Epidural", etc.) are compared using a weighted sum of nine (9) attributes, with "Safety for the baby" (weight=10) and "Less stress for the baby" (weight=9) as the highest weighted items. No Boolean constraints were used.

- **"Pediatrician"** – Nine (9) pediatricians in the Washington, DC area (e.g., "Dr. Scott D. Wissman, MD") are compared using a weighted sum of five (5) attributes with "Patient satisfaction" (weight=10) and "Years of experience" (weight=9) as the highest weighted items. No Boolean constraints were used.

- **"Baby girl name" –** Eight (8) female baby names (e.g., "Bianca", "Jennifer") are compared using a weighted sum, with "No one notorious has same name (serial

[12] The model titles in quotation marks are quoted verbatim from the text participants entered into the form. The participant who created the model for choosing a dishwasher did not enter a title.

killer, thief, etc.)" (weight=50) as the highest weighted. No Boolean constraints were used.

- **"New home"** – Ten (10) condominiums in Laural, MD and Beltsville, MD (e.g., "9408 Nicklaus Ln #88, Laurel, MD 20708") are compared using a weighted sum of four (4) attributes with "driving time to University of Maryland, College Park 20740" (weight=4) as the highest weighted. No Boolean constraints were used.
- **"Buying a new laptop"** – Three (3) laptop computers (e.g., "ASUS X750JA-DB71 17.3-Inch Laptop (Dark Gray)") are compared using a weighted sum of features with price (weight=-50) as the highest weighted. In addition, Boolean constraints were used to require a screen size of at least 13 inches, a price of less than $1500, and an average buyer rating of at least 4 stars.
- **"Which car to purchase"** –Three (3) car manufacturers ("Honda", "Toyota", and "Hyundai") are compared by price and reliability, with a Boolean constraint that a reliability rating must be at least 4 out of 5 stars. This participant seemed distracted. The model was much less detailed than the others.
- **"Dog Breeds"** – Fourteen (14) dog breeds (e.g., "Siberian Husky", "Shibu Inu") are compared using a weighted sum with "known for intelligence/trainability" (weight=5), "typical height and length in inches" (weight=5), and "known for being affectionate and loyal" (weight=5) as the highest weighted. In addition, Boolean constraints were used to require a typical height of between 18 and 30 inches and a typical weight of between 30 and 60 pounds.

- **"Car rental"** – Twenty (20) car rental arrangements (e.g., "Fox – San Jose Airport (Airport Code: SJC)") are compared using a weighted sum with "Pick-up car rental location offers one-way rental to airport code SNA" (weight=1.0) as one the highest weighted, although all but "Availability on Saturday OR Sunday OR Monday" (weight=0.5) were weighted as 1.0.

- **Dishwasher** – Nine (9) dishwasher models (e.g., "Asko D5634XLHS") are compared using only Boolean constraints and no weights. For example, "has stainless steel door" must be "Yes".

- **"Shaver"** – Ten (10) shavers (e.g., "Wahl Professional 8-61") are compared using a weighted sum with "close shave" (weight=6) and "cordless" (weight=5) as the highest weighted.

All of these models were about real-life scenarios, some of them recently past and some of them prospective. In most cases, I used the pediatrician example (section 4.2.1) to explain how it works. At least three of the participants were previously familiar with my work.

Some participants strongly tended toward weighted sums (for negotiable criteria). Others strongly tended toward constraint-based decisions (for strict requirements), seemingly independent of the subject matter..

One participant found it unintuitive to enter weights without a preset range, noting that it was most natural to have the weights add up to a total of 100. Other participants found weights from 1-5 or 1-10 to be natural.

AskSheet is agnostic to the content of the model, so in order to minimize some attributes (e.g., price) while maximizing the rest, it is necessary to enter a negative weight.

One participant found the reasoning to be non-obvious. Another said it was odd, but easy to follow.

While this information is not generalizable beyond these participants, it suggests that future decision interfaces might give users options for how to specify the relative importance of attributes, including the range and the direction (minimize or maximize).

It should be emphasized that this was not designed as a controlled study. The primary goal was to accumulate a collection of illustrations that outline some of the capabilities of the system. The usability of the system was not a target of investigation. While this interface was sufficient for purposes of the study, I can envision future interfaces that could give users greater flexibility and be usable without any training or explanation.

This study demonstrated that some users are indeed able to describe their real decision problems in a way that can be imported into AskSheet. The models demonstrated that some users are even able to create plausible data-driven decision models about deeply personal topics such as childbirth and baby naming. I do not expect anybody would use AskSheet or any other crowd system to make such decisions exclusively, the models demonstrate that such problems have significant data-intensive subproblems that could possibly be aided by offloading some parts of the task to crowd workers.

4.5. Summary of field studies

The studies in this chapter do not answer every possible question about AskSheet, but they validate some key characteristics. First, AskSheet can be used to coordinate workers to efficiently acquire inputs that are then used as input to a spreadsheet model. That was the primary goal.

In addition, the survey of decision problems supplied a set of decision problems that illustrate some of the areas in which crowd workers could be used to gather decision inputs. The modeling study elicited much greater detail about participants' decision problems, demonstrating that users are able to describe even highly personal decision problems such as child birth and naming in terms of attributes and alternatives, and in a form that AskSheet is able to work with.

Chapter 5. Enumerating alternatives

The examples in the preceding chapters all assume that the user of AskSheet will be responsible for entering the alternatives them from some existing list. However, in many cases, this is not the case for people making decisions, such a list is not available. Listing the alternatives is an important part of decision input acquisition, although accomplishing it with crowd labor demands a different approach from AskSheet.

In the case of AskSheet, the number of rows and columns is fixed at the time the user imports the spreadsheet. For each specific cell, there is an understanding of what data is supposed to be entered—a right answer. When making a list, the number of items is not known in advance, and they might come in any order. For example, in the problem of choosing a car, how many types of car are available for sale in the US? If the list were in a spreadsheet, what should cell A12 (the 11th car) be? Clearly, coordinating crowd workers to enumerate the alternatives demands a different kind of system support.

This section focuses on the problem of creating lists. While it was motivated my work with AskSheet, I have considered a wide range of subject matter. Consider the following tasks:

- *List all researchers actively involved in online learning.*
- *List all news articles about the ethics of GMO foods.*
- *List all donut shops in the San Francisco Bay Area.*
- *List all health departments in the US, whether at the city, county, or national level.*

In each of these cases, the number of list items is not known in advance. Furthermore, creativity and knowledge are needed to find a good of web sites to check, and the right search terms, filters, and so on, in order to have the best chance of finding most/all of the items.

A naïve approach to this problem would be to post a task asking each worker to enter a certain number of list items. Then, the items could be deduped using computational techniques from the domain of entity resolution and text processing, or even crowd-based techniques (Wang, Kraska, Franklin, & Feng, 2012).

The limitations of such an approach are obvious. Since each worker would be approaching the problem from the beginning, they would likely start by doing many of the same searches and checking many of the same sites. The diversity of workers would lead to some variation, but there would be little to force them to think of less obvious sources to draw on. Thus, the resulting list would contain a large amount of duplication.

Even if every worker checked different sources, there would still be duplication due to the same item listed in different places. Variation in spelling and synonyms mean that eliminating the duplication after the items are entered would be a computational challenge in itself. Rather than resolve those after the fact, I have designed a list entry task that allows workers to collaborate on choosing sources and entering items.

The process of making such a list individually would generally involve viewing several sources that list related items (i.e., web sites, search results, documents, etc.), and filtering for the items that match the exact criteria needed. An individual would not enter duplicate items because she would remember what was entered before. Also, she would

have an incentive to visit a diverse set of sources in order to maximize the number of unique items obtained for the effort.

Relay is a general system I have built for coordinating paid crowd workers on Amazon Mechanical Turk to create lists. It aims to translate some of these individual strategies into an interface that enables a group of independent workers to achieve similar results. Although I opted to develop Relay as a standalone application, not directly integrated with AskSheet, it demonstrates an approach to this aspect of decision input acquisition so that future researchers could build from the two ideas.

5.1. Background

This work builds on the foundation of several related projects. Zhang et al introduced the idea of crowdsourcing in the presence of global constraints in the domain of travel planning (Zhang et al., 2012). They used a simple autocomplete to reduce duplication of inputs, but it was not a main focus of the work.

Creating ranked lists from search results has also been explored by Parameswaran in the DataSift system (Parameswaran, Teh, Garcia-Molina, & Widom, 2013). In that system workers formulate queries for a particular search engine and then filter them by some given criteria. However, since the items are coming from a single data source, the issues I am most concerned with—diversity of sources and deduplication—are not relevant to DataSift. Two recent efforts have used a database interface to express a range of database-like operations on an existing dataset (Franklin et al., 2011; Marcus et al., 2011b). On the surface, some of the simpler queries could resemble specifications to Relay—selecting all "records" meeting certain criteria. However, they operate on existing data.

Wang et all have used workers on Mechanical Turk for deduplication of existing datasets, in the context of entity resolution. However, they did not consider methods for minimizing the initial input of duplicates (Wang et al., 2012).

The task of list-making can be thought of as a form of collaborative search. Collaborative search has been well-explored for collocated searchers (Amershi & Morris, 2008; Morris & Horvitz, 2007; Morris, Lombardo, & Wigdor, 2010), but not in the context of crowd workers collaborating (i.e., building off each other's contributions) from different locations.

The process of making a comprehensive list could be a precursor to creating a taxonomy, a task that has recently been covered by the Cascade system (Chilton, Little, Edge, Weld, & Landay, 2013).

5.2. Relay

To start the process of creating a list, the requester enters a short noun phrase that the system uses in the worker interface, and a set of fields, which become the column headers in the worker's interface, as well as the output data (Figure 28).

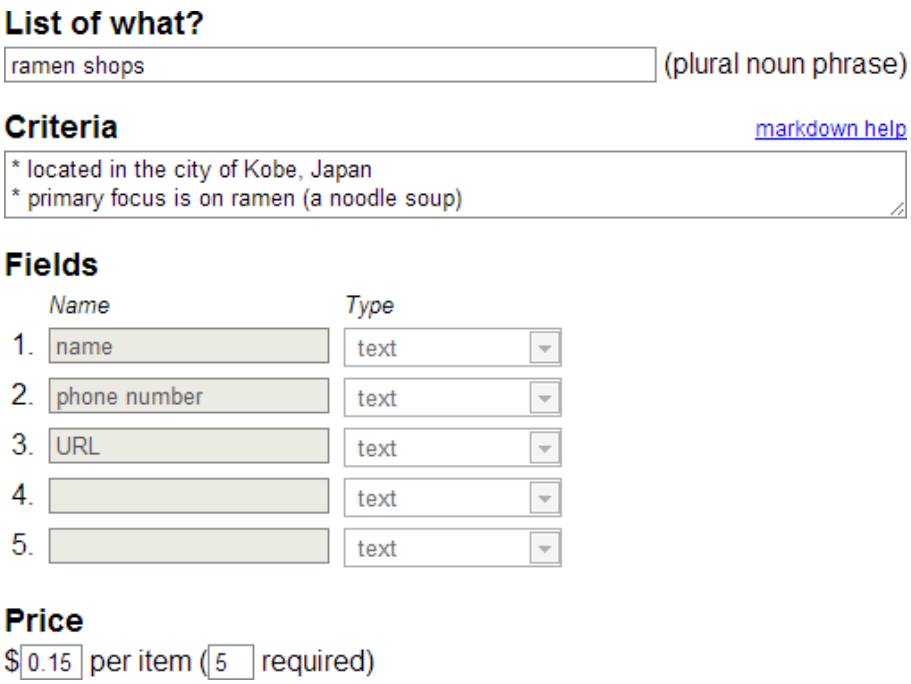


Figure 28. Job specification interface

Workers decide on a source from which to find list items. They are asked to specify exactly where and how they found the items, so that the list items can be associated with their source (Figure 29). That allows for verification of the work (to discourage cheating) and also may be useful information for an end-user.

Another reason for asking workers to specify their sources is so that if a source contains more items than one worker wants to enter at one time, he may stop part-way through and leave the rest for a future worker. In that case, the worker gives some brief instructions at the end about how to continue and how much work remains (Figure 30).

This enables an alternative to finding a new source: Workers can also choose from such existing sources that have been started by prior workers. The workers can continue extracting list items from a source, taking turns, as needed until the source has been exhausted (and hence the name *Relay*).

As a worker enters list items, the system constantly checks the server for possible duplicates, based on the text entered so far in a particular row. An existing result is suggested if each of the fields in the current input row is a subset of an existing record, irrespective of spacing and punctuation.

5.2.1. Real-time collaboration

If the worker finds that something he is entering is already in the list, he can simply click the link to have it automatically fill the input row (Figure 31). This reduction in work gives the worker an incentive to use the feature. It also ensures consistent spelling of the matching fields, so that it can easily detect duplicates or related items programmatically.

This simple mechanism represents a step toward enabling implicit *real-time collaboration* between the workers. If worker A types "Abcdef" in the first cell of a row and workers B begins typing "Abc" just a minute later, worker B will be able to benefit from the work worker A has just done. This can go both ways. Each worker's browser constantly keeps the server updated with everything they have entered so that the automatic suggestions can benefit from everyone's contributions.

Note that this notion of "real-time" crowdsourcing is distinct, and probably complementary to the real-time crowdsourcing introduced by Bernstein et al (Bernstein, Brandt, Miller, David, & Link, 2013), which minimizes the response time after a requester posts a task. That time is not a goal of Relay.

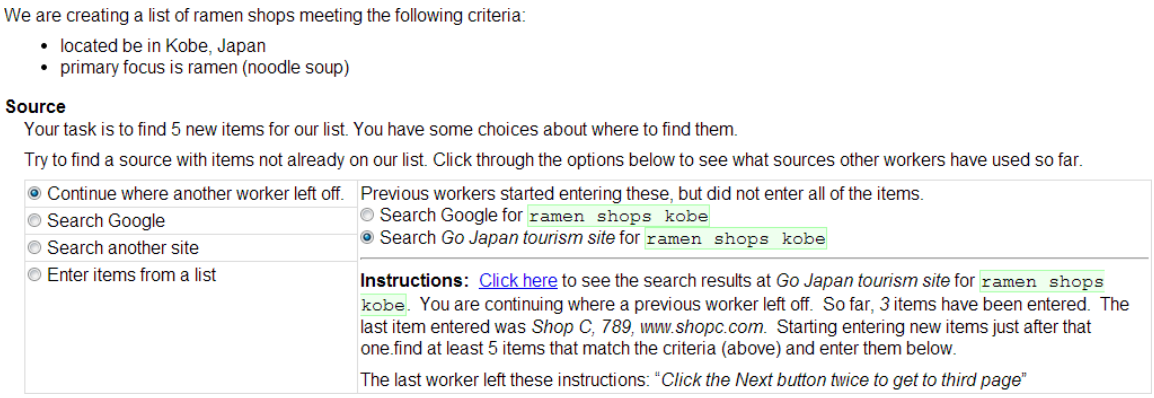


Figure 29. Source specification interface

5.3. Implementation

Relay is implemented as a web application coded in Python and JavaScript. To ensure performance of the automatic filtering, that data is kept in memory at all times.

The entire process continues until a target list size (specified by the requester) has been reached. Until then, the system automatically posts new tasks as results are received, so that there are always a few tasks in progress at any given time.

The price of a task is calculated based on a per-item price. Workers may continue beyond the minimum required number of list items, and they will be paid the remainder as a bonus in the Mechanical Turk system.

In addition to a price per item, the requester also specifies which fields are keys (always unique for a given entity) and which fields are components of a compound key (the combination always unique). When the process finishes—whenever the requester (user) chooses to stop it—Relay uses these keys to group entries that are deemed to be the same entity based on that guidance from the requester.

To generate the suggestions, the JavaScript code periodically sends the current state of the input form back to the server. The server checks all existing entries for that form

(even if not yet submitted) and matches any row where every field in the user's form row is a prefix of a field in an existing row, ignoring punctuation, whitespace, and empty fields.

The system does not incorporate traditional quality management techniques (Ipeirotis et al., 2010). With Relay, the goal is not an accurate answer to some question, but simply an enumeration from a diverse set of possible list items. For most of the list-making applications I have considered, a requester inspecting the list would likely be effective enough that I consider more sophisticated measures optional. Also, workers are required to specify exactly where they found data, so spot-checking can be effective. At any rate, quality has not been a focus of our efforts and is not intended as a contribution.

Whenever the user (requester) views the results, Relay automatically dedupes collapses perfectly duplicate rows, and groups near-duplicates. A row will be perfectly duplicate if the worker has used the suggestion to automatically fill in an existing entry. In addition, during the setup phase of Relay, the user can specify any field as a *key* and/or a *key component*. Any two rows having any key the same or all key components the same will be reduced to one.

5.4. Field trials with Relay

I have tested Relay on a few examples:

- ramen noodle shops in Kobe, Japan
- news articles and blog posts about food safety from 2010 to 2013
- professors who do research about online education

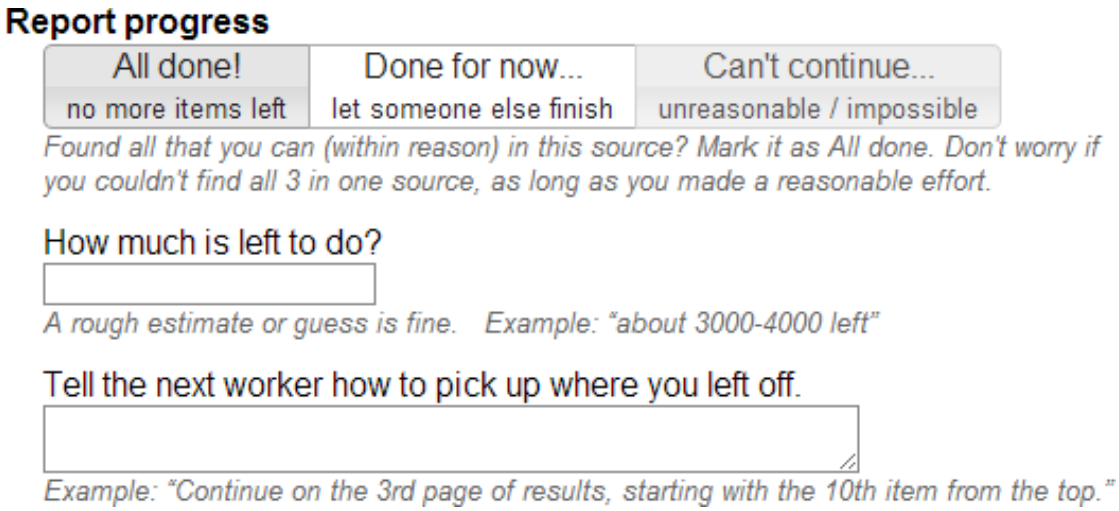


Figure 30. Giving instructions to the next worker

I ran the system on these three problems for about three hours. In the table below, *"gross"* refers to the total number of entries including duplicates, *"net"* refers to the number of entities after Relay's automatic deduplication, and *"good"* refers to the number of truly unique entities, as determined using a thorough manual check.

	yield		***cost***	***hourly rate***
ramen shops	entered	105	$0.15 / entry	$6.95 / hour
	net	71	$0.28 / good item	
	good	69		
professors…	entered	155	$0.20 / entry	$6.91 / hour
	net	137	$0.27 / good item	
	good	131		
articles…	entered	661	$0.20 / entry	$13.93 / hour
	net	434	$0.32 / good item	
	good	420		

This evaluation shows that Relay is able to elicit lists of varying sizes for arbitrary topics. However, I did not investigate that specific sources of duplication, or the degree to which the various features of Relay contributed to avoiding duplicates, encouraging duplicates that can be automatically detected, and promoting completeness (through a variety of sources). Since Relay is only a minor component of my dissertation, I consider such an in-depth evaluation to be out of scope.

5.5. Summary

Relay represents a step toward crowd-powered systems that enumerate new datasets rather than operating on existing ones. I believe the task of list-making is one that will have practical benefits for consumers, businesses, researchers, and other types of users and organizations. It might even be applied as a first step in a literature review in an unfamiliar area.

The examples mentioned in this chapter are a scant subset of the kinds of lists that people make—and that could be generated using Relay. Many lists have a fundamental hierarchy embedded. For example, to list all of the health departments conducting restaurant inspections in the United States, one would need to check at the state, county, and city levels. In some cases, a county may handle all inspections for their area, except for select cities. In the future, I could envision augmenting Relay with more explicit support for such search processes.

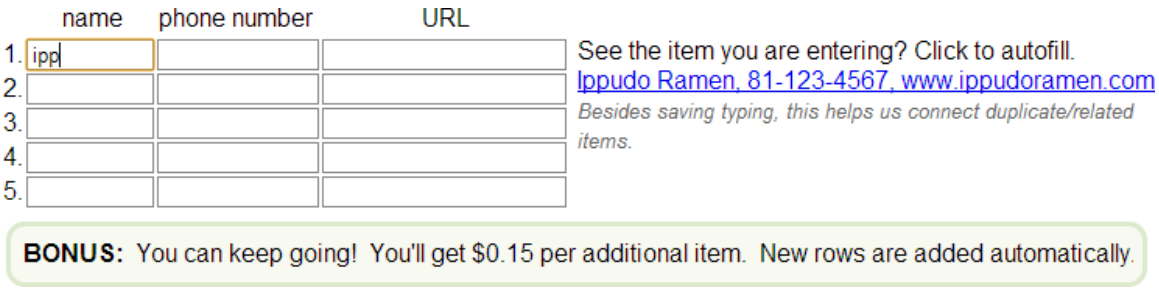


Figure 31. Prior list items are shown beside the editing interface. Clicking these saves the worker keystrokes and enables us to connect duplicate or related entries since they will have same spelling.

Beyond this task, I believe there is significant potential for bringing crowds together in ways that preserve the traditional independence of the crowd workers, while also enabling them to build off one another in richer ways that just iterative improvement or *n*-tier review processes.

Relay is not directly integrated with AskSheet. However, it could be used to create a list of alternatives in the course of creating a model to be used with AskSheet.

Chapter 6. Discussion

The field trials in section 4.2 (page 73) demonstrated the efficiency of AskSheet for delegating the acquisition of inputs for selecting a car, a smartphone, and a doctor, while the modeling study in section 4.4 (page 86) showed the applicability of the approach to an even wider variety of subject matter that users are able to model. However, even these span a significantly more constrained problem domain than what I initially envisioned.

When I began this work, I had expected the method could be applied to more complex institutional decisions, such as screening job applicants, selecting materials for a museum exhibit, or deciding what products a department store should offer in the coming season. Since I have significant personal experience making models for such decisions, I felt confident that any criteria can be modeled.

Through the process of designing, implementing, and testing my implementation, AskSheet, I have come to understand the interactions between some inherent aspects of working with crowds, computational problems, and the limits of explicit modeling of decisions (i.e., in spreadsheets).

6.1. Premise: data-driven analytical decisions

Before proceeding, it should be noted that this work is inherently intended for decisions for which the primary task is to gather and analyze information about the alternatives to determine which best conforms to some pre-established criteria. Thus, it excludes gut decisions and spreadsheets that serve only to store and display information to

help a user arrive at a gut decision. It also excludes decision models that primarily exist to simulate the dynamic properties of a situation.

I have focused primarily on consumer decisions. However, this method could also include business decisions, such as choosing a location for a new office, a venue for an event, or identifying job applicants who meet the criteria. For business decisions, a record of the rationale is often desirable. Some individuals also prefer to make decisions in this way, especially if there will be long-term ramifications.

6.2. Limitations on applicability

This section will discuss the factors that influence the applicability of AskSheet to various types of decision problems. I will focus primarily on general issues that affect not only my implementation (AskSheet), but other future systems for delegating decision input acquisition to crowd workers. For simplicity, I will assume that the notation is always a spreadsheet, even though other notations are possible.

6.2.1. Value: Effort to delegate (or not)

In general, when deciding whether to delegate work, a key consideration is the effort to delegate versus the effort to not delegate (i.e., to do the work yourself).

In this context the effort to delegate includes the following:

- **Create an explicit model of the decision.** Although we may think we know what is important and other aspects of our decision logic, creating an explicit model takes an extra level of effort. This effort is greatest when the criteria are nuanced, complex, or difficult to express in spreadsheet notation. This will also depend on the decision maker's level of comfort with spreadsheet formulas.

- **Write unambiguous instructions.** Writing clear instructions is always a challenge in any domain. Creating instructions for Mechanical Turk can be especially challenging because as a requester, I do not know the background of the workers who will read and interpret the instructions. If workers misunderstand, it will influence the results. It takes extra effort to author instructions that are specific enough to avoid ambiguity and yet concise enough to suit workers' desire to minimize the overhead of reading them. This effort can be especially great for scenarios that require some domain knowledge (e.g., knowing what the "wheel base" of a car means).

The effort to *not* to delegate normally entails the following:

- **Search for information (alone).** For decisions that depend on sorting through a lot of information or if the information is spread across many sources, this may be onerous.

- **Analyze the information.** This could be done mentally, using pen and paper, a spreadsheet, or other tools, depending on the type and quantity of data.

The above factors influence whether it would be worth the effort to delegate the information acquisition part of a decision process to crowd workers using a model-driven system like AskSheet—instead of the decision maker doing that work alone. These factors do not restrict what *can* be done.

6.2.2. Performance: time to compute value of information

Perhaps the greatest limiting factor affecting the applicability of the AskSheet method is the time to compute the priorities. For purposes of this discussion, I define "performance" as the practical capability to perform a computation (calculating the value of information for each of the inputs).

Unlike most of the other factors mentioned in this section, this is largely implementation dependent. However, as I will explain below, some of the calculations that would be needed for my implementation to support more interesting models have no known polynomial time algorithm.

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

The running time for the prioritization depends entirely on the formulas in the model. Even for the weighted sums model, a perfect expression of the complexity is elusive because it depends on the number of possible values in the output distribution of each `=SUM(…)` function.

Consider the absolute simplest weighted sums model: identical operands each depending on one identical request and no weights.

	A	B	C
1	=ASK("1 to 6")	=ASK("1 to 6")	=SUM(A1:B1)
2	=ASK("1 to 6")	=ASK("1 to 6")	=SUM(A2:B2)
3	=MAX(C1:C2)		

In order to calculate the distribution for the `=MAX(…)`, it must first calculate the distribution of each `=SUM(…)`. The number of distinct values in the output distribution

is the factor that will influence the running time of the `=MAX(…)` so that is what we care about. For this model, it can be thought of as the number of unique sums from rolling two 6-sided dice. One could imagine asking workers to role dice and enter the values into a form. From two dice, there are 11 possible sums. For 100 dice, there are 501 possible sums.

In this good case, the number of distinct sums is $bc - b + 1$, for b=«number of operands» (i.e., dice) and c=«number of possible values per operand» (e.g., sides per die). That corresponds to a running time of $O(c^2b^2)$ to calculate the output distribution of a `=SUM(…)` function. (Each of these claims is proven in Appendix C.7 on page 135.)

In terms of the weighted-sum model, let a=«number of alternatives», b=«number of attributes», and c=«number of possible values for each input».

Since a weighted-sum model has one `=SUM(…)` per alternative, the above must be repeated once for each alternative. To calculate the utilities, it must calculate this repeatedly: once for each request that the `=SUM(…)` depends on (supposing that request were acquired next) × once for each possible value of that request (supposing the selected request received that value). This is because it considers the effect if each request were chosen next, but the utility must be averaged over the effect of each possible value of the request.

There are ab requests each having c possible values, so we have a running time of $O(a^2b^2c^3)$ to calculate the output distribution for each of the sums in the model. This is the dominant factor, so this is the running time for calculating the prioritization. Using the optimization outlined in Appendix C.7.3 (page 141), this can be reduced to $O(a^2b^2c^2)$

by calculating the `=SUM(…)` output distribution once for each register instead of once for each request × each value of the request.

To understand the running time more practically, I 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 32.

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. The prioritization cannot be calculated until the data the worker is submitting has been entered into the model. Thus, it would not be possible to run them while the worker is working.

		number of attributes				
		5	***10***	***15***	***20***	***25***
number of alternatives	***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 32. Time (secs.) to prioritize a benchmark model based on the sum of 5-star ratings for each alternative.

To reduce the recalculation time, it 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, I am aware of several untapped opportunities for optimizing the code. Using dynamic programming, I previously improved performance of calculations for the `=SUM(…)` operation. I believe a similar approach could be used for other operations.

For some operations, I 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, one possibility would be to approximate the probabilities by random sampling. However, this is complicated by the large number of parameters. I have not attempted to implement this.

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.

6.2.3. Efficiency and suitability: Expressiveness of model

In this case, I define efficiency as the savings from culling inputs (determining that they are not needed) relative to the potential total cost if it were to acquire all inputs. If all inputs have the same cost, then it is the proportion of inputs that were culled (determined

not to be needed). This affects the degree to which the AskSheet method could outperform existing systems such as SmartSheet, which gather all of the inputs.

Suitability is the degree to which the decision result would likely reflect the user's actual values about the decision relative to the actual possibilities in the world (i.e., did it actually pick the best one). While this would be difficult to measure, there are some factors that clearly have an effect. Recall that AskSheet is actually just calculating a value for the root cells in the spreadsheet; it is agnostic to whether the spreadsheet is about a decision or something else. None of these factors influence the accuracy of that result. However, if the system cannot support models that are expressive enough to discern between an excellent alternative and an ideal one, then ultimately the user may miss the ideal, settling for mere excellence.

I would not expect the AskSheet method to achieve the same level of efficiency that an individual would achieve doing the work alone. There are several reasons for this, but center around the expressiveness of the model. An individual decision-maker has the benefit of reliable tacit knowledge of the situation and her criteria. These issues can be summarized as follows:

- **Model is typically an imperfect representation of criteria.** For the kinds of models shown in the field trials, the formulas probably do not fully express the decision maker's values about the situation. For example, in the smartphone example (page 80), perhaps RAM is only important if the camera has a large megapixel count since the RAM is primarily used for manipulating photos. Many such nuances are conceivable. Many could be expressed with more complex formulas, but creating those formulas comes at a cost.

- **Cannot satisfice—always aims for a perfect decision.** There is no allowance for accepting an approximate result (i.e., a suboptimal decision result), even if the benefit to the user of the more precise answer is far outweighed by the cost of acquiring more information. For example, even if the system found an option that results in a score that is 99% as much as that maximum possible score (e.g., a nearly perfect car), it would continue acquiring information until it could infer that no other alternative could result in a higher score.

- **Unaware of domain-specific relations (e.g., memory ~ CPU)**. AskSheet has no way of knowing if two attributes are inherently related. Consider the smartphone example again. Ideally, if it finds that one option has the minimum amount of RAM, it should adjust the probability distribution of the CPU for that alternative (e.g., model) to reflect a low likelihood of a powerful CPU. However, AskSheet (my implementation) has no knowledge of such relationships and no way for the user to specify them. While a future incarnation of the idea could provide such a facility to the user—and I considered doing so myself—it would impose a significant burden on the user, which would ultimately offset the overall value of AskSheet.

- **Bounds may be inaccurate**. The bounds specified by the user may in many cases be mere estimates of the highest or lowest values the user would expect. For example, the user may not be aware that newer smartphones come with more RAM can store up to 160 GB (total including a micro SD card). The effect is that the system might stop too soon.

Despite the above limitations, AskSheet still provides a significant increase in efficiency versus any existing technology. The only current practically useful service for gathering inputs in support of data-driven decisions in SmartSheet. As I discussed in section 2.1.1 (page 26), it allows a user to enter the row and column headings, and use Mechanical Turk to fill in the values. With SmartSheet, there is virtually no way to control the order in which inputs are acquired.

6.3. Optimality

In this discussion, optimality is the computational property of an implementation—as opposed to the more human-oriented properties above—of being able to maximize the efficiency, given an input model and settings. 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.

- **Inputs 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 below. 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.

6.3.1. 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("0 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 my 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 would be to simplify the formulas algebraically before calculating need probabilities for comparison operators. Software libraries exist for performing such algebraic manipulation (Joyner, Čertík, Meurer, & Granger, 2012).

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. More generally, for any equality operation $a=b$, instead of calculating Pr($a=b$) directly, it would convert the expression to $a–b=0$, algebraically simplify the expression for $a–b$, and then calculate the probability distribution of the reduced form of $a–b$. In the example above, the expression A1=B1 would be reduced as follows:

```
A1 - B1

A1 - (A1 * 1)

A1 - A1

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 (Cheatham, Holloway, & Townley, 1979).

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

6.4. More potential applications

In this section I outline two use cases that illustrate my vision for this project, and briefly discuss why these are not technically feasible with my implementation.

6.4.1. 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.

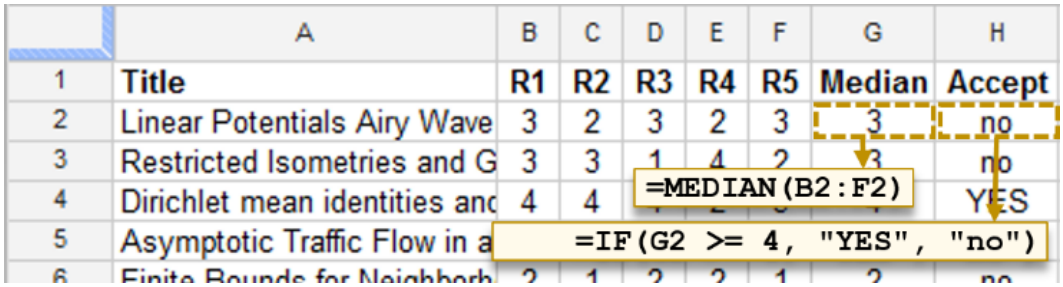
BEST CHOICE			
Destination	**Depart**	**Duration**	**Return**
Florida	3/22/2013	4	3/26/2013

The large number of combinations makes this infeasible with my 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.

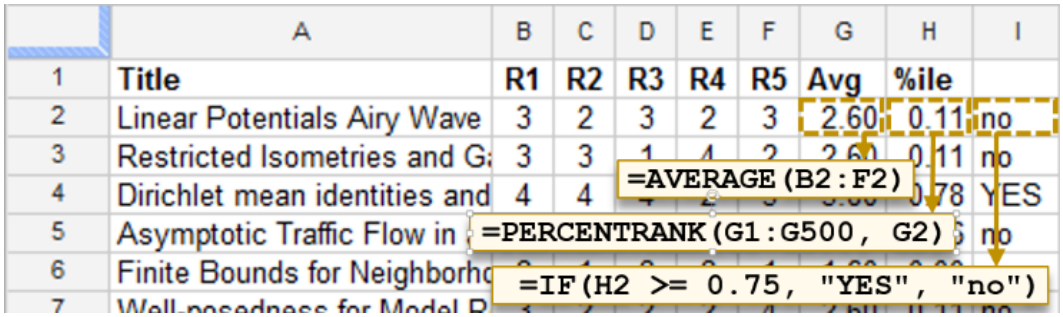
6.4.2. Example: Reviewing conference paper submissions

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

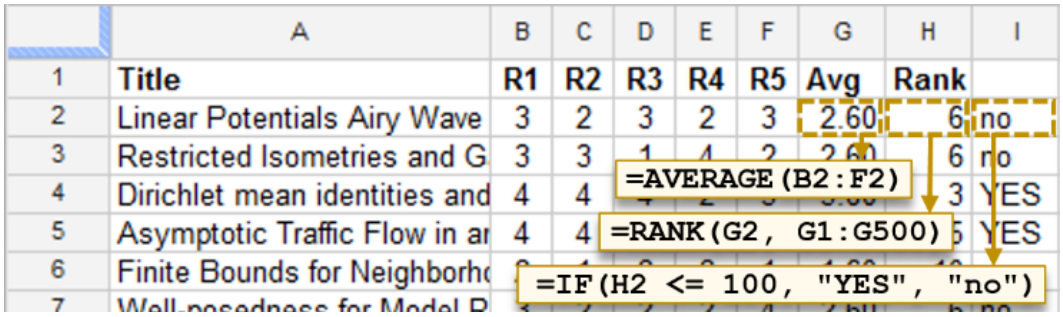
This could be modelled as accepting any paper where the median of five scores is at least 4.0 out of 5.0. If the first three are more (or less) than 4.0, then no more are needed.



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.

All of the above variants on the paper reviewing model are believed to be technically infeasible due to the inability to calculate the probability distribution of order statistics for non-identically distributed random variables.

6.5. Future work

In this dissertation, I have developed and studied a method for coordinating workers to efficiently enter data to be supplied as the inputs to a spreadsheet decision model. In the course of designing, building, and studying AskSheet—as well as the companion system, Relay—I have come to understand several important general challenges that would be valuable to address in the future.

- **Move beyond Mechanical Turk.** Although the foundation for much of the contemporary research on human computation and crowdsourcing, the semantics of working with Mechanical Turk can be constraining. For example, it is difficult to select workers with knowledge of a particular topic. The key benefit of working with Mechanical Turk is that it already has a substantial base of workers on the site every day, so it makes it easier to launch parallel workflows at any time. Thus, with a less populated labor pool, turnaround might be slower, and it might be harder to achieve the additional efficiency afforded by having parallel workflows.

- **Improve interfaces for describing the types of inputs.** Writing clear instructions takes work. For AskSheet, any ambiguity in the instructions or the

description of fields (i.e., row or column headings) can lead even the best worker to enter information that was not expected, thus resulting in a poor end result. This is a significant hurdle that was not explored in this work, but is of importance to practically all crowdsourcing projects.

- **Investigate interactions with quality control mechanisms.** As I stated in 4.1 (page 70), I chose to exclude the accuracy of the inputs—and hence the robustness of the decision result—from the scope of this dissertation. However, a true realization of the potential benefits of crowd labor for decision input acquisition would naturally require a reliable means of ensuring that the inputs are correct so that the outputs can be depended on. This largely depends on the ongoing research in the area of quality management. However, to the extent that gathering multiple judgments affects the ability of a system to efficiently coordinate workflows, that interaction should be explored, as well.

- **Develop methods for calculating value of information on order-related functions of random variables**. For example, if there were a tractable algorithm for the rank or median of a random variable within some array, AskSheet could be applied to a richer set of models, such as those sketched in section 6.4 (page 115). I attempted to solve these problems directly and by approximation but ultimately found that they are open problems in mathematics (Glueck & Karimpour-Fard, 2008).

- **Apply similar methodology to more robust decision methodologies.** For example, the analytical hierarchy process generally provides better quality

decision results than weighted sums models. However, calculating a decision result with AHP entails the solution of eigenvalue problems, which is well beyond the mathematical capabilities of AskSheet.

- **Develop robust methods of eliciting non-numeric values that are inter-consistent.** The method of text scoring and reconciliation that I introduced in section 3.4 (page 63) was essentially a stopgap to allow me to proceed with core of my work.

- **Enable the machine to infer the implicit relationships between attributes. (i.e., CPU is high when memory is high) or else directly elicit them from the user.** For small models run in isolation, this would be difficult to achieve without increasing the cost beyond a comfortable level. One solution would be to develop a future system that would track prices and other details about the alternatives across multiple models, and estimate these associations through automated inference. Alternatively, a future system could have an interface for the user to specify such associations based on tacit knowledge of the domain.

- **Develop labor markets—or new ways of tapping into existing ones—that make it easier to tap into workers' existing domain knowledge.** A persistent challenge when writing instructions for crowd workers is that it is difficult to predict what they will know. For example, when creating the smartphone example, I had to take care that the units for RAM would not be misconstrued. If I am expecting gigabytes and the user enters 512 (megabytes), that alternative may incorrectly end up as the decision result (winner).

Tackling these challenges would have value not only for AskSheet but for other efforts in the realm of crowdsourcing, decision support, and probabilistic computation.

Appendix A. Spreadsheet formula grammar accepted by AskSheet

This grammar describes the set of formulas that AskSheet supports. The functions accepted are a small subset of what major spreadsheet applications accept. Also, it does not support array formulas, cross-sheet references, or array literals (e.g., `{1, 2, 3}`). However, it has been sufficient to enable a wide variety of spreadsheet models to be run with AskSheet.

```
cell        ::= ''
              | number
              | str_bare
              | formula
              | 'ASK(' (cmp_expr ','){0,5} cmp_expr ')'
              | 'HYPERLINK(' expr ',' expr ')'
number      ::= '-'? [0-9]+ ('.' [0-9]+)?
str_bare    ::= [^=] .*
str_quoted  ::= '"' ([^"]|'""')* '"'
formula     ::= '=' expression
cellref_a1  ::= '$'? [A-Z]+ '$'? [1-9][0-9]*
cellref_rc  ::= 'R' rc_index 'C' rc_index
rc_index    ::= [1-9][0-9]*
              | '[' [1-9][0-9]* ']' | ''
cmp_expr    ::= (add_expr cmp_op)? add_expr
cmp_op      ::= '<' | '>' | '<=' | '>=' | '<>' | '='
add_expr    ::= (add_expr ('+' | '-'))* mul_expr
mul_expr    ::= (mul_expr ('+' | '-'))* fun_expr
fun_expr    ::= 'MIN(' arg_list ')' | 'MAX(' arg_list ')'
              | 'SUM(' arg_list ')' | 'NOT(' expr ')'
              | 'AND(' arg_list ')' | 'OR(' arg_list ')'
              | 'INDEX(' expr ',' expr ',' expr ')'
              | 'MATCH' expr ',' arr_expr ',' expr ')'
arr_expr    ::= cellref_rc ':' cellref_rc
              | cellref_a1 ':' cellref_a1
              | pri_expr
pri_expr    ::= '(' expr ')'
              | cellref_rc  |  cellref_a1
              | str_quoted  | number
arg_list    ::= (arg_list ',')* expr
```

Figure 33. Full grammar of spreadsheet formulas accepted by AskSheet

Appendix B. Algorithms for task generation and management

Chapter 3 (page 48) described the implementation of AskSheet in terms of the components of the system and some specific aspects that are central to the contribution of this dissertation. This appendix augments that chapter with explicit algorithms for the key operations in AskSheet. The algorithms for calculating utilities—generally more mathematical—are covered in Appendix C (page 129).

The Python-like pseudocode used to describe the algorithms is not the same as the AskSheet source code. Each of the functions below was specifically written for purposes of illustration. These deliberately omit details that are not core to understanding the process, such as database access, multiple judgments per input, web application logic, worker tracking, payment, models with multiple root cells, certain edge cases, caching, optimization, and fine tuning for consistency between runs. The names used here do not necessarily match the real code.

The actual AskSheet implementation consists of about 29,000 lines of my code (83% Python, 10% JavaScript, 5% HTML-Python templates, 2% CSS), 199 Python classes, and two SQLite databases containing 29 tables. This includes the CrowdLib library, which I created to simplify working with the Mechanical Turk API. Since many details of the implementation are routine programming, I have endeavored to distill out the core algorithms that enable AskSheet to coordinate workers to efficiently gather inputs to decision spreadsheets.

B.1. Refreshing HITs

When a model is first started, AskSheet calculates the utilities for each request and then enables or disables HITs, as needed.

```
def refresh(model):
    # model ------ object containing spreadsheet formulas

    utilities = get_utilities(model.root) # p. 130
    # returns a dictionary {request:utility}

    batches_by_role_name = make_batches(model, utilities)# p. 125
    # returns a dictionary {role_name:[request, request, …], …}

    enable_or_disable_hits(batches_by_role_name) # p. 127
    # Decide which roles should be available for workers to work
    # on (or not), and then post or cancel HITs, as needed.
```

B.2. Receiving inputs

When a worker submits a form, the inputs are saved and the prioritization is recalculated before refreshing the HITs, as before. All of this is done in real-time when the worker presses the *Submit* button on the form. The reason is that AskSheet is designed to minimize the number of HITs posted at a given time, to avoid having more workers gather inputs that may not turn out to be necessary.

At its core, the procedure for receiving inputs is simple:

```
def on_input(inputs, model):
  # inputs -- a dictionary of {addr:v, ...} where `addr` is the
  #           cell address of a request (e.g., "B16") and `v`
  #           is the value that was entered by a worker.
  for addr,v in inputs.items():
    request = model.requests[addr] # instance of AskFunction
    request.value = v

  refresh(model) # p. 124
```

The actual implementation has standard logic for aggregating multiple judgments, either by consensus (i.e., vote of *k* workers) or by averaging. For multiple judgments, a value is only assigned to a request once enough judgments have been received.

B.3. Assembling the input form

When a worker accesses a HIT via the Mechanical Turk web site, an input form provided by AskSheet is displayed inside an `IFRAME` element. The selection of which requests to display in the form is delayed until the moment it is accessed. With multiple roles, it is possible that inputs from a worker on one role may change the priorities and alter which requests should be shown for another role.

```
def make_form(model, role_name)
  # model ------ object containing spreadsheet formulas
  #
  # role_name -- as passed to ASK(…) functions; in this case it
  #              is referenced by the CGI parameters when the
  #              worker accesses the IFRAME in the HIT.

  utilities = get_utilities(model.root) # p. 130

  batches_by_role_name = make_batches(model, utilities) # p. 125

  top_batch = batches_by_role_name[role_name][0]

  return render_form(top_batch)
  # render_form(…) is not shown in this document.
```

B.4. Making batches

Each form that a worker accesses consists of a set of requests. The number of requests is set by the decision maker in the setup screen. The algorithm for forming batches represents a tradeoff between strict adherence to the priority order versus the need to allow workers to gather inputs in an order that is efficient with respect to the web sites

that must be visited. For example, gathering four specifications about the same car model would probably be more time-efficient than gathering the same attribute about four different cars.

```
def make_batches(model, utilities):
  # model ------ object containing spreadsheet formulas
  #
  # utilities -- dictionary {r:u} where `r` is a request
  #              (instance of AskFunction) and u is the expected
  #              savings if `r` were acquired next
  #
  # Returns ---- dictionary {rn:batches} where rn is a role_name
  #              (as passed to ASK(…)) and batches is a list of
  #              lists of requests (instances of AskFunction),
  #              sorted by the sum of the request utilities

  # Group requests by role.
  buckets = {role_name:[] for role_name in model.role_names}
  for r in model.requests:
    buckets[r.role_name].append(r)

  # Initialize empty structure
  batches_by_role_name = {rn:[] for rn in model.role_names}

  # Form the batches
  for role_name in model.role_names:
    batch_size = model.role_settings[role_name].batch_size

    # Temporarily sort requests by decision alternative
    # (itemLabels) first, and then by utility (descending order).
    requests = buckets[role_name]
    sort_fn = lambda r:(r.item_labels, -utilities[r], r.row, r.col)
    requests.sort(key=sort_fn)

    # Create batches
    for i in range(len(requests)):

      # Start a new batch, if needed.
      if i % batch_size == 0:
        batches_by_role_name[role_name].append([])
```

```
      # Add the i'th request to the current batch.
      batches_by_role_name[role_name][-1].append(requests[i])

    # Sort batches by the sum of the utilities of their requests
    batch_key_fn = lambda batch:sum(-utilities[r] for r in batch)
    batches_by_role_name[role_name].sort(key=batch_sort_fn)

  return batches_by_role_name
```

B.5. Enabling and disabling HITs

AskSheet is unlike most applications of Mechanical Turk in that instead of posting many related HITs at once, AskSheet posts a minimal number of HITs at a time. When the decision maker has requested only one judgment per input—the assumption throughout this appendix—this means that AskSheet allows only one HIT for a given role to be active at any given time. That prevents other workers from entering inputs that may turn out to be unneeded.

```
def enable_or_disable_hits(utilities, batches_by_role_name):
  # utilities ------------ (same as parameter to make_batches)
  #
  # batches_by_role_name -- (as returned by make_batches)

  COMBINED_UTILITY_PERCENTILE_THRESHOLD_TO_ENABLE_FORM  = 0.80
  COMBINED_UTILITY_PERCENTILE_THRESHOLD_TO_DISABLE_FORM = 0.20

  role_names = batches_by_role_name.keys()

  # Get all batches
  batches = [batch for role_name in role_names
                   for batch in batches_by_role_name[role_name]]

  # Get all batch utilities, sorted in ascending order
  batch_utilities_all = [sum(utilities[r] for r in batch)
                                          for batch in batches]
  batch_utilities_all.sort()
```

```
# If the max utility batch for a role is at least this much,
# and it's not already active, then enable it.
threshold_to_enable = len(batch_utilities_all) * \
        COMBINED_UTILITY_PERCENTILE_THRESHOLD_TO_ENABLE_FORM

threshold_to_disable = len(batch_utilities_all) * \
        COMBINED_UTILITY_PERCENTILE_THRESHOLD_TO_DISABLE_FORM

for role_name in role_names:
  top_batch = batches_by_role_name[role_name][0]
  batch_utility_max = sum(utilities[r] for r in top_batch)
  active = amt_manager.has_active_hits(role_name)

  if not active and batch_utility_max >= threshold_to_enable:
    # If the highest utility batch for this role is in the top
    # 20%ile of all batches (all roles), but it has no active
    # HITs, then post one to begin or resume work on this role.
    amt_manager.post_hit(role_name)

  elif active and batch_utility_max <= threshold_to_disable:
    # If the highest utility batch for this role is in the
    # bottom 20%ile of all batches (all roles), but it has an
    # an active HIT, then cancel it to halt work on this role.
    amt_manager.cancel_all_hits(role_name)

  elif active:
    # If the role is already active (i.e., a worker just
    # submitted a form), then add another HIT so s/he can
    # continue working on it.
    amt_manager.post_hit(role_name)

# The amt_manager functions are not shown in this document.
```

Appendix C. Algorithms for calculating utilities

The prioritization method used by AskSheet starts with the algorithm for calculating utilities. That algorithm depends on the algorithms for calculating need probabilities, which in turn usually relies on the algorithms for calculating output distributions (probability mass function, or PMF). These algorithms are different for every supported spreadsheet operation. In effect, AskSheet includes a spreadsheet that treats every cell as a discrete random variable. This appendix will describe these algorithms for every operation supported by AskSheet.

As in Appendix B (page 123), *the Python-like pseudocode used to describe the algorithms is not the same as the AskSheet source code.* While there is some high-level correspondence, each function has been aggressively pared down from the actual implementation code. For example, most caching logic and routine optimizations have been omitted and external helper functions inlined, unless integral to understanding the algorithm.

C.1. `get_output_distribution(…)`

The `get_output_distribution(next, value)` method of each operation calculates the probability mass function (PMF).

It returns a `Distribution` class which can be referenced as a tuple of 2-tuples (value, probability) sorted by value, but with some methods useful for calculating probabilities. For example, the output distribution of `=ASK("1 to 6")` (equivalent to

rolling a 6-sided die one time) would be expressed as `Distribution((1, 0.166), (2, 0.166), (3, 0.166), (4, 0.166), (5, 0.166), (6, 0.166))`.

The `next` and `value` parameters refer to a condition on which the probability distribution will be based. With each invocation, we suppose that the request `next` were fulfilled next and that it receives the value `value`.

C.2. `get_need_probabilities(...)`

The `get_need_probabilities(next, value)` method of each operation calculates, for each request that the operation depends on, the probability that the request will be needed to calculate the result of the operation, marginalized over all possible values of other requests.

It returns a dictionary associating each descendent request with the probability that it will be needed. For example, if cells A1 and A2 contain requests r_1 and r_2 (`=ASK(...)` formulas), respectively, and C1 contains `=IF(1>0, A1, A2)`, the need probabilities of C1 would be {r_1: 1.0, r_2: 0.0).

The `next` and `value` parameters have the same meaning as for the `get_output_distribution(next, value)` method; they refer to a condition on which the need probabilities will be based. As a rule, Pr("operation needs `next`") = 1.0.

C.3. Calculating utility of each request

The algorithm below applies to a simplified case in which the decision model has only one *root* (cell with no ancestors and one or more `=ASK(...)` formulas as descendants).

The running time is $O(ab^2)$, where a=«average number of possible values for each request», and b=«number of requests that the root depends on».

```
def get_utilities(root):
  # Return a dictionary with the relative utility of each request
  # that the given root note depends on.

  base_needs = root.get_need_probabilities(next=None, value=None)
  utilities = {}
  for r in root.requests:

    cost_if_r_next = 0.0
    for (v, p_r_eq_v) in r.get_output_distribution():
      needs = root.get_need_probabilities(next=r, value=v)
      for r_o in root.requests:
        if r_o is r:
          p_need_r_o = 1.0
        else:
          p_need_r_o = needs[r_o]
        cost_if_r_next += p_need_r_o * p_r_eq_v * r_o.cost

    base_cost = 0.0
    for r_o in root.requests:
      base_cost += base_needs[r] * r_o.cost

    utilities[r] = base_cost - cost_if_r_next
  return utilities
```

C.4. Binary operators (+ - * / = <> < > >= <=), output distribution

For addition, the distribution for the sum of two random variables is the convolution of their distributions. Using dynamic programming, this can be done in $O(|lhs.\text{odist}|\cdot|rhs.\text{odist}|)$ time, where $|lhs.\text{odist}|$ refers to the number of unique values the left-hand side could take, and likewise for the right-hand side.

The running time is $O(a+b+cd)$, where a=«time to calculate output distribution of the left-hand side», b=«time to calculate output distribution of the right-hand side», c=$|lhs.\text{odist}|$ (number of distinct possible values in left-hand side), and d=$|rhs.\text{odist}|$.

```
class AddOperator(BinaryOperator):
  @memoize
```

```
def get_output_distribution(self, next, value):
  # The interface for this method is given on page 129.
  lhs_odist = self.lhs.get_output_distribution(next, value)
  rhs_odist = self.rhs.get_output_distribution(next, value)

  odist_dict = {}  # initially represent as a dictionary
  for v_lhs,p_lhs in lhs_odist:
    for v_rhs,p_rhs in rhs_odist:

      v_result = v_lhs + v_rhs # or other operator: -, *, /

      p_result = p_lhs * p_rhs
      # Presume lhs and rhs to be indepdendent

      # Initialize to 0.0, if necessary
      if not sum_dist.has_key(v_result):
        odist_dict[v_result] = 0.0

      odist_dict[v_result] += p_result
      # Each way of reaching this result is distinct, so
      # these cases are indeed mutually exclusive.

  odist = sorted( odist_dict.items() )
  return Distribution(odist)
```

For the other binary operators (`- * / = <> < <= > >=`), the calculation is the same, but with the line `v_result = v_l + v_p` changed to use the appropriate operator instead of +.

C.5. Comparison operators (= <> < <= > >=), need probabilities

For the comparison operators, we consider a request needed if and only if either operand needs it and that operand itself is needed. To evaluate when an operand is needed, we calculate the probability that it would be needed if the other side of the operator were acquired first.

The running time is $O(a+b+c+d+e)$, where a=|*lhs*.odist|, b=|*lhs*.odist|, c=|*requests*|, d=«average time to calculate the output distribution of either of the operands», and e=«average time to calculate need probabilities of either of the operands».

```
class ComparisonOperator:
  @memoize
  def get_need_probabilities(self, next, value):
    # The interface for this method is given on page 130.

    lhs_needs = self.lhs.get_need_probabilities(next, value)
    rhs_needs = self.rhs.get_need_probabilities(next, value)

    lhs_odist = self.lhs.get_output_distribution(next, value)
    rhs_odist = self.rhs.get_output_distribution(next, value)

    lhs_values = lhs_odist.get_values()
    rhs_values = rhs_odist.get_values()

    # Calculate Pr(need lhs) and Pr(need rhs)

    # Equal or Not-Equal operators
    if self.symbol == "=" or self.symbol == "<>":
      p_need_rhs = sum(p for v,p in lhs_odist
              if v in rhs_odist.values)
      p_need_lhs = sum(p for v,p in rhs_odist
              if v in lhs_odist.values)

    # Greater-Than or Less-Than-Or-Equal operators
    elif self.symbol == ">" or self.symbol == "<=":
      p_need_rhs = sum(p for v,p in lhs_odist
              if rhs_odist.min_value >= v > rhs_odist.max_value
      p_need_lhs = sum(p for v,p in rhs_odist
              if lhs_odist.min_value > v >= lhs_odist.max_value

    # Less-Than or Greater-Than-Or-Equal operators
    elif self.symbol == "<" or self.symbol == ">=":
      p_need_rhs = sum(p for v,p in lhs_odist
              if rhs_odist.min_value <= v < rhs_odist.max_value)
      p_need_lhs = sum(p for v,p in rhs_odist
              if lhs_odist.min_value < v <= lhs_odist.max_value)
```

```
needs = {}
for r in self.requests:

  # Calculate Pr(need lhs ∧ lhs needs r)
  if lhs_needs.has_key(r):
    need_r_for_lhs = lhs_needs[r] * p_need_lhs
    # Presume "need lhs" and "lhs needs r" are independent
  else:
    need_r_for_lhs = 0.0

  # Calculate Pr(need rhs ∧ rhs needs r)
  if rhs_needs.has_key(r):
    need_r_for_rhs = rhs_needs[r] * p_need_rhs
    # Presume "need rhs" and "rhs needs r" are independent
  else:
    need_r_for_rhs = 0.0

  p_need_r = need_r_for_lhs + need_r_for_rhs - \
           need_r_for_lhs * need_r_for_rhs
  # Presume "need r for lhs" and "need r for rhs" independent

  if p_need_r > 0:
    needs[r] = p_need_r

needs[next] = 1.0
return needs
```

C.6. Binary arithmetic operators (+ − * /), need probabilities

The intuition for finding the need probabilities for binary arithmetic operators is that the operator needs a request if and only if either operand needs it. This algorithm is the same for all four binary arithmetic operators, except that in the case of multiplication and division, if either operand is always 0 (e.g., numeric literal or input that has already been acquired), then it reports needing no requests other than the request given by the `next` parameter. (AskSheet does not handle division-by-zero errors.)

The running time is $O(a+b)$, where a=«average time to calculate need probabilities of either of the operands» and b=|*requests*|.

```
class BinaryOperator:
  @memoize
  def get_need_probabilities(self, next, value):
    # Need request if either operand needs it
    lhs_needs = self.lhs.get_need_probabilities(next, value)
    rhs_needs = self.rhs.get_need_probabilities(next, value)

    needs = {}
    for r in self.requests:
      if r is next:
        needs[r] = 1.0

      else:
        if lhs_needs.has_key(r):
          needs[r] = lhs_needs[r]
        else:
          needs[r] = 0.0

        if rhs_needs.has_key(r):
          needs[r] += rhs_needs[r] - needs[r] * rhs_needs[r]
          # Presume "lhs needs r" and "rhs needs r" independent

    return needs
```

C.7. `=SUM(...)`, output distribution

The obvious difference between `=SUM(...)` and the addition operator is that `=SUM(...)` accepts an arbitrary number of arguments. Thus, one way to calculate the PMF would be to repeatedly apply the method used by the addition operator, as follows:

```
class SumFunction:
  @memoize
  def calculate_output_distribution(self, next, value):
    # The interface for this method is given on page 129.

    odist = self.operands[0].get_output_distribution(next, value)
```

```
    odist_dict = dict(odist)
    # this will accumulate each additional operand

    for op in self.operands[1:]:            # "outer loop"
      op_odist = op.get_output_distribution(next, value)

      for v_lhs,p_lhs in odist_dict.items():  # "middle loop"

        for v_rhs,p_rhs in rhs_odist:         # "inner loop"

          v_result = v_lhs + v_rhs

          p_result = p_lhs * p_rhs
          # Presume lhs and rhs to be indepdendent

          # Initialize to 0.0, if necessary
          if not odist_dict.has_key(v_result):
            odist_dict[v_result] = 0.0

          odist_dict[v_result] += p_result
          # Each way of reaching this result is distinct, so
          # these cases are indeed mutually exclusive.

    odist = sorted( odist_dict.items() )
    return Distribution( odist )
```

There are two main factors that make this expensive. First, `add_distribution(…)`, which is already $O(|lhs.\text{odist}|\cdot|rhs.\text{odist}|)$, must be performed `len(self.operands)-1` times. At each step, $|lhs.\text{odist}|$ is the number of unique sums that can be made with the possible values of the operands seen so far. Below, I elaborate on the running time of that. Second, all of this will be executed once for every possible value of every descendent request. Later in this section, I will describe an optimization that I use to mitigate that (see section C.7.3 on page 141).

The running time can vary widely depends heavily on the operands. As I will show below, it can be $O(mn)$ in good cases, but $O(m^n)$ in a theoretical worst case.

C.7.1. Running time, worst case

The worst case for a `=SUM(…)` having n operands each up to m possible values each is $O(m^n)$. This running time is extremely unlikely to arise in practice but serves as an upper bound. Below, I show how that running time is derived.

The body of the "middle loop" is executed once for every distinct sum value for the operands processed so far. When all of the operands are identically distributed, there may be many ways to reach a given sum value, so the number of distinct values is limited. It is akin to the problem of counting the number of distinct sums that could result from rolling n 6-sided dice. The worst case uses an unlikely arrangement of operands from which each sum value can be generated only one way. The proof below is based on just two sets of integers, representing the respective sets of possible values from two operands.

Claim: The set of distinct sums from two sets of integers A and B, each of arbitrary length, may have as many as |A|·|B| elements, but no more. More formally,

$$\text{Let SumSet}(S_1 + \ldots + S_n) = \{x_1 + \ldots + x_n \mid x_i \in S_i , \quad 1 \le i \le n\}$$

$$A \subset \mathbb{Z}$$

$$B \subset \mathbb{Z}$$

$$\exists\ A,B : |\text{SumSet}(A+B)| = |A|\cdot|B| \qquad \text{"… as many as …"}$$

$$\forall\ A,B : |\text{SumSet}(A+B)| \le |A|\cdot|B| \qquad \text{"… but no more."}$$

Proof: We will prove that for some A and B of arbitrary length, |SumSet(A, B)| = |A| · |B|. Since each element in SumSet(A, B) is the sum of an element from A and an element from B, there cannot be any more than |A|·|B| combinations. To prove that it is possible to have such a worst case with |A|·|B| *distinct* sums, consider the following construction.

Let A and B_i be non-empty sets of positive integers ($A \subset \mathbb{Z}^+$, $B_i \subset \mathbb{Z}^+$)

Let $b_{i+1} = \max(\text{SumSet}(A + B_i)) + 1$

Let $B_{i+1} = B_i + \{b_{i+1}\}$

$b_{i+1} \notin B_i$ by definition of b_i

$\{b_{i+1}\} \cap |B_i| = \emptyset$

$|B_{i+1}| = |B_i| + 1$ by definition of B_{i+1}

$\forall x, x \in \text{SumSet}(A + B_i) \quad x < b_i$ by definitions of b_i, A, B_i

$\forall x, x \in \text{SumSet}(A + \{b_{i+1}\}) \quad x > b_i$ by definitions of b_i, A

$\text{SumSet}(A + B_i) \cap \text{SumSet}(A + \{b_{i+1}\}) = \emptyset$

$|\text{SumSet}(A + B_i)| = |\text{SumSet}(A + B_i)| + |\text{SumSet}(A + \{b_{i+1}\})|$

$\text{SumSet}(A + B_{i+1}) = \text{SumSet}(A + B_{i+1}) \cup \text{SumSet}(A + \{b_{i+1}\})$

$|\text{SumSet}(A + B_{i+1})| = |\text{SumSet}(A + B_i)| + |A|$

$|\text{SumSet}(A + B)| = |A|$ whenever $|B|=1$ by definitions of b_i, A, B_i

Let $B_0 = \{1\}$

$|B_0| = 1$ by definition of B_0

$|\text{SumSet}(A + B_0)| = |A|$

$$
\begin{aligned}
|\text{SumSet}(A + B_1)| &= |\text{SumSet}(A + B_0)| + |A| \\
&= |A| + |A| \\
&= 2 \cdot |A| \\
|\text{SumSet}(A + B_2)| &= |\text{SumSet}(A + B_1)| + |A| \\
&= 2 \cdot |A| + |A| \\
&= 3 \cdot |A| \\
&\dots \\
|\text{SumSet}(A + B_k)| &= |\text{SumSet}(A + B_{k-1})| + |A| \\
&= (k + 1) \cdot |A|
\end{aligned}
$$

Since $|B_0| = 1$ and $|B_{i+1}| = |B_i| + 1$, then by induction, $|B_k| = k + 1$. Therefore,

$$
\begin{aligned}
|\text{SumSet}(A + B_k)| &= (k + 1) \cdot |A| \\
&= |B_k| \cdot |A| \\
&= |A| \cdot |B_k| \qquad \square
\end{aligned}
$$

Based on the above proof, we know that after folding in the second operand to the `=SUM(…)`, `len(odist)` could be as much as `len(operands[0]) * len(operands[1].odist)`. More generally, after processing the i^{th} operand, `len(odist)` could be as much as $\prod_{0 \le j \le i}$ `len(operands[j].odist)`. Thus, in the worst case, a `=SUM(…)` with n operands each having up to m possible values may have as many as mn items in its output distribution.

The body of the *outer loop* (page 136) is executed n-1 times. In the i^{th} pass, the body of the middle loop is executed m^i times. For each pass through the body of the middle loop, the body of the inner loop is executed m times. In total, the body of the inner loop is executed $(m + m^2 + \ldots + m^{n-1})(m) = (m^2)(n-1)(n-2)$ times. Thus, the worst case running time for calculating the output distribution for `=SUM(…)` is $O(m^n)$.

C.7.2. Running time, good case

The worst case bound described above is based on a contrived construction. For decision models, the terms of the sum typically have distributions that are either identical or at least very similar.

For a "good" case[13], consider a `=SUM(…)` with n identical operands, each being `=ASK("1 to m")`. Such a model would correspond to a weighted sums model where every input is a 5-star rating (i.e., m=5) with no weights. This can be represented as follows:

Suppose A = {1, 2, …, m}.

Let SumSets(nA) = SumSets($\underbrace{S + S + \cdots + S}_{n}$)

SumSets(A) = {1, …, m}

SumSets(2A) = {2, …, 2m}

SumSets(3A) = {3, …, 3m}

SumSets(nA) = {n, …, nm}

[13] I do not know if this is the best case or not.

$$|\text{SumSets}(n\text{A})| = nm - n + 1$$

Since this corresponds to the number of times the body of the "middle loop" is executed, the running time for calculating the output distribution will be $O(nm^2)$.

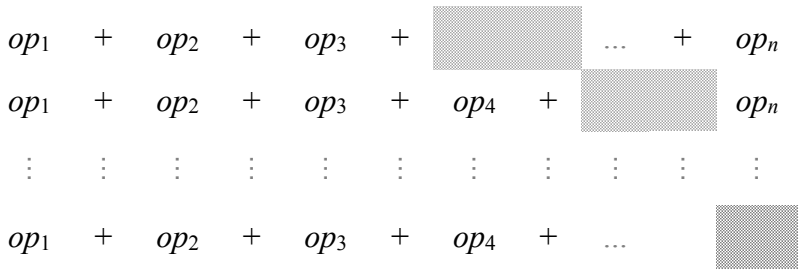
As in the worst case, the body of the *outer loop* (page 136) is executed n-1 times, but for the good case, in the i^{th} pass, the body of the middle loop is executed mi times. For each pass through the body of the middle loop, the body of the inner loop is executed m times. In total, the body of the inner loop is executed $(m + 2m + \ldots + (n\text{-}1)m)(m) = (m^2)(n\text{-}1)(n\text{-}2)$ times. Thus, the running time for calculating the output distribution for `=SUM(…)` in the good case is $O(m^2n^2)$.

The complexity depends heavily on the specifics of the model. At any rate, profiling has shown that this is a significant part of the time AskSheet spends prioritizing many models. Therefore, I have added some optimization.

C.7.3. Optimization with partial sum precalculation

As with all operations, the output distribution must be calculated repeatedly for different conditions: once for every request (`next`), and for every value (`value`) of that request. Typically, only one operand changes each time. Therefore, AskSheet precomputes and caches the output distribution for the sum of all but one operands—for each operand. For a model with n operands (op_1, op_2, …, op_n), it precomputes n partial sums.

		op_2	+	op_3	+	op_4	+	…	+	op_n
op_1	+			op_3	+	op_4	+	…	+	op_n
op_1	+	op_2	+			op_4	+	…	+	op_n



All of these partial sums can be computed in running time proportional to what would be required to calculate the output distribution for a single scenario (i.e., one combination of `next` and `value`). To get each of the sums $op1 + \ldots + op_i$ ($0 \le i < n$), it effectively follows the same algorithm (page 136) but saves a copy of `odist_dict` at the end of each pass through the *outer loop*. For $op1 + \ldots + op_i$ ($0 \le i < n$), it effectively does the same, but visiting the operands in the reverse order. With those pre-calculated, then for any `=SUM(…)` of which no more than one operand depends on any given request, each additional scenario can be calculated by simply adding in whichever operand depends on `next`. This entails a single pass through the body of the outer loop instead of n passes, which improves the overall running time for some important model types (See the discussion of performance and overall running time in section 6.2.2 on page 107.)

C.8. `=SUM(…)`, need probabilities

The calculation of need probabilities for `=SUM(…)` is relatively simple compared to its output distribution. It needs a request if and only if any of its operands needs it. The running time is $O(ab+ac)$ where a=|operands|, b=«number of requests each operand depends on», and c=«average time to calculate output distribution of an operand».

```
class SumFunction:
  def get_need_probabilities(self, next, value):
    # The interface for this method is given on page 130.
```

```
    needs = {}

    for op in self.operands:
      op_needs = op.get_need_probabilities(next, value)

      for r,p_op_needs_r in op_needs:
        if not needs.has_key(r):
          needs[r] = 0.0

        needs[r] += p_op_needs_r - p_op_needs_r * needs[r]

    needs[next] = 1.0
    return needs
```

C.9. =MAX(…), output distribution

The key idea is that Pr(result = v) = Pr(all operands ≤ v) – Pr(all operands < v) for any v among the set of possible output values. This is because for the maximum to be v, no operand may be greater than v, but at least one must be equal to v. Since =MAX(…) is the same as the first order statistic, this formula is consistent with the formula for the probability mass function for order statistics on discrete random variables (Casella & Berger, 2002, pp. 227–228). The running time is $O(ab+ac)$ where a=|operands|, b=«average time to calculate the output distribution of an operand», c=«average number of distinct values found in any operand».

```
class MaxFunction:
  @memoize
  def get_output_distribution(self, next=None, value=None):
    # The interface for this method is given on page 129.

    op_odists = "list of output dists of every operand"
    min_val  = max(d.min_val for d in op_odists)
    out_vals = "sorted list of all operands' vals ≥min_val"
    op_odist_cursors = [0   for _ in op_odists]
    p_each_op_lt_v   = [0.0 for _ in op_odists]
    p_each_op_le_v   = [0.0 for _ in op_odists]
```

```
  for v in out_vals:
    p_all_lt_v = 1.0
    p_all_le_v = 1.0
    for op_idx in range(len(op_odists)):
      op_odist = op_odists[op_idx]
      op_odist_cursor = op_odist_cursors[op_idx]
      p_op_lt_v = p_each_op_lt_v[op_idx]
      p_op_le_v = p_each_op_le_v[op_idx]

      while op_odist_cursor + 1 < len(op_odist) and \
           op_odist[op_odist_cursor + 1] <= v:
        op_odist_cursor += 1
        _v,_p = op_odist[op_odist_cursor]
        p_op_lt_v += _p
        if _v == v:
          p_op_le_v += _p
        # Each value is a distinct, mutually exclusive case

      op_odist_cursors[op_idx] = op_odist_cursor

      p_each_op_lt_v[op_idx] = p_op_lt_v
      p_each_op_le_v[op_idx] = p_op_le_v

    p_all_lt_v *= p_op_lt_v
    p_all_le_v *= p_op_le_v
    # Presume that all operands are independent

    # ***** KEY FORMULA ******
    p_eq_v = p_op_le_v - p_op_lt_v

    odist.append( (v, p_eq_v) )

  return Distribution(odist)
```

C.10. =MAX(…), need probabilities

The key idea is that we need a request if we need any operand that needs it. We need an operand if there is any possibility that it will be greater than the output value. Stated differently, for any *v* in the set of possible output values, we need every operand of

which the maximum possible value is greater than *v*. In addition, we need one operand of which the value is *v*. The running time is $O(ab + ac + ad)$ where *a*=«number of operands», *b*=«average number of distinct output values for any operand», *c*=«average time to calculate the output distribution of an operand», *d*=«average number of requests per operand».

A known flaw in this algorithm is that for each value *v*, it chooses one operand (the *value owner*) arbitrarily as the one that is needed whenever the value of the `=MAX(...)` is *v*. That might result in a small, unjustified bias for some requests over others.

```
class MaxFunction:
  @memoize
  def get_need_probabilities(self, next=None, value=None):
    # The interface for this method is given on page 130.

    # Calculate "value owners" for each possible value
    val_owners = {}
    for op in operands
        op_odist = op.get_output_distribution(
               r_suppose_next=r_suppose_next, r_val=r_val)
        op_max = op_odist.max_value
        for v,p in op_odist:
          if v not in val_owners or val_owners[v][1] < op_max:
            val_owners[v] = op

    # Calculate Pr(op needs r) for every operand op and request r
    for op in self.operands:
      if op.requests == (next,):
        p_need_op = 1.0

      else:
        op_odist = op.get_output_distribution(next, value)
        op_max = op_odist.max_value
        if not val_owners.has_key(op_max):
          # Never needed ... The fact that op_max has no value
          # owner implies that this operand always less than the
          # minimum possible value of at least one other operand
```

```
        p_need_op = 0.0

      elif val_owners[op_max] is op:
        # Need this operand only when MAX is less than or equal
        # to the least possible value for this operand
        p_need_op = odist.probability_le(op_min)

      else:
        # Same as the previous case, but < instead of <=
        p_need_op = odist.probability_lt(op_min)

    # Calculate Pr(MAX needs r) for each request r
    for op in self.operands:
      op_needs = op.get_need_probabilities(next, value)
      for r,p_op_needs_r in op_needs.items():
        needs[r] += p_need_op * p_op_needs_r
        # Presume that operands are independent and only one may
        # depend on a given request (i.e., "op1 needs r" is
        # mutually exclusive with "op2 needs r").  This will be
        # true for tree-structured models but not for some others
    return needs
```

C.11. `=IF(...)`, output distribution

The syntax is of an `=IF(...)` function is as follows:

```
=IF(condition,  value_if_true, value_if_false)
```

Its output distribution is the sum of `value_if_true` and `value_if_false` weighted by the respective probabilities that `condition` is true or `false`. The running time is $O(a + b + c + d + e)$ where a, b, and c are the times to calculate the output of the three operands (respectively), d=«number of distinct possible values of `value_if_true`», and e=«number of distinct possible values of `value_if_false`».

```
class IfFunction:
  @memoize
  def get_output_distribution(self, next, value):
    # The interface for this method is given on page 129.
```

```
    cond_odist = self.cond.get_output_distribution(next, value)

    # Output distribution of the result of the =IF(…)
    odist_dict = {}

    # Add in case of when condition=True
    val_if_t_odist = self.value_if_true.get_output_distribution(
                                                 next, value)
    for v,p in val_if_t_odist:
      odist_dict[v] = p * cond_odist[True]
      # Presume condition and value_if_true are independent

    # Add in case of when condition=False
    val_if_f_odist = self.value_if_false.get_output_distribution(
                                                 next, value)
    for v,p in val_if_f_odist:
      if not odist_dict.has_key(v):
        odist_dict[v] = 0.0
      odist_dict[v] += p * cond_odist[False]
      # Mutually exclusive because condition is either T or F
      # Presume condition and value_if_true are independent

    # Return the distribution in the usual format
    odist_dict = sorted(odist_dict.iteritems())
    return Distribution(odist_dict)
```

C.12. `=IF(…)`, need probabilities

The `=IF(…)` function needs a request that `value_if_true` depends on if and only if `condition=True`, and likewise for requests that `value_if_false` depends on. Running time is $O(a + b + c + d)$ where a=«time to calculate the output distribution of the `condition`», b=«average time to calculate the need probabilities any of the operands», c=«number of distinct possible values of `value_if_true`», and d=«number of distinct possible values of `value_if_false`».

```
class IfFunction:
```

```
  @memoize
  def get_need_probabilities(self, next, value):
    # The interface for this method is given on page 130.
    needs = {}
    cond_odist = self.cond.get_output_distribution(next, value)
    p_true  = cond_odist[True]
    p_false = cond_odist[False]

    cond_needs = self.cond.get_need_probabilities(next, value)

    for (op,p_need_op) in ((self.value_if_true,  p_true),
                           (self.value_if_false, p_false),
                           (self,cond,           1.0)):
      op_needs = op.get_need_probabilities(next, value)
      for r,p_op_needs_r in op_needs.items():
        if not needs.has_key(r):
          needs[r] = 0.0

        p_need_r_for_op = p_need_op * p_op_needs_r
        needs[r] += p_need_r_for_op - p_need_r_for_op * needs[r]
    return needs
```

C.13. =ASK(…), output distribution

As a premise of AskSheet's design, every =ASK(…) formula has a discrete uniform distribution. In principle, this encompasses every possible value that could be entered into the input form, within the range given by the parameters. An earlier version of the AskSheet implementation used every possible value. For example, =ASK("1 to 5") would have 5 possible values, but =ASK("$1.00 to $5.00") would have 501 possible values. For performance reasons, the implementation now uses sampling with a configurable sample size (currently set to 5). Samples are taken at uniform intervals and then rounded as appropriate (e.g., for integer inputs). The running time is O(|*samples*|), or *O*(1) if you consider the number of samples as a constant.

```
class AskFunction:
  @memoize
```

```
  def get_output_distribution(self, next, value):
    # The interface for this method is given on page 129.

    odist = []
    if next is self:
      odist.append( (value, 1.0) )
    else:
      p = 1.0 / len(self.samples)
      for v in self.samples:
        odist.append( (v, p) )
    return Distribution(odist)
```

C.14. =ASK(...), need probabilities

An =ASK(...) formula needs itself if and only if it has not yet been fulfilled. As usual, any request passed as a condition (next) is also needed. The running time is $O(1)$.

```
class AskFunction:
  @memoize
  def get_need_probabilities(self, next, value):
    # The interface for this method is given on page 130.
    needs = {}
    if not self.is_stable:
      needs[self] = 1.0
    needs[next] = 1.0
    return needs
```

C.15. =AND(...), output distribution

The output distribution is simply the conjunction of the output distributions of every operand. As usual, we presume all operands are independent, even though that may not always be the case for non-tree-structured models. The running time is $O(ab)$ where a=|*operands*| and b=«average time to calculate the output distribution of any of the operands».

```
class AndFunction:
  @memoize
  def get_output_distribution(self, next, value):
    # The interface for this method is given on page 129.
```

```
    p_all_t = 1.0
    for op in self.operands:
      op_odist = op.get_output_distribution(next, value)
      p_all_t *= op_odist.probability_true
      # Presume all operands are independent

    p_any_f = 1.0 - p_all_t
    return Distribution(((False, p_any_f), (True, p_all_t)))
```

C.16. =AND(...), need probabilities

Assume the operands were chosen at random. The probability that a given operand is needed is the probability that evaluation does not stop due to short-circuit evaluation before it is chosen. The running time is $O(ab + ac)$ where a=|operands|, b=«average time to calculate the output distribution of any of the operands», and c=«average time to calculate the output distribution of any of the operands».

```
class AndFunction:
  @memoize
  def get_need_probabilities(self, next, value):
    # The interface for this method is given on page 130.

    # Check if any operand is guaranteed to always be False
    # e.g., due to inputs already received
    is_short_circuited = False
    for op in self.operands:
      odist = op.get_output_distribution(next, value)
      if odist.probability_false == 1.0:
        is_short_circuited = True
        break

    if not is_short_circuited:
      num_ops = len(self.operands)

      p_each_op_allows = []
      for op in self.operands:
        op_odist = op.get_output_distribution(next, value)
        p_op_is_true = op_odist.probability_true
```

```
      p_each_op_allows.append( 0.5 * p_op_is_true + 0.5 )


    # To keep this linear, accumulate forwards then backwards
    p_cumulative = 1.0
    p_need_each_op = []
    for i in xrange(num_ops):          # 0 ... num_ops-1
      p_need_each_op.append( p_cumulative )
      p_cumulative *= p_each_op_allows[i]

    p_cumulative = 1.0
    for i in range(num_ops-1, -1, -1): # num_ops-1 ... 0
      p_need_each_op[i] *= p_cumulative
      p_cumulative *= p_each_op_allows[i]

    needs = {}
    for i in range(num_ops):          # 0 ... num_ops-1
      op = self.operands[i]
      op_needs = op.get_need_probabilities(next, value)
      for r,p_op_needs_r in op_needs.items():
        if not needs.has_key(r):
          needs[r] = 0.0
        p_need_op = p_need_each_op[i]
        p_need_r_for_op = p_op_needs_r * p_need_op
        needs[r] += p_need_r_for_op - p_need_r_for_op*needs[r]

  return needs
```

C.17. =OR(…), output distribution

The output distribution is the probability that any of the operands is true. AskSheet calculates this as the inverse of the probability that every operand is false, under the presumption that all operands are independent of one another. The running time is $O(ab)$ where a=|operands| and b=«average time to calculate the output distribution of any of the operands».

```
class OrFunction:
  @memoize
  def get_output_distribution(self, next, value):
```

```
    # The interface for this method is given on page 129.

    p_all_f = 1.0
    for op in self.operands:
      op_odist = op.get_output_distribution(next, value)
      p_all_f *= op_odist.probability_false
      # Presume all operands are independent

    p_any_t = 1.0 - p_all_f
    return Distribution(((False, p_all_f), (True, p_any_t)))
```

C.18. =OR(...), need probabilities

The strategy for =OR(...) is the essentially the same as for =AND(...). A request is needed if—while selecting operands at random—an operand that depends on the request is chosen before a true value is encountered. The running time is $O(ab + ac)$ where a=|operands|, b=«average time to calculate the output distribution of any of the operands», and c=«average time to calculate the output distribution of any of the operands».

```
class OrFunction:
  @memoize
  def get_need_probabilities(self, next, value):
    # The interface for this method is given on page 130.

    # Check if any operand is guaranteed to always be True
    # e.g., due to inputs already received
    is_short_circuited = False
    for op in self.operands:
      odist = op.get_output_distribution(next, value)
      if odist.probability_true == 1.0:
        is_short_circuited = True
        break

    if not is_short_circuited:
      num_ops = len(self.operands)

      p_each_op_allows = []
      for op in self.operands:
        op_odist = op.get_output_distribution(next, value)
```

```
      p_op_is_false = op_odist.probability_false
      p_each_op_allows.append( 0.5 * p_op_is_false + 0.5 )

    # To keep this linear, accumulate forwards then backwards
    p_cumulative = 1.0
    p_need_each_op = []
    for i in xrange(num_ops):          # 0 ... num_ops-1
      p_need_each_op.append( p_cumulative )
      p_cumulative *= p_each_op_allows[i]

    p_cumulative = 1.0
    for i in range(num_ops-1, -1, -1): # num_ops-1 ... 0
      p_need_each_op[i] *= p_cumulative
      p_cumulative *= p_each_op_allows[i]

    needs = {}
    for i in range(num_ops):           # 0 ... num_ops-1
      op = self.operands[i]
      op_needs = op.get_need_probabilities(next, value)
      for r,p_op_needs_r in op_needs.items():
        if not needs.has_key(r):
          needs[r] = 0.0
        p_need_op = p_need_each_op[i]
        p_need_r_for_op = p_op_needs_r * p_need_op
        needs[r] += p_need_r_for_op - p_need_r_for_op*needs[r]

  return needs
```

C.19. =NOT(...), output distribution

The output distribution of =NOT(...) is just the distribution of its sole operand with the values true and false swapped. The running time is $O(a + b)$ where a=«time to calculate the output distribution of the operand» and b=«number of distinct possible values that the operand could take».

```
class NotFunction:
  @memoize
  def get_output_distribution(self, next, value):
    # The interface for this method is given on page 129.
```

```
    op_odist = self.arg.get_output_distribution(next, value)
    odist = []
    for v,p in op_odist:
      v = not v
      odist.append( (v, p) )
    odist.sort()
    return Distribution( odist )
```

C.20. =NOT(...), need probabilities

The =NOT(...) function needs all requests that its sole operand depends on. The running time is *O*(«time to calculate the need probabilities of the operand»).

```
class NotFunction:
  @memoize
  def get_need_probabilities(self, next, value):
    # The interface for this method is given on page 130.
    return self.arg.get_need_probabilities(next, value)
```

C.21. =INDEX(...), output distribution

The =INDEX(...) function is used to look up values in a data table, usually in conjunction with the =MATCH(...) function. The syntax is as follows:

```
=INDEX(array, row_number, column_number)
```

For example, =INDEX(A1:C10, 4, 3) would return the value of cell C4. The output distribution is a marginal probability distribution of the value of the selected cell, over all possible cells that could be selected.

The running time is $O(a + b + abc + abd)$ where a=«number of distinct possible values of row_number», b=«number of distinct possible values of row_number», c=«average time to calculate the output distribution of any of the cells in array that row_number or column_number might point to», and d=«average number of distinct

possible values in the output distribution of any of the cells in `array` that `row_number` or `column_number` might point to».

```
class IndexFunction:
  @memoize
  def get_output_distribution(self, next, value):
    # The interface for this method is given on page 129.

    row_odist = self.row_num.get_output_distribution(next, value)
    col_odist = self.col_num.get_output_distribution(next, value)

    odist_dict = {}
    for row_num,row_num_p in row_odist:
      for col_num,col_num_p in col_odist:
        cell = self.array.get(row_num, col_num)
        cell_odist = cell.get_output_distribution(next, value)

        for v,p in cell_odist:

          if not odist_dict.has_key(v):
            odist_dict[v] = 0.0

          odist_dict[v] += p * col_num_p * row_num_p
          # Mutually exclusive because only one cell can be
          # selected, and it can have only one value

    dist_items = sorted(odist_dict.items())
    return Distribution(odist_dict)
```

C.22. `=INDEX(...)`, need probabilities

The `=INDEX(...)` function always needs its `row_number` and `column_number` operands and any requests that they may depend on. For other cells, the need probability is the marginal probability that the cell is selected by the values of `row_number` and `column_number`.

The running time is $O(a + b + abc + abd)$ where a=«number of distinct possible values of `row_number`», b=«number of distinct possible values of `row_number`»,

c=«average time to calculate the need probabilities of any of the cells in `array` that `row_number` or `column_number` might point to», and *d*=«average number of descendent requests of any of the cells in `array` that `row_number` or `column_number` might point to».

```
class IndexFunction:
  @memoize
  def get_need_probabilities(self, next, value):
    # The interface for this method is given on page 130.

    row_odist = self.row_num.get_output_distribution(next, value)
    col_odist = self.col_num.get_output_distribution(next, value)

    # Add in the needs due to cells
    needs = {}
    for row_num_v,row_num_p in row_odist:
      for col_num_v,col_num_p in col_odist:
        cell = self.array.get(row_num_v, col_num_v)
        p_need_cell = row_num_p * col_num_p
        cell_needs = cell.get_need_probabilities(next, value)
        for r,p_cell_needs_r in cell_needs.items():
          if not needs.has_key(r):
            needs[r] = 0.0
          p_need_r_for_cell = p_cell_needs_r * p_need_cell
          needs[r] += p_need_r_for_cell - \
                                p_need_r_for_cell * needs[r]

    # Add in the needs due to the row_num parameter
    row_needs = self.row_num.get_need_probabilities(next, value)
    for r,p in row_needs.items():
      if not needs.has_key(r):
        needs[r] = 0.0
      needs[r] += p - needs[r] * p

    # Add in the needs due to the col_num parameter
    col_needs = self.col_num.get_need_probabilities(next, value)
    for r,p in col_needs.items():
      if not needs.has_key(r):
        needs[r] = 0.0
```

```
        needs[r] += p - needs[r] * p

    return needs
```

C.23. =MATCH(…), output distribution

The =MATCH(…) function is used to search a row or column for a particular value. The syntax is as follows:

```
=MATCH(key, array, match_type)
```

`key` ≔ value to search for

`array` ≔ row or column to search in

`match_type` ≔ always 0; obscure feature of Microsoft Excel[14]

It returns the *first* cell number (i.e., column number or row number) within the array where the key was found. The cell number is 1-based. The =MATCH(…) function is often used in conjunction with =INDEX(…) to perform table lookups.

Since it returns the *first* cell number where the key matched, the output distribution calculation cannot ignore order entirely. The calculation can be summarized as follows:

$$\Pr(\text{MATCH} = n) = \Pr(\text{“}n^{\text{th}}\text{ cell} = key\text{”} \wedge \text{“}key\text{ did not match in cells 1 to } (n\text{-}1)\text{”})$$

---

[14] In Microsoft Excel and other spreadsheet applications that use the same syntax, the `match_type` parameter to =MATCH(…) may be 1 (default), 0, or -1.
- =MATCH(…, 0) returns the first cell number with a value equal to `key`.
- =MATCH(…, 1) returns the cell number with the largest value less than or equal to `key`.
- =MATCH(…, -1) returns the cell number with the smallest value greater than or equal to `key`.

The running time is $O(a + bc + bd)$, where a=«time to calculate output distribution of `key`», b=«number of cells in `array`», c=«average time to calculate output distribution of any of the cells in `array`», and d=«average number of distinct possible values in the output distribution of any of the cells in `array`».

```
class MatchFunction:
  @memoize
  def get_output_distribution(self, next, value):
    # The interface for this method is given on page 129.

    odist = []
    key_odist = self.key.get_output_distribution(next, value)

    p_not_matched_yet = 1.0
    p_matched_any = 0.0

    cell_num = 0  # will be 1-based
    for cell in self.array.cells:

      cell_odist = cell.get_output_distribution(next, value)

      cell_num += 1   # 1-based

      # Pr(MATCH = cell_num)
      p_key_eq_v = 0.0

      for v,p_cell_eq_v in cell_odist:

        # Pr(cell = key)
        p_key_eq_v = key_odist.get(v, default=0.0)

        p_cell_eq_key_eq_v = p_key_eq_v * p_cell_eq_v
        # Presume cell and key are independent

        p_cell_num += p_cell_eq_key_eq_v * (1.0 - p_matched_any)

        p_matched_any += p_cell_eq_key_eq_v
        # Mutually exclusive because p_matched_any refers only
        # to potential matches prior to examining this cell and
```

```
      # possible value

    odist.append( (cell_num, p_cell_num) )

  return Distribution(odist)
```

C.24. =MATCH(…), need probabilities

The =MATCH(…) function always needs the `key` parameter and any requests that it depends on. The output distribution can be used to determine the probability that a given cell—and its descendent requests—will be needed. Recall that =MATCH(…) returns the cell number of the first cell in the array that matches the `key` parameter.

The running time is $O(a + bc + bd)$, where a=«time to calculate output distribution of this function», b=«number of cells in `array`», c=«average time to calculate the need probabilities of any of the cells in `array`», and d=«average number of descendent requests of any of the cells in `array`».

```
class MatchFunction:
  @memoize
  def get_need_probabilities(self, next, value):
    # The interface for this method is given on page 130.

    odist = self.get_output_distribution(next, value)
    odist_dict = dict(odist) # convert to dictionary {v => p}

    # Initialize needs
    needs = {}
    for r in self.requests:
      needs[r] = 0.0

    cell_num = 0  # 1-based
    for cell in self.array.cells:
      cell_num += 1 # 1-based

      if odist_dict.has_key(cell_num):
        p_need_cell = odist_dict[cell_num]
```

```
      cell_needs = cell.get_need_probabilities(next, value)

      for r,p_cell_needs_r in cell_needs.items():
        p_need_r_for_cell = p_cell_needs_r * p_need_cell
        needs[r] += p_need_r_for_cell - \
                              p_need_r_for_cell * needs[r]
        # Presume "need r for this cell" and "need r for
        # a prior cell" to be independent (but not necessarily
        # mutually exclusive).

    key_needs = self.key.get_need_probabilities(next, value)
    for r,p_key_needs_r in key_needs.items():
      needs[r] += p_key_needs_r - p_key_needs_r * needs[r]
      # Presume "need r for key" and "need r for ≥1 cell(s)" to
      # be independent (but not necessarily mutually exclusive)

    needs[next] = 1.0
    return needs
```

Appendix D. Raw results from survey of decision problems

D.1. Question text

Version 1

> List 3-5 types of decisions that you find especially time consuming (i.e., time spent sifting through information, gathering input from others, etc.). These could be personal or work-related decisions.

Version 2

> List 3-5 types of decisions that you find especially time-consuming (i.e., time spent sifting through information, gathering input from others). These could be

Version 2

List 3-5 types of decisions that you find especially time-consuming (i.e., time spent sifting through information, gathering input from others). These could be personal or work-related, past or present.

Examples:

- Deciding which ____ to ____ requires looking up a lot information from various web sites.
- Deciding which ____ to ____ requires a lot of asking around.
- Deciding when/what/where/whether to ____ takes time to dig through web sites for details.
- Deciding when/what/where/whether to ____ requires consulting a lot of people.
- ...

D.2. Raw (verbatim)

1. Deciding where or what to eat can take an hour for me
2. Deciding whether or not some project I do is acceptable takes a lot of time
3. deciding which place to move to requires a lot of asking around
4. Deciding what papers to read takes a lot of time finding them/reading abstracts
5. Deciding which expensive items to buy (electronics, furniture) takes a lot of time reading and comparing items
6. Deciding what software to use requires a lot of reading and asking friends
7. hotel to stay in

8. airline to use
9. vendor to order from
10. which system to purchase
11. what the requirements are for a project
12. Writing lessons
13. doing taxes
14. and buying a house
15. Buying new technology
16. Job search
17. Going on vacation
18. Buying a new home
19. financial decisions (purchases, account management, filing out tax forms, etc.)
20. decisions in a committee
21. choosing a place to eat lunch
22. Buying a home.
23. Buying a Car.
24. Deciding on a new Cell phone providers.
25. Deciding on a new name for my business.
26. Choosing the most suitable insurance company for home, car, and life insurance
27. Choosing appropriate for our way of breed of a dog to adopt
28. Making decision whether we can afford a new car
29. Ordering tasks in my day
30. Choosing a research topic

31. Gathering papers to read
32. scheduling a meeting for others
33. deciding what to say/not say in emails
34. planning an outing (and collecting all of the information needed
35. even if it's just me going)
36. what to make for dinner
37. which vegetables to buy at the super market
38. the order of my routine when I wake up or arrive home (optimizing repetitive tasks takes time
39. so I don't have to take the time once I'm home).
40. Medical disability paperwork
41. resume and cover-letter preparation
42. and lately online shopping for electronics.
43. Gathering information
44. planning implementation of new concepts
45. evaluating the effectiveness of outcomes.
46. gathering information of a domain that I am not familiar with,
47. gathered information leads me to even bigger area to understand,
48. gathered information does not follow the structure I had in my mind
49. buying a new car
50. living arrangements as I age
51. ways to get together with family
52. deciding what to put in answer to this question

53. what book to read next
54. investments, e.g. stocks, other paper assets, real estate, etc. (seriously time-consuming)
55. travel arrangements, the destination, and then the accommodations and transport.

D.3. Summarized (edited)

A. Food
    1. what to make for dinner
    2. which vegetables to buy at the super market
    3. where or what to eat can take an hour for me
    4. choosing a place to eat lunch
B. Home
    1. which place to move to requires a lot of asking around
    2. which house to buy
    3. buying a new home
    4. which house to buy
C. Research reading
    1. what papers to read takes a lot of time finding them/reading abstracts
    2. what book to read next
    3. gathering papers to read
D. Technology
    1. which expensive items to buy (electronics, furniture) takes a lot of time reading and comparing items
    2. what software to use requires a lot of reading and asking friends

3. which new technology to buy

4. which Cell phone provider to use

5. online shopping for electronics.

E. Travel

1. which hotel to stay in

2. which airline to use

3. going on vacation

4. ways to get together with family

5. planning an outing

6. travel arrangements, the destination, and then the accommodations and transport

F. Purchasing (general)

1. which vendor to order from

2. which system to purchase

G. Car

1. which car to buy

2. whether we can afford a new car

3. buying a new car

H. Finances

1. financial decisions (purchases, account management, filing out tax forms, etc.)

2. living arrangements as I age

3. investments, e.g. stocks, other paper assets, real estate, etc. (seriously time-

consuming)

4. which insurance company for home, car, and life insurance

I. Time management

1. whether or not some project I do is acceptable takes a lot of time

2. ordering tasks in my day

3. the order of my routine when I wake up or arrive home (optimizing repetitive tasks takes time so I don't have to take the time once I'm home).

J. Coordinating people

1. scheduling a meeting for others

2. decisions in a committee

K. Setting goals

1. what the requirements are for a project

2. choosing a research topic

L. Other

1. which new name for my business

2. which breed of a dog to adopt

3. deciding what to say/not say in emails

Appendix E. Modeling study models

E.1. “Options for Child Birth”

	A	B	C	D	E	F	G	H	I	J	K
1		Easy of movement with greater mobility	Relaxation during and between contractions	Safe and effective pain management	Reduction of blood pressure	A sense of control as mother occupies her s...	Facilitation of cervical dilation	Safety for the baby	Less stress for the baby	Easier in/out belly transition for the baby	score
2	*weights*	*4*	*4*	*5*	*6*	*4*	*7*	*10*	*9*	*8*	
3	Natural Vaginal Birth	=ASK("score: hard < easy", A3, B1, "A")	=ASK("score: hard < easy", A3, C1, "A")	=ASK("score: hard < easy", A3, D1, "A")	=ASK("score: hard < easy", A3, E1, "B")	=ASK("score: hard < easy", A3, F1, "A")	=ASK("score: hard < easy", A3, G1, "B")	=ASK("score: hard < easy", A3, H1, "C")	=ASK("score: hard < easy", A3, I1, "C")	=ASK("score: hard < easy", A3, J1, "C")	=SUM(B3*B2, C3*C2, D3*D2, E3*E2, F3*F2, G3*G2, H3*H2, I3*I2, J3*J2)
4	Hospital Vaginal Birth without Epidural	=ASK("score: hard < easy", A4, B1, "A")	=ASK("score: hard < easy", A4, C1, "A")	=ASK("score: hard < easy", A4, D1, "A")	=ASK("score: hard < easy", A4, E1, "B")	=ASK("score: hard < easy", A4, F1, "A")	=ASK("score: hard < easy", A4, G1, "B")	=ASK("score: hard < easy", A4, H1, "C")	=ASK("score: hard < easy", A4, I1, "C")	=ASK("score: hard < easy", A4, J1, "C")	=SUM(B4*B2, C4*C2, D4*D2, E4*E2, F4*F2, G4*G2, H4*H2, I4*I2, J4*J2)
5	Hospital Vaginal Birth without Epidural with Hydrotherapy	=ASK("score: hard < easy", A5, B1, "A")	=ASK("score: hard < easy", A5, C1, "A")	=ASK("score: hard < easy", A5, D1, "A")	=ASK("score: hard < easy", A5, E1, "B")	=ASK("score: hard < easy", A5, F1, "A")	=ASK("score: hard < easy", A5, G1, "B")	=ASK("score: hard < easy", A5, H1, "C")	=ASK("score: hard < easy", A5, I1, "C")	=ASK("score: hard < easy", A5, J1, "C")	=SUM(B5*B2, C5*C2, D5*D2, E5*E2, F5*F2, G5*G2, H5*H2, I5*I2, J5*J2)
6	Hospital Vaginal Birth without Epidural with Ball	=ASK("score: hard < easy", A6, B1, "A")	=ASK("score: hard < easy", A6, C1, "A")	=ASK("score: hard < easy", A6, D1, "A")	=ASK("score: hard < easy", A6, E1, "B")	=ASK("score: hard < easy", A6, F1, "A")	=ASK("score: hard < easy", A6, G1, "B")	=ASK("score: hard < easy", A6, H1, "C")	=ASK("score: hard < easy", A6, I1, "C")	=ASK("score: hard < easy", A6, J1, "C")	=SUM(B6*B2, C6*C2, D6*D2, E6*E2, F6*F2, G6*G2, H6*H2, I6*I2, J6*J2)
7	Hospital Vaginal Birth with Epidural	=ASK("score: hard < easy", A7, B1, "A")	=ASK("score: hard < easy", A7, C1, "A")	=ASK("score: hard < easy", A7, D1, "A")	=ASK("score: hard < easy", A7, E1, "B")	=ASK("score: hard < easy", A7, F1, "A")	=ASK("score: hard < easy", A7, G1, "B")	=ASK("score: hard < easy", A7, H1, "C")	=ASK("score: hard < easy", A7, I1, "C")	=ASK("score: hard < easy", A7, J1, "C")	=SUM(B7*B2, C7*C2, D7*D2, E7*E2, F7*F2, G7*G2, H7*H2, I7*I2, J7*J2)
8	Hospital Scheduled Induction without Epidural	=ASK("score: hard < easy", A8, B1, "A")	=ASK("score: hard < easy", A8, C1, "A")	=ASK("score: hard < easy", A8, D1, "A")	=ASK("score: hard < easy", A8, E1, "B")	=ASK("score: hard < easy", A8, F1, "A")	=ASK("score: hard < easy", A8, G1, "B")	=ASK("score: hard < easy", A8, H1, "C")	=ASK("score: hard < easy", A8, I1, "C")	=ASK("score: hard < easy", A8, J1, "C")	=SUM(B8*B2, C8*C2, D8*D2, E8*E2, F8*F2, G8*G2, H8*H2, I8*I2, J8*J2)
9	Hospital Scheduled Induction with Epidural	=ASK("score: hard < easy", A9, B1, "A")	=ASK("score: hard < easy", A9, C1, "A")	=ASK("score: hard < easy", A9, D1, "A")	=ASK("score: hard < easy", A9, E1, "B")	=ASK("score: hard < easy", A9, F1, "A")	=ASK("score: hard < easy", A9, G1, "B")	=ASK("score: hard < easy", A9, H1, "C")	=ASK("score: hard < easy", A9, I1, "C")	=ASK("score: hard < easy", A9, J1, "C")	=SUM(B9*B2, C9*C2, D9*D2, E9*E2, F9*F2, G9*G2, H9*H2, I9*I2, J9*J2)
10	Hospital Cesarean Birth	=ASK("score: hard < easy", A10, B1, "A")	=ASK("score: hard < easy", A10, C1, "A")	=ASK("score: hard < easy", A10, D1, "A")	=ASK("score: hard < easy", A10, E1, "B")	=ASK("score: hard < easy", A10, F1, "A")	=ASK("score: hard < easy", A10, G1, "B")	=ASK("score: hard < easy", A10, H1, "C")	=ASK("score: hard < easy", A10, I1, "C")	=ASK("score: hard < easy", A10, J1, "C")	=SUM(B10*B2, C10*C2, D10*D2, E10*E2, F10*F2, G10*G2, H10*H2, I10*I2, J10*J2)
11	max	=MAX(K3:K10)									
12	best	=IF(B11=0, "", INDEX(A3:A10, MATCH(B11, K3:K10, 0), 1))									

E.2. “Pediatrician”

	A	B	C	D	E	F	G
1		Distance from home	Availability without appointment	Years of experience	Patient satisfaction	Time spent with patient	score
2	*weights*	*6*	*7*	*9*	*10*	*8*	
3	Dr. Scott D. Wissman, MD	=ASK("score: 15miles < 0", A3, B1, "A")	=ASK("score: hard < easy", A3, C1, "A")	=ASK("score: 10 < 40", A3, D1, "A")	=ASK("score: little < much", A3, E1, "A")	=ASK("score: 10' < 40'", A3, F1, "A")	=SUM(B3*B2, C3*C2, D3*D2, E3*E2, F3*F2)
4	Dr. Leila T. Hall, MD	=ASK("score: 15miles < 0", A4, B1, "A")	=ASK("score: hard < easy", A4, C1, "A")	=ASK("score: 10 < 40", A4, D1, "A")	=ASK("score: little < much", A4, E1, "A")	=ASK("score: 10' < 40'", A4, F1, "A")	=SUM(B4*B2, C4*C2, D4*D2, E4*E2, F4*F2)
5	Dr. Kenneth N. Rosenbaum, MD	=ASK("score: 15miles < 0", A5, B1, "A")	=ASK("score: hard < easy", A5, C1, "A")	=ASK("score: 10 < 40", A5, D1, "A")	=ASK("score: little < much", A5, E1, "A")	=ASK("score: 10' < 40'", A5, F1, "A")	=SUM(B5*B2, C5*C2, D5*D2, E5*E2, F5*F2)
6	Dr. Shabnam Foroughi, MD	=ASK("score: 15miles < 0", A6, B1, "A")	=ASK("score: hard < easy", A6, C1, "A")	=ASK("score: 10 < 40", A6, D1, "A")	=ASK("score: little < much", A6, E1, "A")	=ASK("score: 10' < 40'", A6, F1, "A")	=SUM(B6*B2, C6*C2, D6*D2, E6*E2, F6*F2)
7	Dr. Nitya Ramachandran, MD	=ASK("score: 15miles < 0", A7, B1, "A")	=ASK("score: hard < easy", A7, C1, "A")	=ASK("score: 10 < 40", A7, D1, "A")	=ASK("score: little < much", A7, E1, "A")	=ASK("score: 10' < 40'", A7, F1, "A")	=SUM(B7*B2, C7*C2, D7*D2, E7*E2, F7*F2)
8	Dr. Neena I. Bhatti, MD	=ASK("score: 15miles < 0", A8, B1, "A")	=ASK("score: hard < easy", A8, C1, "A")	=ASK("score: 10 < 40", A8, D1, "A")	=ASK("score: little < much", A8, E1, "A")	=ASK("score: 10' < 40'", A8, F1, "A")	=SUM(B8*B2, C8*C2, D8*D2, E8*E2, F8*F2)
9	Dr. Rebecca M. Gross, MD	=ASK("score: 15miles < 0", A9, B1, "A")	=ASK("score: hard < easy", A9, C1, "A")	=ASK("score: 10 < 40", A9, D1, "A")	=ASK("score: little < much", A9, E1, "A")	=ASK("score: 10' < 40'", A9, F1, "A")	=SUM(B9*B2, C9*C2, D9*D2, E9*E2, F9*F2)
10	Dr. Marlorie P. Stinfil, MD	=ASK("score: 15miles < 0", A10, B1, "A")	=ASK("score: hard < easy", A10, C1, "A")	=ASK("score: 10 < 40", A10, D1, "A")	=ASK("score: little < much", A10, E1, "A")	=ASK("score: 10' < 40'", A10, F1, "A")	=SUM(B10*B2, C10*C2, D10*D2, E10*E2, F10*F2)
11	Dr. Louis H. Bland, MD	=ASK("score: 15miles < 0", A11, B1, "A")	=ASK("score: hard < easy", A11, C1, "A")	=ASK("score: 10 < 40", A11, D1, "A")	=ASK("score: little < much", A11, E1, "A")	=ASK("score: 10' < 40'", A11, F1, "A")	=SUM(B11*B2, C11*C2, D11*D2, E11*E2, F11*F2)
12	Dr. Gabe B. Mirkin, MD	=ASK("score: 15miles < 0", A12, B1, "A")	=ASK("score: hard < easy", A12, C1, "A")	=ASK("score: 10 < 40", A12, D1, "A")	=ASK("score: little < much", A12, E1, "A")	=ASK("score: 10' < 40'", A12, F1, "A")	=SUM(B12*B2, C12*C2, D12*D2, E12*E2, F12*F2)
13	max	=MAX(G3:G12)					
14	best	=IF(B13=0, "", INDEX(A3:A12, MATCH(B13, G3:G12, 0), 1))					

E.3. “Baby Girl Name”

	A	B	C	D	E
1		Less than 5% of girls born in 2013 in Maryland and Virginia have this name	No one notorious has same name (serial killer, thief, etc.)	Name can be reasonably well pronounced by Americans	score
2	*weights*	*25*	*50*	*25*	
3	Bianca	=ASK(A14:B15, A3, B1, "A")	=ASK(A14:B15, A3, C1, "B")	=ASK("score: sounds terrible < easily pronounced", A3, D1, "C")	=SUM(B3*B2, C3*C2, D3*D2)
4	Jennifer	=ASK(A14:B15, A4, B1, "A")	=ASK(A14:B15, A4, C1, "B")	=ASK("score: sounds terrible < easily pronounced", A4, D1, "C")	=SUM(B4*B2, C4*C2, D4*D2)
5	Sofia	=ASK(A14:B15, A5, B1, "A")	=ASK(A14:B15, A5, C1, "B")	=ASK("score: sounds terrible < easily pronounced", A5, D1, "C")	=SUM(B5*B2, C5*C2, D5*D2)
6	Clara	=ASK(A14:B15, A6, B1, "A")	=ASK(A14:B15, A6, C1, "B")	=ASK("score: sounds terrible < easily pronounced", A6, D1, "C")	=SUM(B6*B2, C6*C2, D6*D2)
7	Delfina	=ASK(A14:B15, A7, B1, "A")	=ASK(A14:B15, A7, C1, "B")	=ASK("score: sounds terrible < easily pronounced", A7, D1, "C")	=SUM(B7*B2, C7*C2, D7*D2)
8	Milana	=ASK(A14:B15, A8, B1, "A")	=ASK(A14:B15, A8, C1, "B")	=ASK("score: sounds terrible < easily pronounced", A8, D1, "C")	=SUM(B8*B2, C8*C2, D8*D2)
9	Raffaela	=ASK(A14:B15, A9, B1, "A")	=ASK(A14:B15, A9, C1, "B")	=ASK("score: sounds terrible < easily pronounced", A9, D1, "C")	=SUM(B9*B2, C9*C2, D9*D2)
10	Gioia	=ASK(A14:B15, A10, B1, "A")	=ASK(A14:B15, A10, C1, "B")	=ASK("score: sounds terrible < easily pronounced", A10, D1, "C")	=SUM(B10*B2, C10*C2, D10*D2)
11	max	=MAX(E3:E10)			
12	best	=IF(B11=0, "", INDEX(A3:A10, MATCH(B11, E3:E10, 0), 1))			
13					
14	Yes	1			
15	No	0			

E.4. “New Home”

	A	B	C	D	E	F
1		number of bed rooms	number of bath rooms	driving time to University of Maryland, College Park 20742	year of built	score
2	*weights*	*3*	*3*	*4*	*3*	
3	9408 Nicklaus Ln #88, Laurel, MD 20708	=ASK("2 to 3", A3, B1, "A")	=ASK("2 to 3", A3, C1, "A")	=ASK("15 to 35", A3, D1, "B")	=ASK("1980 to 2014", A3, E1, "A")	=SUM(B3*B2, C3*C2, D3*D2, E3*E2)
4	9311 Player Dr #114, Laurel, MD 20708	=ASK("2 to 3", A4, B1, "A")	=ASK("2 to 3", A4, C1, "A")	=ASK("15 to 35", A4, D1, "B")	=ASK("1980 to 2014", A4, E1, "A")	=SUM(B4*B2, C4*C2, D4*D2, E4*E2)
5	9268 Cherry Ln #62, Laurel, MD 20708	=ASK("2 to 3", A5, B1, "A")	=ASK("2 to 3", A5, C1, "A")	=ASK("15 to 35", A5, D1, "B")	=ASK("1980 to 2014", A5, E1, "A")	=SUM(B5*B2, C5*C2, D5*D2, E5*E2)
6	11352 Cherry Hill Rd Unit 301, Beltsville, MD 20705	=ASK("2 to 3", A6, B1, "A")	=ASK("2 to 3", A6, C1, "A")	=ASK("15 to 35", A6, D1, "B")	=ASK("1980 to 2014", A6, E1, "A")	=SUM(B6*B2, C6*C2, D6*D2, E6*E2)
7	4409 Romlon St Apt 201, Beltsville, MD 20705	=ASK("2 to 3", A7, B1, "A")	=ASK("2 to 3", A7, C1, "A")	=ASK("15 to 35", A7, D1, "B")	=ASK("1980 to 2014", A7, E1, "A")	=SUM(B7*B2, C7*C2, D7*D2, E7*E2)
8	11336 Cherry Hill Rd Unit 230, Beltsville, MD 20705	=ASK("2 to 3", A8, B1, "A")	=ASK("2 to 3", A8, C1, "A")	=ASK("15 to 35", A8, D1, "B")	=ASK("1980 to 2014", A8, E1, "A")	=SUM(B8*B2, C8*C2, D8*D2, E8*E2)
9	11354 Cherry Hill Rd # 1X103, Beltsville, MD 20705	=ASK("2 to 3", A9, B1, "A")	=ASK("2 to 3", A9, C1, "A")	=ASK("15 to 35", A9, D1, "B")	=ASK("1980 to 2014", A9, E1, "A")	=SUM(B9*B2, C9*C2, D9*D2, E9*E2)
10	4509 Romlon St Apt 302, Beltsville, MD 20705	=ASK("2 to 3", A10, B1, "A")	=ASK("2 to 3", A10, C1, "A")	=ASK("15 to 35", A10, D1, "B")	=ASK("1980 to 2014", A10, E1, "A")	=SUM(B10*B2, C10*C2, D10*D2, E10*E2)
11	4505 Romlon St Apt 204, Beltsville, MD 20705	=ASK("2 to 3", A11, B1, "A")	=ASK("2 to 3", A11, C1, "A")	=ASK("15 to 35", A11, D1, "B")	=ASK("1980 to 2014", A11, E1, "A")	=SUM(B11*B2, C11*C2, D11*D2, E11*E2)
12	11403 Falcon Ridge Ct, Beltsville, MD 20705	=ASK("2 to 3", A12, B1, "A")	=ASK("2 to 3", A12, C1, "A")	=ASK("15 to 35", A12, D1, "B")	=ASK("1980 to 2014", A12, E1, "A")	=SUM(B12*B2, C12*C2, D12*D2, E12*E2)
13	max	=MAX(F3:F12)				
14	best	=IF(B13=0, "", INDEX(A3:A12, MATCH(B13, F3:F12, 0), 1))				

E.5. “Buying a new laptop”

	A	B	C	D	E	F	G	H	I
1		screen size of 13" or larger	full size keyboard	dvd rom drive	price	no customization needed	average review	ok?	score
2	*weights*		*30*	*20*	*-50*	*20*			
3	Lenovo IdeaPad U310 13.3-Inch Touchscreen Ultrabook (Graphite Gray)	=ASK(A9:B10, A3, B1, "A")	=ASK(A9:B10, A3, C1, "A")	=ASK(A9:B10, A3, D1, "A")	=ASK("$400.00 to $1500.00", A3, E1, "A")	=ASK(A9:B10, A3, F1, "A")	=ASK(A12:B22, A3, G1, "A")	=AND(B3=1, E3<1500, G3>=4)	=SUM(C3*C2, D3*D2, E3*E2, F3*F2) * H3
4	Apple MacBook Air MD760LL/B 13.3-Inch Laptop (NEWEST VERSION)	=ASK(A9:B10, A4, B1, "A")	=ASK(A9:B10, A4, C1, "A")	=ASK(A9:B10, A4, D1, "A")	=ASK("$400.00 to $1500.00", A4, E1, "A")	=ASK(A9:B10, A4, F1, "A")	=ASK(A12:B22, A4, G1, "A")	=AND(B4=1, E4<1500, G4>=4)	=SUM(C4*C2, D4*D2, E4*E2, F4*F2) * H4
5	ASUS X750JA-DB71 17.3-Inch Laptop (Dark Gray)	=ASK(A9:B10, A5, B1, "A")	=ASK(A9:B10, A5, C1, "A")	=ASK(A9:B10, A5, D1, "A")	=ASK("$400.00 to $1500.00", A5, E1, "A")	=ASK(A9:B10, A5, F1, "A")	=ASK(A12:B22, A5, G1, "A")	=AND(B5=1, E5<1500, G5>=4)	=SUM(C5*C2, D5*D2, E5*E2, F5*F2) * H5
6	max	=MAX(I3:I5)							
7	best	=IF(B6=0, "", INDEX(A3:A5, MATCH(B6, I3:I5, 0), 1))							
8									
9	Yes	1							
10	No	0							
11									
12	0.0 stars	0.0							
13	0.5 stars	0.5							
14	1.0 stars	1.0							
15	1.5 stars	1.5							
16	2.0 stars	2.0							
17	2.5 stars	2.5							
18	3.0 stars	3.0							
19	3.5 stars	3.5							
20	4.0 stars	4.0							
21	4.5 stars	4.5							
22	5.0 stars	5.0							

E.6. “Which car to purchase”

	A	B	C	D
1		Price	Reliability	ok?
2	Honda	=ASK("$10000.00 to $25000.00", A2, B1, "A")	=ASK(A7:B17, A2, C1, "B")	=AND(C2>4)
3	Toyota	=ASK("$10000.00 to $25000.00", A3, B1, "A")	=ASK(A7:B17, A3, C1, "B")	=AND(C3>4)
4	Hyundai	=ASK("$10000.00 to $25000.00", A4, B1, "A")	=ASK(A7:B17, A4, C1, "B")	=AND(C4>4)
5	any ok?	=OR(D2:D4)		
6				
7	0.0 stars	0.0		
8	0.5 stars	0.5		
9	1.0 stars	1.0		
10	1.5 stars	1.5		
11	2.0 stars	2.0		
12	2.5 stars	2.5		
13	3.0 stars	3.0		
14	3.5 stars	3.5		
15	4.0 stars	4.0		
16	4.5 stars	4.5		
17	5.0 stars	5.0		

E.7. “Dog Breeds”

	A	B	C	D	E	F	G	H	I	J	K
1		typical height and length in inches	typical weight	typical life span	shedding	known for intelligence/ trainability	known for being affectionate and loyal	bred for retrieving	bred for guarding	ok?	score
2	*weights*	*5*	*4*	*3*	*-4*	*5*	*5*	*1*	*2*		
3	Golden Retriever	=ASK("11 to 34", A3, B1, "A")	=ASK(A21:B22, A3, C1, "A")	=ASK("8 to 20", A3, D1, "A")	=ASK("score: < ", A3, E1, "B")	=ASK("score: < ", A3, F1, "B")	=ASK("score: < ", A3, G1, "B")	=ASK(A21:B22, A3, H1, "B")	=ASK(A21:B22, A3, I1, "B")	=AND(AND(B3>=18, B3<=30), AND(C3>=30, C3<=60))	=SUM(B3*B2, C3*C2, D3*D2, E3*E2, F3*F2, G3*G2, H3*H2, I3*I2) * J3
4	Labrador Retriever	=ASK("11 to 34", A4, B1, "A")	=ASK(A21:B22, A4, C1, "A")	=ASK("8 to 20", A4, D1, "A")	=ASK("score: < ", A4, E1, "B")	=ASK("score: < ", A4, F1, "B")	=ASK("score: < ", A4, G1, "B")	=ASK(A21:B22, A4, H1, "B")	=ASK(A21:B22, A4, I1, "B")	=AND(AND(B4>=18, B4<=30), AND(C4>=30, C4<=60))	=SUM(B4*B2, C4*C2, D4*D2, E4*E2, F4*F2, G4*G2, H4*H2, I4*I2) * J4
5	Siberian Husky	=ASK("11 to 34", A5, B1, "A")	=ASK(A21:B22, A5, C1, "A")	=ASK("8 to 20", A5, D1, "A")	=ASK("score: < ", A5, E1, "B")	=ASK("score: < ", A5, F1, "B")	=ASK("score: < ", A5, G1, "B")	=ASK(A21:B22, A5, H1, "B")	=ASK(A21:B22, A5, I1, "B")	=AND(AND(B5>=18, B5<=30), AND(C5>=30, C5<=60))	=SUM(B5*B2, C5*C2, D5*D2, E5*E2, F5*F2, G5*G2, H5*H2, I5*I2) * J5
6	German Shepherd	=ASK("11 to 34", A6, B1, "A")	=ASK(A21:B22, A6, C1, "A")	=ASK("8 to 20", A6, D1, "A")	=ASK("score: < ", A6, E1, "B")	=ASK("score: < ", A6, F1, "B")	=ASK("score: < ", A6, G1, "B")	=ASK(A21:B22, A6, H1, "B")	=ASK(A21:B22, A6, I1, "B")	=AND(AND(B6>=18, B6<=30), AND(C6>=30, C6<=60))	=SUM(B6*B2, C6*C2, D6*D2, E6*E2, F6*F2, G6*G2, H6*H2, I6*I2) * J6
7	Border Collie	=ASK("11 to 34", A7, B1, "A")	=ASK(A21:B22, A7, C1, "A")	=ASK("8 to 20", A7, D1, "A")	=ASK("score: < ", A7, E1, "B")	=ASK("score: < ", A7, F1, "B")	=ASK("score: < ", A7, G1, "B")	=ASK(A21:B22, A7, H1, "B")	=ASK(A21:B22, A7, I1, "B")	=AND(AND(B7>=18, B7<=30), AND(C7>=30, C7<=60))	=SUM(B7*B2, C7*C2, D7*D2, E7*E2, F7*F2, G7*G2, H7*H2, I7*I2) * J7
8	Australian Shepherd	=ASK("11 to 34", A8, B1, "A")	=ASK(A21:B22, A8, C1, "A")	=ASK("8 to 20", A8, D1, "A")	=ASK("score: < ", A8, E1, "B")	=ASK("score: < ", A8, F1, "B")	=ASK("score: < ", A8, G1, "B")	=ASK(A21:B22, A8, H1, "B")	=ASK(A21:B22, A8, I1, "B")	=AND(AND(B8>=18, B8<=30), AND(C8>=30, C8<=60))	=SUM(B8*B2, C8*C2, D8*D2, E8*E2, F8*F2, G8*G2, H8*H2, I8*I2) * J8
9	Aidi	=ASK("11 to 34", A9, B1, "A")	=ASK(A21:B22, A9, C1, "A")	=ASK("8 to 20", A9, D1, "A")	=ASK("score: < ", A9, E1, "B")	=ASK("score: < ", A9, F1, "B")	=ASK("score: < ", A9, G1, "B")	=ASK(A21:B22, A9, H1, "B")	=ASK(A21:B22, A9, I1, "B")	=AND(AND(B9>=18, B9<=30), AND(C9>=30, C9<=60))	=SUM(B9*B2, C9*C2, D9*D2, E9*E2, F9*F2, G9*G2, H9*H2, I9*I2) * J9
10	Belgian Shepherd Tervuren	=ASK("11 to 34", A10, B1, "A")	=ASK(A21:B22, A10, C1, "A")	=ASK("8 to 20", A10, D1, "A")	=ASK("score: < ", A10, E1, "B")	=ASK("score: < ", A10, F1, "B")	=ASK("score: < ", A10, G1, "B")	=ASK(A21:B22, A10, H1, "B")	=ASK(A21:B22, A10, I1, "B")	=AND(AND(B10>=18, B10<=30), AND(C10>=30, C10<=60))	=SUM(B10*B2, C10*C2, D10*D2, E10*E2, F10*F2, G10*G2, H10*H2, I10*I2) * J10
11	Bearded Collie	=ASK("11 to 34", A11, B1, "A")	=ASK(A21:B22, A11, C1, "A")	=ASK("8 to 20", A11, D1, "A")	=ASK("score: < ", A11, E1, "B")	=ASK("score: < ", A11, F1, "B")	=ASK("score: < ", A11, G1, "B")	=ASK(A21:B22, A11, H1, "B")	=ASK(A21:B22, A11, I1, "B")	=AND(AND(B11>=18, B11<=30), AND(C11>=30, C11<=60))	=SUM(B11*B2, C11*C2, D11*D2, E11*E2, F11*F2, G11*G2, H11*H2, I11*I2) * J11
12	Flat-Coated Retriever	=ASK("11 to 34", A12, B1, "A")	=ASK(A21:B22, A12, C1, "A")	=ASK("8 to 20", A12, D1, "A")	=ASK("score: < ", A12, E1, "B")	=ASK("score: < ", A12, F1, "B")	=ASK("score: < ", A12, G1, "B")	=ASK(A21:B22, A12, H1, "B")	=ASK(A21:B22, A12, I1, "B")	=AND(AND(B12>=18, B12<=30), AND(C12>=30, C12<=60))	=SUM(B12*B2, C12*C2, D12*D2, E12*E2, F12*F2, G12*G2, H12*H2, I12*I2) * J12
13	Garafiano Shepherd	=ASK("11 to 34", A13, B1, "A")	=ASK(A21:B22, A13, C1, "A")	=ASK("8 to 20", A13, D1, "A")	=ASK("score: < ", A13, E1, "B")	=ASK("score: < ", A13, F1, "B")	=ASK("score: < ", A13, G1, "B")	=ASK(A21:B22, A13, H1, "B")	=ASK(A21:B22, A13, I1, "B")	=AND(AND(B13>=18, B13<=30), AND(C13>=30, C13<=60))	=SUM(B13*B2, C13*C2, D13*D2, E13*E2, F13*F2, G13*G2, H13*H2, I13*I2) * J13
14	Basque Shepherd	=ASK("11 to 34", A14, B1, "A")	=ASK(A21:B22, A14, C1, "A")	=ASK("8 to 20", A14, D1, "A")	=ASK("score: < ", A14, E1, "B")	=ASK("score: < ", A14, F1, "B")	=ASK("score: < ", A14, G1, "B")	=ASK(A21:B22, A14, H1, "B")	=ASK(A21:B22, A14, I1, "B")	=AND(AND(B14>=18, B14<=30), AND(C14>=30, C14<=60))	=SUM(B14*B2, C14*C2, D14*D2, E14*E2, F14*F2, G14*G2, H14*H2, I14*I2) * J14
15	Shiba Inu	=ASK("11 to 34", A15, B1, "A")	=ASK(A21:B22, A15, C1, "A")	=ASK("8 to 20", A15, D1, "A")	=ASK("score: < ", A15, E1, "B")	=ASK("score: < ", A15, F1, "B")	=ASK("score: < ", A15, G1, "B")	=ASK(A21:B22, A15, H1, "B")	=ASK(A21:B22, A15, I1, "B")	=AND(AND(B15>=18, B15<=30), AND(C15>=30, C15<=60))	=SUM(B15*B2, C15*C2, D15*D2, E15*E2, F15*F2, G15*G2, H15*H2, I15*I2) * J15
16	Shikoku	=ASK("11 to 34", A16, B1, "A")	=ASK(A21:B22, A16, C1, "A")	=ASK("8 to 20", A16, D1, "A")	=ASK("score: < ", A16, E1, "B")	=ASK("score: < ", A16, F1, "B")	=ASK("score: < ", A16, G1, "B")	=ASK(A21:B22, A16, H1, "B")	=ASK(A21:B22, A16, I1, "B")	=AND(AND(B16>=18, B16<=30), AND(C16>=30, C16<=60))	=SUM(B16*B2, C16*C2, D16*D2, E16*E2, F16*F2, G16*G2, H16*H2, I16*I2) * J16
17	Bernese Mountain Dog	=ASK("11 to 34", A17, B1, "A")	=ASK(A21:B22, A17, C1, "A")	=ASK("8 to 20", A17, D1, "A")	=ASK("score: < ", A17, E1, "B")	=ASK("score: < ", A17, F1, "B")	=ASK("score: < ", A17, G1, "B")	=ASK(A21:B22, A17, H1, "B")	=ASK(A21:B22, A17, I1, "B")	=AND(AND(B17>=18, B17<=30), AND(C17>=30, C17<=60))	=SUM(B17*B2, C17*C2, D17*D2, E17*E2, F17*F2, G17*G2, H17*H2, I17*I2) * J17
18	max	=MAX(K3:K17)									
19	best	=IF(B18=0, "", INDEX(A3:A17, MATCH(B18, K3:K17, 0), 1))									
20											
21	Yes	1									
22	No	0									

E.8. “Car Rental”

	A	B	C	D	E	F	G
1		Is Full-Size SUV OR Mini-Van OR Cargo Truck	Pick-up car rental location distance from zip code 95120	Pick-up car rental location offers one-way rental to airport code SNA	Availability on Saturday OR Sunday OR Monday	Total Price ($)	score
2	*weights*	*1*	*1*	*1*	*0.5*	*-1*	
3	Dollar Rent-A-Car - San Jose Airport (Airport Code: SJC)	=ASK(A26:B27, A3, B1, "A")	=ASK("0 to 50", A3, C1, "A")	=ASK(A26:B27, A3, D1, "A")	=ASK(A26:B27, A3, E1, "A")	=ASK("100 to 600", A3, F1, "A")	=SUM(B3*B2, C3*C2, D3*D2, E3*E2, F3*F2)
4	Thrifty Car Rental - San Jose Airport (Airport Code: SJC)	=ASK(A26:B27, A4, B1, "A")	=ASK("0 to 50", A4, C1, "A")	=ASK(A26:B27, A4, D1, "A")	=ASK(A26:B27, A4, E1, "A")	=ASK("100 to 600", A4, F1, "A")	=SUM(B4*B2, C4*C2, D4*D2, E4*E2, F4*F2)
5	Avis - San Jose Airport (Airport Code: SJC)	=ASK(A26:B27, A5, B1, "A")	=ASK("0 to 50", A5, C1, "A")	=ASK(A26:B27, A5, D1, "A")	=ASK(A26:B27, A5, E1, "A")	=ASK("100 to 600", A5, F1, "A")	=SUM(B5*B2, C5*C2, D5*D2, E5*E2, F5*F2)
6	Fox - San Jose Airport (Airport Code: SJC)	=ASK(A26:B27, A6, B1, "A")	=ASK("0 to 50", A6, C1, "A")	=ASK(A26:B27, A6, D1, "A")	=ASK(A26:B27, A6, E1, "A")	=ASK("100 to 600", A6, F1, "A")	=SUM(B6*B2, C6*C2, D6*D2, E6*E2, F6*F2)
7	Ace - San Jose Airport (Airport Code: SJC)	=ASK(A26:B27, A7, B1, "A")	=ASK("0 to 50", A7, C1, "A")	=ASK(A26:B27, A7, D1, "A")	=ASK(A26:B27, A7, E1, "A")	=ASK("100 to 600", A7, F1, "A")	=SUM(B7*B2, C7*C2, D7*D2, E7*E2, F7*F2)
8	National - San Jose Airport (Airport Code: SJC)	=ASK(A26:B27, A8, B1, "A")	=ASK("0 to 50", A8, C1, "A")	=ASK(A26:B27, A8, D1, "A")	=ASK(A26:B27, A8, E1, "A")	=ASK("100 to 600", A8, F1, "A")	=SUM(B8*B2, C8*C2, D8*D2, E8*E2, F8*F2)
9	National - Redwood City, CA	=ASK(A26:B27, A9, B1, "A")	=ASK("0 to 50", A9, C1, "A")	=ASK(A26:B27, A9, D1, "A")	=ASK(A26:B27, A9, E1, "A")	=ASK("100 to 600", A9, F1, "A")	=SUM(B9*B2, C9*C2, D9*D2, E9*E2, F9*F2)
10	National - San Francisco Airport (Airport Code: SFO)	=ASK(A26:B27, A10, B1, "A")	=ASK("0 to 50", A10, C1, "A")	=ASK(A26:B27, A10, D1, "A")	=ASK(A26:B27, A10, E1, "A")	=ASK("100 to 600", A10, F1, "A")	=SUM(B10*B2, C10*C2, D10*D2, E10*E2, F10*F2)
11	Avis - San Francisco Airport (Airport Code: SFO)	=ASK(A26:B27, A11, B1, "A")	=ASK("0 to 50", A11, C1, "A")	=ASK(A26:B27, A11, D1, "A")	=ASK(A26:B27, A11, E1, "A")	=ASK("100 to 600", A11, F1, "A")	=SUM(B11*B2, C11*C2, D11*D2, E11*E2, F11*F2)
12	Ace - San Francisco Airport (Airport Code: SFO)	=ASK(A26:B27, A12, B1, "A")	=ASK("0 to 50", A12, C1, "A")	=ASK(A26:B27, A12, D1, "A")	=ASK(A26:B27, A12, E1, "A")	=ASK("100 to 600", A12, F1, "A")	=SUM(B12*B2, C12*C2, D12*D2, E12*E2, F12*F2)
13	Fox - San Francisco Airport (Airport Code: SFO)	=ASK(A26:B27, A13, B1, "A")	=ASK("0 to 50", A13, C1, "A")	=ASK(A26:B27, A13, D1, "A")	=ASK(A26:B27, A13, E1, "A")	=ASK("100 to 600", A13, F1, "A")	=SUM(B13*B2, C13*C2, D13*D2, E13*E2, F13*F2)
14	Dollar Rent-A-Car - Oakland Airport (Airport Code: OAK)	=ASK(A26:B27, A14, B1, "A")	=ASK("0 to 50", A14, C1, "A")	=ASK(A26:B27, A14, D1, "A")	=ASK(A26:B27, A14, E1, "A")	=ASK("100 to 600", A14, F1, "A")	=SUM(B14*B2, C14*C2, D14*D2, E14*E2, F14*F2)
15	Thrifty - Oakland Airport (Airport Code: OAK)	=ASK(A26:B27, A15, B1, "A")	=ASK("0 to 50", A15, C1, "A")	=ASK(A26:B27, A15, D1, "A")	=ASK(A26:B27, A15, E1, "A")	=ASK("100 to 600", A15, F1, "A")	=SUM(B15*B2, C15*C2, D15*D2, E15*E2, F15*F2)
16	Avis - Oakland Airport (Airport Code: OAK)	=ASK(A26:B27, A16, B1, "A")	=ASK("0 to 50", A16, C1, "A")	=ASK(A26:B27, A16, D1, "A")	=ASK(A26:B27, A16, E1, "A")	=ASK("100 to 600", A16, F1, "A")	=SUM(B16*B2, C16*C2, D16*D2, E16*E2, F16*F2)
17	U-Haul - 15 Cottle Rd San Jose, CA 95123	=ASK(A26:B27, A17, B1, "A")	=ASK("0 to 50", A17, C1, "A")	=ASK(A26:B27, A17, D1, "A")	=ASK(A26:B27, A17, E1, "A")	=ASK("100 to 600", A17, F1, "A")	=SUM(B17*B2, C17*C2, D17*D2, E17*E2, F17*F2)
18	U-Haul - 705 Curtner Av San Jose, CA 95125	=ASK(A26:B27, A18, B1, "A")	=ASK("0 to 50", A18, C1, "A")	=ASK(A26:B27, A18, D1, "A")	=ASK(A26:B27, A18, E1, "A")	=ASK("100 to 600", A18, F1, "A")	=SUM(B18*B2, C18*C2, D18*D2, E18*E2, F18*F2)
19	U-Haul - 15367 Los Gatos Bl Los Gatos, CA 95032	=ASK(A26:B27, A19, B1, "A")	=ASK("0 to 50", A19, C1, "A")	=ASK(A26:B27, A19, D1, "A")	=ASK(A26:B27, A19, E1, "A")	=ASK("100 to 600", A19, F1, "A")	=SUM(B19*B2, C19*C2, D19*D2, E19*E2, F19*F2)
20	U-Haul - 2395 Senter Rd San Jose, CA 95112	=ASK(A26:B27, A20, B1, "A")	=ASK("0 to 50", A20, C1, "A")	=ASK(A26:B27, A20, D1, "A")	=ASK(A26:B27, A20, E1, "A")	=ASK("100 to 600", A20, F1, "A")	=SUM(B20*B2, C20*C2, D20*D2, E20*E2, F20*F2)
21	U-Haul - 1266 White Oaks Rd Campbell, CA 95008	=ASK(A26:B27, A21, B1, "A")	=ASK("0 to 50", A21, C1, "A")	=ASK(A26:B27, A21, D1, "A")	=ASK(A26:B27, A21, E1, "A")	=ASK("100 to 600", A21, F1, "A")	=SUM(B21*B2, C21*C2, D21*D2, E21*E2, F21*F2)
22	U-Haul - 17471 Farley Rd W Los Gatos, CA 95030	=ASK(A26:B27, A22, B1, "A")	=ASK("0 to 50", A22, C1, "A")	=ASK(A26:B27, A22, D1, "A")	=ASK(A26:B27, A22, E1, "A")	=ASK("100 to 600", A22, F1, "A")	=SUM(B22*B2, C22*C2, D22*D2, E22*E2, F22*F2)
23	max	=MAX(G3:G22)					
24	best	=IF(B23=0, "", INDEX(A3:A22, MATCH(B23, G3:G22, 0), 1))					
25							
26	True	1					
27	False	0					

E.9. Dishwasher

	A	B	C	D	E	F	G	H	I	J	K
1		has stainless steel door	has stainless tub inside dishwasher	has adjustable racks	has separate rack for flatware	holds 12 or more place settings	energy efficient labeling	price no more than $1200 in Seattle WA area	water consumption, gallons per normal operation	noise level during operation no more than 46	ok?
2	Asko D5634XLHS	=ASK(A13:B14, A2, B1, "A")	=ASK(A13:B14, A2, C1, "A")	=ASK(A13:B14, A2, D1, "A")	=ASK(A13:B14, A2, E1, "A")	=ASK(A13:B14, A2, F1, "A")	=ASK(A13:B14, A2, G1, "A")	=ASK(A13:B14, A2, H1, "B")	=ASK(A13:B14, A2, I1, "A")	=ASK(A13:B14, A2, J1, "B")	=AND(B2=1, C2=1, D2=1, E2=1, F2=1, G2=1, H2=1, I2=1, J2=1)
3	Bloomberg DW24100W	=ASK(A13:B14, A3, B1, "A")	=ASK(A13:B14, A3, C1, "A")	=ASK(A13:B14, A3, D1, "A")	=ASK(A13:B14, A3, E1, "A")	=ASK(A13:B14, A3, F1, "A")	=ASK(A13:B14, A3, G1, "A")	=ASK(A13:B14, A3, H1, "B")	=ASK(A13:B14, A3, I1, "A")	=ASK(A13:B14, A3, J1, "B")	=AND(B3=1, C3=1, D3=1, E3=1, F3=1, G3=1, H3=1, I3=1, J3=1)
4	Bosch SGE63E06UC	=ASK(A13:B14, A4, B1, "A")	=ASK(A13:B14, A4, C1, "A")	=ASK(A13:B14, A4, D1, "A")	=ASK(A13:B14, A4, E1, "A")	=ASK(A13:B14, A4, F1, "A")	=ASK(A13:B14, A4, G1, "A")	=ASK(A13:B14, A4, H1, "B")	=ASK(A13:B14, A4, I1, "A")	=ASK(A13:B14, A4, J1, "B")	=AND(B4=1, C4=1, D4=1, E4=1, F4=1, G4=1, H4=1, I4=1, J4=1)
5	Bosch SHE3ARL2UC	=ASK(A13:B14, A5, B1, "A")	=ASK(A13:B14, A5, C1, "A")	=ASK(A13:B14, A5, D1, "A")	=ASK(A13:B14, A5, E1, "A")	=ASK(A13:B14, A5, F1, "A")	=ASK(A13:B14, A5, G1, "A")	=ASK(A13:B14, A5, H1, "B")	=ASK(A13:B14, A5, I1, "A")	=ASK(A13:B14, A5, J1, "B")	=AND(B5=1, C5=1, D5=1, E5=1, F5=1, G5=1, H5=1, I5=1, J5=1)
6	Dacor RDW245	=ASK(A13:B14, A6, B1, "A")	=ASK(A13:B14, A6, C1, "A")	=ASK(A13:B14, A6, D1, "A")	=ASK(A13:B14, A6, E1, "A")	=ASK(A13:B14, A6, F1, "A")	=ASK(A13:B14, A6, G1, "A")	=ASK(A13:B14, A6, H1, "B")	=ASK(A13:B14, A6, I1, "A")	=ASK(A13:B14, A6, J1, "B")	=AND(B6=1, C6=1, D6=1, E6=1, F6=1, G6=1, H6=1, I6=1, J6=1)
7	Electrolux EIDW5905JS	=ASK(A13:B14, A7, B1, "A")	=ASK(A13:B14, A7, C1, "A")	=ASK(A13:B14, A7, D1, "A")	=ASK(A13:B14, A7, E1, "A")	=ASK(A13:B14, A7, F1, "A")	=ASK(A13:B14, A7, G1, "A")	=ASK(A13:B14, A7, H1, "B")	=ASK(A13:B14, A7, I1, "A")	=ASK(A13:B14, A7, J1, "B")	=AND(B7=1, C7=1, D7=1, E7=1, F7=1, G7=1, H7=1, I7=1, J7=1)
8	Frigidaire FBD2400KS	=ASK(A13:B14, A8, B1, "A")	=ASK(A13:B14, A8, C1, "A")	=ASK(A13:B14, A8, D1, "A")	=ASK(A13:B14, A8, E1, "A")	=ASK(A13:B14, A8, F1, "A")	=ASK(A13:B14, A8, G1, "A")	=ASK(A13:B14, A8, H1, "B")	=ASK(A13:B14, A8, I1, "A")	=ASK(A13:B14, A8, J1, "B")	=AND(B8=1, C8=1, D8=1, E8=1, F8=1, G8=1, H8=1, I8=1, J8=1)
9	Kitchen Aid KDFE454CSS	=ASK(A13:B14, A9, B1, "A")	=ASK(A13:B14, A9, C1, "A")	=ASK(A13:B14, A9, D1, "A")	=ASK(A13:B14, A9, E1, "A")	=ASK(A13:B14, A9, F1, "A")	=ASK(A13:B14, A9, G1, "A")	=ASK(A13:B14, A9, H1, "B")	=ASK(A13:B14, A9, I1, "A")	=ASK(A13:B14, A9, J1, "B")	=AND(B9=1, C9=1, D9=1, E9=1, F9=1, G9=1, H9=1, I9=1, J9=1)
10	General Electric GDF510PGDBB	=ASK(A13:B14, A10, B1, "A")	=ASK(A13:B14, A10, C1, "A")	=ASK(A13:B14, A10, D1, "A")	=ASK(A13:B14, A10, E1, "A")	=ASK(A13:B14, A10, F1, "A")	=ASK(A13:B14, A10, G1, "A")	=ASK(A13:B14, A10, H1, "B")	=ASK(A13:B14, A10, I1, "A")	=ASK(A13:B14, A10, J1, "B")	=AND(B10=1, C10=1, D10=1, E10=1, F10=1, G10=1, H10=1, I10=1, J10=1)
11	any ok?	=OR(K2:K10)									
12											
13	Yes	1									
14	No	0									

E.10. “Shaver”

	A	B	C	D	E	F	G	H	I
1		cordless	light weight of device	close shave	battery charge duration	Available battery replacement	Travel case included	price	score
2	*weights*	*5*	*3*	*6*	*4*	*3*	*4*		
3	Philips Norelco PT730	=ASK(A16:B17, A3, B1, "A")	=ASK("score: 50 < 200", A3, C1, "A")	=ASK(A19:B29, A3, D1, "A")	=ASK("20 to 60", A3, E1, "A")	=ASK(A16:B17, A3, F1, "A")	=ASK(A19:B29, A3, G1, "A")	=ASK("$40.00 to $100.00", A3, H1, "A")	=SUM(B3*B2, C3*C2, D3*D2, E3*E2, F3*F2, G3*G2)
4	Panasonic ES3831K	=ASK(A16:B17, A4, B1, "A")	=ASK("score: 50 < 200", A4, C1, "A")	=ASK(A19:B29, A4, D1, "A")	=ASK("20 to 60", A4, E1, "A")	=ASK(A16:B17, A4, F1, "A")	=ASK(A19:B29, A4, G1, "A")	=ASK("$40.00 to $100.00", A4, H1, "A")	=SUM(B4*B2, C4*C2, D4*D2, E4*E2, F4*F2, G4*G2)
5	Panasonic ES-RE51=S	=ASK(A16:B17, A5, B1, "A")	=ASK("score: 50 < 200", A5, C1, "A")	=ASK(A19:B29, A5, D1, "A")	=ASK("20 to 60", A5, E1, "A")	=ASK(A16:B17, A5, F1, "A")	=ASK(A19:B29, A5, G1, "A")	=ASK("$40.00 to $100.00", A5, H1, "A")	=SUM(B5*B2, C5*C2, D5*D2, E5*E2, F5*F2, G5*G2)
6	Braun 3Series 340S-4	=ASK(A16:B17, A6, B1, "A")	=ASK("score: 50 < 200", A6, C1, "A")	=ASK(A19:B29, A6, D1, "A")	=ASK("20 to 60", A6, E1, "A")	=ASK(A16:B17, A6, F1, "A")	=ASK(A19:B29, A6, G1, "A")	=ASK("$40.00 to $100.00", A6, H1, "A")	=SUM(B6*B2, C6*C2, D6*D2, E6*E2, F6*F2, G6*G2)
7	Philips Norelco PT724/31	=ASK(A16:B17, A7, B1, "A")	=ASK("score: 50 < 200", A7, C1, "A")	=ASK(A19:B29, A7, D1, "A")	=ASK("20 to 60", A7, E1, "A")	=ASK(A16:B17, A7, F1, "A")	=ASK(A19:B29, A7, G1, "A")	=ASK("$40.00 to $100.00", A7, H1, "A")	=SUM(B7*B2, C7*C2, D7*D2, E7*E2, F7*F2, G7*G2)
8	Panasonic ES8243A	=ASK(A16:B17, A8, B1, "A")	=ASK("score: 50 < 200", A8, C1, "A")	=ASK(A19:B29, A8, D1, "A")	=ASK("20 to 60", A8, E1, "A")	=ASK(A16:B17, A8, F1, "A")	=ASK(A19:B29, A8, G1, "A")	=ASK("$40.00 to $100.00", A8, H1, "A")	=SUM(B8*B2, C8*C2, D8*D2, E8*E2, F8*F2, G8*G2)
9	Philips Norelco 6948XLI/41	=ASK(A16:B17, A9, B1, "A")	=ASK("score: 50 < 200", A9, C1, "A")	=ASK(A19:B29, A9, D1, "A")	=ASK("20 to 60", A9, E1, "A")	=ASK(A16:B17, A9, F1, "A")	=ASK(A19:B29, A9, G1, "A")	=ASK("$40.00 to $100.00", A9, H1, "A")	=SUM(B9*B2, C9*C2, D9*D2, E9*E2, F9*F2, G9*G2)
10	Panasonic ES-RW30-S	=ASK(A16:B17, A10, B1, "A")	=ASK("score: 50 < 200", A10, C1, "A")	=ASK(A19:B29, A10, D1, "A")	=ASK("20 to 60", A10, E1, "A")	=ASK(A16:B17, A10, F1, "A")	=ASK(A19:B29, A10, G1, "A")	=ASK("$40.00 to $100.00", A10, H1, "A")	=SUM(B10*B2, C10*C2, D10*D2, E10*E2, F10*F2, G10*G2)
11	Wahl Professional 8061	=ASK(A16:B17, A11, B1, "A")	=ASK("score: 50 < 200", A11, C1, "A")	=ASK(A19:B29, A11, D1, "A")	=ASK("20 to 60", A11, E1, "A")	=ASK(A16:B17, A11, F1, "A")	=ASK(A19:B29, A11, G1, "A")	=ASK("$40.00 to $100.00", A11, H1, "A")	=SUM(B11*B2, C11*C2, D11*D2, E11*E2, F11*F2, G11*G2)
12	Braun Series 7-790cc	=ASK(A16:B17, A12, B1, "A")	=ASK("score: 50 < 200", A12, C1, "A")	=ASK(A19:B29, A12, D1, "A")	=ASK("20 to 60", A12, E1, "A")	=ASK(A16:B17, A12, F1, "A")	=ASK(A19:B29, A12, G1, "A")	=ASK("$40.00 to $100.00", A12, H1, "A")	=SUM(B12*B2, C12*C2, D12*D2, E12*E2, F12*F2, G12*G2)
13	max	=MAX(I3:I12)							
14	best	=IF(B13=0, "", INDEX(A3:A12, MATCH(B13, I3:I12, 0), 1))							
15									
16	Yes	1							
17	No	0							
18									
19	0.0 stars	0.0							
20	0.5 stars	0.5							
21	1.0 stars	1.0							
22	1.5 stars	1.5							
23	2.0 stars	2.0							
24	2.5 stars	2.5							
25	3.0 stars	3.0							
26	3.5 stars	3.5							
27	4.0 stars	4.0							
28	4.5 stars	4.5							
29	5.0 stars	5.0							

Appendix F. AskSheet field trial results

F.1. Doctor #1

Doctor's name	RateMDs	HealthGrades	CareFirst	Driving time	OK?		
Margaret S Shar	not needed	0	not needed	not needed	false		Accepted
Janet L Adams	3	4	not needed	10	false		Not accepted
Eduardo Arment	not needed	0	not needed	not needed	false		
Kiran Kumar	not needed	0	not needed	not needed	false		
Nasim Sarwar	not needed	0	not needed	5	false		
Shaaron Towns	4	5	Accepted	8	true		
Jolan S Rhodes	5	5	Accepted	12	true		
Mezgebe Haile	0	4	not needed	5	false		
Deborah Hanso	0	not needed	not needed	2	false		
Angela M Hubba	not needed	not needed	not needed	25	false		
Iradj Mahdavi	not needed	not needed	not needed	44	false		
Naseem Banu S	0	not needed	not needed	not needed	false		
Danita Tucker-P	0	not needed	not needed	not needed	false		
Gwendolyn V Yo	1	4	not needed	not needed	false		
Sukhveen K Ajra	0	0	not needed	not needed	false		
Polachira K Alex	3	2	not needed	not needed	false		
Markos Yibas	3	3	not needed	not needed	false		
Juliet Anandu	0	not needed	not needed	not needed	false		
Farira Baghai	not needed	not needed	Accepted	not needed	{false···		
Juliet C Naldo	not needed	not needed	Accepted	not needed	{false···		
Idalia Rosado	not needed	not needed	not needed	not needed	{false···		
Felicia Bassey-A	not needed	not needed	not needed	not needed	{false···		
	not		not				

Figure 34. This screenshot is truncated. The original contained 50 doctors. No data was collected for the rest. The result is Dr. Shaaron Towns, the first one with *true* in the *OK* column. The formulas are shown in Figure 24 on page 74.

F.2. Doctor #2

Doctor's name	RateMDs	HealthGrades	CareFirst	Driving time	OK?		
Margaret S Shar	0	0	Not accepted	not needed	false		Accepted
Janet L Adams	0	3	Not accepted	not needed	false		Not accepted
Eduardo Arment	0	0	Not accepted	not needed	false		
Kiran Kumar	0	0	Not accepted	not needed	false		
Nasim Sarwar	not needed	0	not needed	3	false		
Shaaron Towns	4	5	Accepted	9	true		
Jolan S Rhodes	not needed	5	Not accepted	13	false		
Mezgebe Haile	0	5	Accepted	7	false		
Deborah Hanso	not needed	not needed	Not accepted	26	false		
Angela M Hubba	not needed	not needed	Not accepted	5	false		
Iradj Mahdavi	not needed	not needed	Not accepted	5	false		
Naseem Banu S	not needed	not needed	Not accepted	5	false		
Danita Tucker-P	not needed	5	Not accepted	5	false		
Gwendolyn V Yo	not needed	3	not needed	not needed	false		
Sukhveen K Ajra	0	4	Not accepted	5	false		
Polachira K Alex	not needed	2	not needed	not needed	false		
Markos Yibas	3	4	Accepted	5	false		
Juliet Anandu	not needed	5	Not accepted	29	false		
Farira Baghai	not needed	0	not needed	not needed	false		
Juliet C Naldo	not needed	3	not needed	not needed	false		
Idalia Rosado	4	0	Accepted	14	false		
Felicia Bassey-A	0	4	Accepted	13	false		
Emma Yang	not needed	not needed	Accepted	31	false		
Rita Onyewueny	0	5	Accepted	8	false		
Hooshang Hodj	0	5	Accepted	8	false		
V V Nayar	not needed	0	not needed	not needed	false		
Twyla D Cathion	not needed	0	not needed	not needed	false		
Maislyn A Christ	0	4	Accepted	10	false		
Jeffrey P Bernste	4	4	Accepted	15	true		
Daniel Feldman	0	5	Accepted	19	false		
Samuel H Leibo	5	3	not needed	27	false		
Linda A Paxton	not needed	5	Accepted	15	{false--		
Robin G Witkin	not needed	4	not needed	15	{false--		
	not		not				

Figure 35. This screenshot is truncated. The original contained 50 doctors. No data was collected for the rest. The result is Dr. Shaaron Towns, the first one with *true* in the *OK* column. The formulas are shown in Figure 24 on page 74.

F.3. Doctor #3

Doctor's name	RateMDs	HealthGrades	CareFirst	Driving time	OK?		
Margaret S Shar	not needed	not needed	Not accepted	50	false		Accepted
Janet L Adams	3	not needed	not needed	not needed	false		Not accepted
Eduardo Armen	not needed	0	Not accepted	8	false		
Kiran Kumar	not needed	not needed	Not accepted	32	false		
Nasim Sarwar	not needed	0	Not accepted	4	false		
Shaaron Towns	4	5	Accepted	7	true		
Jolan S Rhodes	5	5	Accepted	11	true		
Mezgebe Haile	0	4	Accepted	5	false		
Deborah Hanso	not needed	0	not needed	25	false		
Angela M Hubba	not needed	4	Not accepted	6	false		
Iradj Mahdavi	not needed	0	Not accepted	6	false		
Naseem Banu S	not needed	not needed	Not accepted	6	false		
Danita Tucker-P	not needed	not needed	Not accepted	6	false		
Gwendolyn V Yo	not needed	3	Accepted	5	false		
Sukhveen K Ajra	not needed	4	Not accepted	5	false		
Polachira K Alex	not needed	0	Not accepted	6	false		
Markos Yibas	not needed	4	Not accepted	5	false		
Juliet Anandu	not needed	not needed	Not accepted	not needed	false		
Farira Baghai	0	0	not needed	14	false		
Juliet C Naldo	not needed	3	not needed	13	false		
Idalia Rosado	not needed	3	not needed	13	false		
Felicia Bassey-A	0	4	not needed	13	false		
Emma Yang	0	not needed	not needed	not needed	false		
Rita Onyewueny	0	not needed	not needed	not needed	false		
			not				

Figure 36. This screenshot is truncated. The original contained 50 doctors. No data was collected for the rest. The result is Dr. Shaaron Towns, the first one with *true* in the *OK* column. The formulas are shown in Figure 24 on page 74.

F.4. Smartphone #1

RESULTS

Status: Stopped

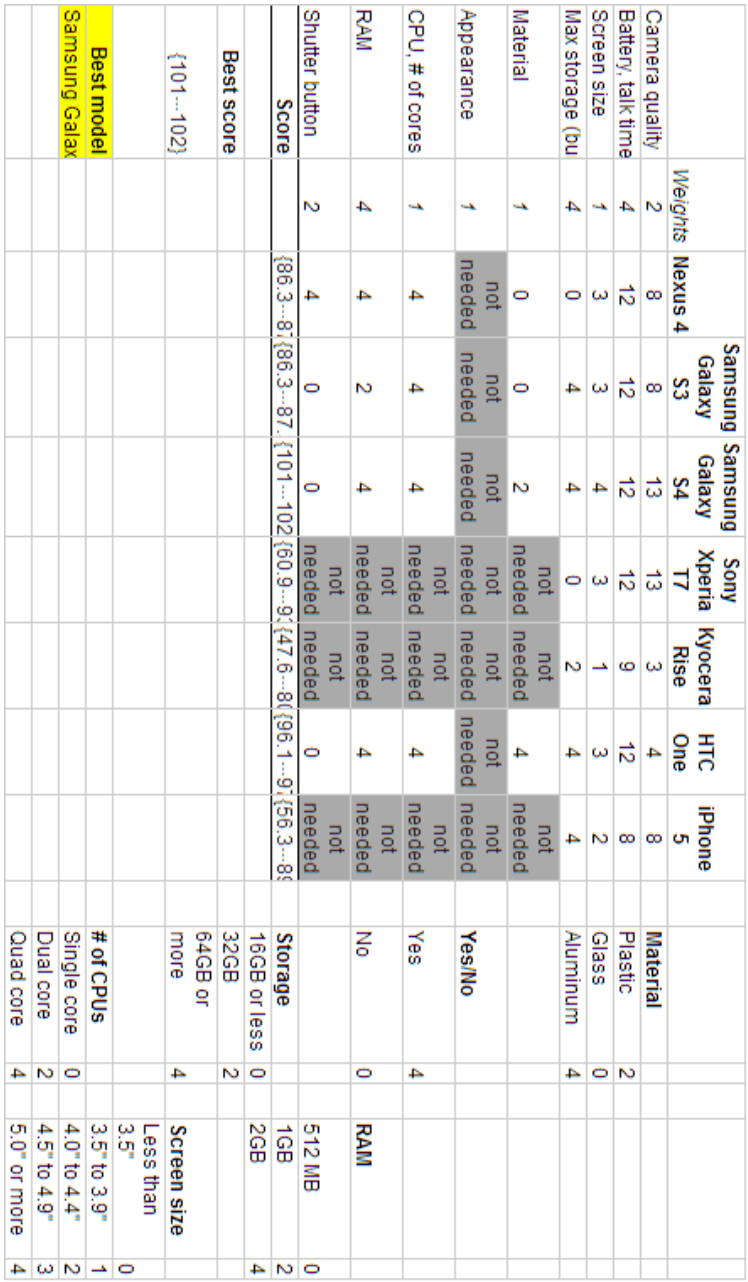
Start Stop Clear activity Force refresh

Role	Recipient	form	Number of inputs: total	=	empty	+	started	+	completed	+	abandoned	+	not needed	Mechanical Turk HITs: service	# open	# closed	link	reward
specs	AMT (production)	jdpy6y	56	=	0	+	0	+	40	+	0	+	16	production	0	5		$0.80
appearance	AMT (production)	65men7	7	=	0	+	0	+	0	+	0	+	7					

	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	Material		
Battery, talk time, hours	4	12	11	12	7	9	12	8	Plastic	2	
Screen size	1	not needed	3	4	not needed	not needed	3	not needed	Glass	0	
Max storage (built-in + card)	4	0	2	4	2	2	4	4	Aluminum	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	Yes/No		
CPU, # of cores	1	not needed	4	4	not needed	not needed	4	not needed	Yes	4	
RAM	4	4	2	4	2	0	4	2	No	0	RAM
Shutter button	2	not needed	0	0	not needed	not needed	0	not needed			512 MB 0
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}	Storage		1GB 2
									16GB or less	0	2GB 4
Best score									32GB	2	
{101···102}									64GB or more	4	Screen size
											Less than 3.5" 0
Best model									# of CPUs		3.5" to 3.9" 1
Samsung Galaxy S4									Single core	0	4.0" to 4.4" 2
									Dual core	2	4.5" to 4.9" 3
									Quad core	4	5.0" or more 4

Figure 37. The decision result is "Samsung Galaxy S4". The formulas are shown in Figure 26 on page 82.

F.5. Smartphone #2



RESULTS

Status: Stopped

Start Stop Clear activity Force refresh

Role	Recipient	form	Number of inputs: total	=	empty	+	started	+	completed	+	abandoned	+	not needed	Mechanical Turk HITs: service	# open	# closed	link	reward
Smartphone, thesis, 1 of 3 appearance	AMT (production)	mfwjfm	56	=	0	+	0	+	44	+	0	+	12	production	0	7		$0.80
	AMT (production)	ufjac9	7	=	0	+	0	+	0	+	0	+	7					

	Weights	Nexus 4	Samsung Galaxy S3	Samsung Galaxy S4	Sony Xperia T7	Kyocera Rise	HTC One	iPhone 5	
Camera quality	2	8	8	13	13	3	4	8	Material
Battery, talk time	4	12	12	12	12	9	12	8	Plastic
Screen size	1	3	3	4	3	1	3	2	Glass
Max storage (bu	4	0	4	4	0	2	4	4	Aluminum
Material	1	0	0	2	not needed	not needed	4	not needed	
Appearance	1	not needed	not needed	not needed	not needed	not needed	not needed	not needed	Yes/No
CPU, # of cores	1	4	4	4	not needed	not needed	4	not needed	Yes
RAM	4	4	2	4	not needed	not needed	4	not needed	No
Shutter button	2	4	0	0	not needed	not needed	0	not needed	
Score		{86.3…87.	{86.3…87.	{101…102	{60.9…91	{47.6…80	{96.1…97	{56.3…85	

Storage	
16GB or less	0
32GB	2
64GB or more	4

Screen size	
Less than 3.5"	0
3.5" to 3.9"	1
4.0" to 4.4"	2
4.5" to 4.9"	3
5.0" or more	4

RAM	
512 MB	0
1GB	2
2GB	4

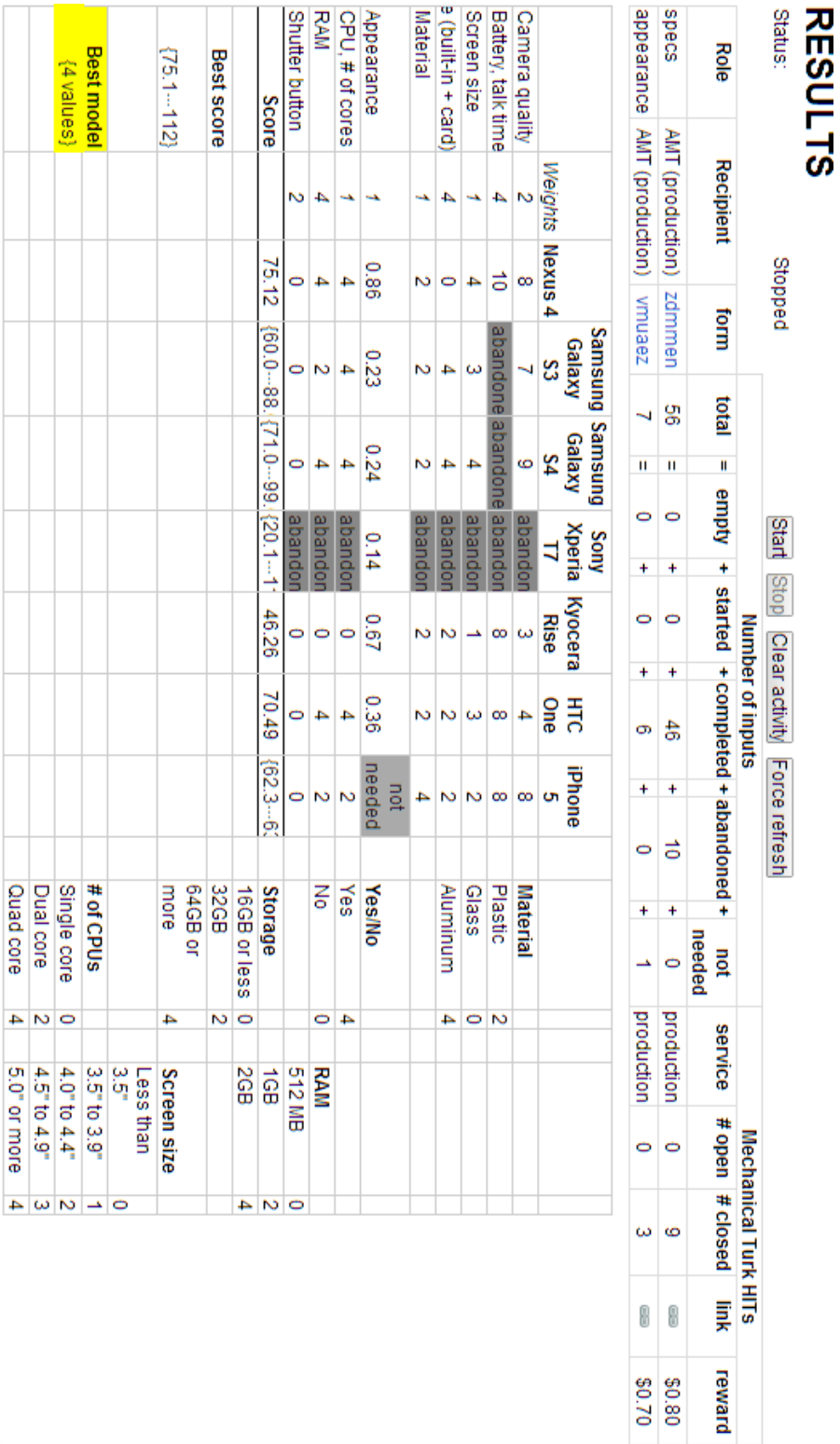
# of CPUs	
Single core	0
Dual core	2
Quad core	4

Best score
{101…102}

Best model
Samsung Galax

Figure 38. The decision result was “Samsung Galaxy S4”. Although the label is cut off, the value under “Best Score” indicates a range of “101…102” which corresponds to the S4 model. The formulas are shown in Figure 26 on page 82.

F.6. Smartphone #3



RESULTS

Status: Stopped

Start Stop Clear activity Force refresh

Role	Recipient	form	total	=	Number of inputs: empty	+	started	+	completed	+	abandoned	+	not needed	Mechanical Turk HITs: service	# open	# closed	link	reward
specs	AMT (production)	zdmmen	56	=	0	+	0	+	46	+	10	+	0	production	0	9		$0.80
appearance	AMT (production)	vmuaez	7	=	0	+	0	+	6	+	0	+	1	production	0	3		$0.70

	Weights	Nexus 4	Samsung Galaxy S3	Samsung Galaxy S4	Sony Xperia T7	Kyocera Rise	HTC One	iPhone 5			
Camera quality	2	8	7	9	abandon	3	4	8	**Material**		
Battery, talk time	4	10	abandone	abandone	abandon	8	8	8	Plastic	2	
Screen size	1	4	3	4	abandon	1	3	2	Glass	0	
e (built-in + card)	4	0	4	4	abandon	2	2	2	Aluminum	4	
Material	1	2	2	2	abandon	2	2	4			
Appearance	1	0.86	0.23	0.24	0.14	0.67	0.36	not needed	**Yes/No**		
CPU, # of cores	1	4	4	4	abandon	0	4	2	Yes	4	
RAM	4	4	2	4	abandon	0	4	2	No	0	**RAM**
Shutter button	2	0	0	0	abandon	0	0	0			512 MB
Score		75.12	{60.0···88.{	{71.0···99.	{20.1···1	46.26	70.49	{62.3···6	**Storage**		1GB
Best score									16GB or less	0	2GB
									32GB	2	
									64GB or more	4	**Screen size**
{75.1···112}											Less than 3.5"
Best model {4 values}									**# of CPUs**		3.5" to 3.9"
									Single core	0	4.0" to 4.4"
									Dual core	2	4.5" to 4.9"
									Quad core	4	5.0" or more

Figure 39. Due to a worker who entered blanks for several inputs, no final decision was reached. The possible ranges indicate that the Samsung Galaxy S4 would most likely be the result. The formulas are shown in Figure 26 on page 82.

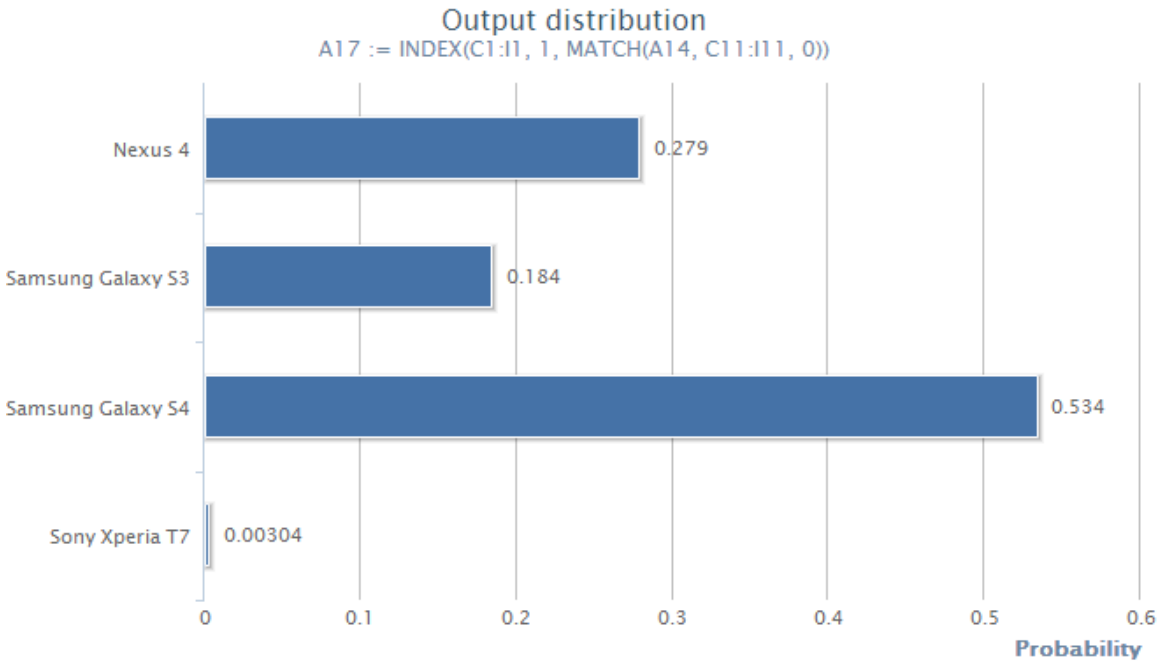


Figure 40. Based on the information entered for the third repetition of the smartphone example, the probability of the Samsung Galaxy S4 being chosen in the end is 0.534. That model was the decision result (winner) in the other two repititions.

F.7. Car #1

RESULTS

Status: Stopped Start Stop Clear activity Force refresh

			Number of inputs										Mechanical Turk HITs				
Role	Recipient	form	total	=	empty	+ started	+	completed	+ abandoned	+	not needed	service	# open	# closed	link	reward
hatchback	AMT (production)	hpuked	20	=	0	+ 0	+	14	+ 0	+	6	production	0	7		$0.50
MPG	AMT (production)	9sua9f	20	=	0	+ 0	+	11	+ 0	+	9	production	0	4		$0.50
features	AMT (production)	5dy2rh	60	=	0	+ 0	+	18	+ 0	+	42	production	0	4		$0.50
wheel base	AMT (production)	jin4df	20	=	0	+ 0	+	19	+ 0	+	1	production	0	7		$0.50

	Available as hatchback?	MPG, highway	Available with 4-wheel drive	Safety features	Seats 5 people comfortably?	Wheel base (inches)	score		
Volvo C30	Yes	30	No	0.24	0.29	104	0.93		Yes
Ford Fusion	not needed	not needed	not needed	not needed	not needed	112	0		No
Infiniti M37/M56	not needed	not needed	not needed	not needed	not needed	114	0		
Toyota Matrix	Yes	32	No	0.58	0.39	102	1.44		
Acura ILX	No	not needed	not needed	not needed	not needed	105	0		
Lexus ES 350	No	not needed	not needed	not needed	not needed	111	0		
Jeep Patriot	No	28	not needed	not needed	not needed	104	0		
Toyota Avalon	not needed	not needed	not needed	not needed	not needed	111	0		
Kia Optima	not needed	not needed	not needed	not needed	not needed	110	0		
Subaru Outback	Yes	30	Yes	0.86	0.60	107	2.35		
Hundai Tucson	Yes	26	No	0.61	0.86	103	1.7		
Lexus GS	No	not needed	not needed	not needed	not needed	not needed	0		
Volkswagen CC	Yes	32	No	0.30	0.14	106	0.92		
Honda CR-V	Yes	31	Yes	0.14	0.18	103	1.26		
Honda Accord	No	34	not needed	not needed	not needed	109	0		
Dodge Avenger	No	29	not needed	not needed	not needed	109	0		
Toyota Camry	No	35	not needed	not needed	not needed	100	0		
Lincoln MKS	not needed	not needed	not needed	not needed	not needed	112	0		
Subaru Legacy	No	32	not needed	not needed	not needed	108	0		
Hyundai Sonata	not needed	not needed	not needed	not needed	not needed	110	0		
							2.35		
						WINNER:	Subaru Outback		

Figure 41. The decision result was "Subaru Outback" although this was different on each of the three times this model was run due to some differences in the inputs that were entered. The formulas are shown in Figure 25 on page 48.

F.8. Car #2

RESULTS

Status: Stopped [Start] [Stop] [Clear activity] [Force refresh]

			Number of inputs										Mechanical Turk HITs				
Role	Recipient	form	total	=	empty	+ started	+ completed	+ abandoned	+ not needed	service	# open	# closed	link	reward			
hatchback	AMT (production)	nvnudj	20	=	0	+ 0	+ 12	+ 0	+ 8	production	0	5		$0.50			
MPG	AMT (production)	xjru93	20	=	0	+ 0	+ 5	+ 0	+ 15	production	0	2		$0.50			
features	AMT (production)	ivmsip	60	=	0	+ 0	+ 15	+ 0	+ 45	production	0	3		$0.50			
wheel base	AMT (production)	8gvdkb	20	=	0	+ 0	+ 20	+ 0	+ 0	production	0	5		$0.50			

	Available as hatchback?	MPG, highway	Available with 4-wheel drive	Safety features	Seats 5 people comfortably?	Wheel base (inches)	score		
Volvo C30	not needed	not needed	not needed	not needed	not needed	110	0		Yes
Ford Fusion	No	not needed	not needed	not needed	not needed	107	0		No
Infiniti M37/M56	No	not needed	not needed	not needed	not needed	100	0		
Toyota Matrix	Yes	32	Yes	0.32	0.63	105	1.93		
Acura ILX	No	not needed	not needed	not needed	not needed	105	0		
Lexus ES 350	not needed	not needed	not needed	not needed	not needed	111	0		
Jeep Patriot	No	not needed	not needed	not needed	not needed	103	0		
Toyota Avalon	not needed	not needed	not needed	not needed	not needed	111	0		
Kia Optima	not needed	not needed	not needed	not needed	not needed	110	0		
Subaru Outback	Yes	30	Yes	0.83	0.17	108	1.9		
Hundai Tucson	No	not needed	not needed	not needed	not needed	104	0		
Lexus GS	not needed	not needed	not needed	not needed	not needed	112	0		
Volkswagen CC	No	not needed	not needed	not needed	not needed	107	0		
Honda CR-V	Yes	31	Yes	0.63	0.82	103	2.38		
Honda Accord	No	not needed	not needed	not needed	not needed	109	0		
Dodge Avenger	Yes	29	Yes	0.17	0.60	109	1.62		
Toyota Camry	Yes	35	Yes	0.76	0.83	109	2.69		
Lincoln MKS	not needed	not needed	not needed	not needed	not needed	113	0		
Subaru Legacy	not needed	not needed	not needed	not needed	not needed	111	0		
Hyundai Sonata	not needed	not needed	not needed	not needed	not needed	110	0		
							2.69		
						WINNER:	Toyota Camry		

Figure 42. The decision result was "Toyota Camry" although this was different on each of the three times this model was run due to some differences in the inputs that were entered. The formulas are shown in Figure 25 on page 48.

F.9. Car #3

RESULTS

Status: Started (no activity) Start Stop Clear activity Force refresh

			Number of inputs											Mechanical Turk HITs				
Role	Recipient	form	total	=	empty	+	started	+	completed	+	abandoned	+	not needed	service	# open	# closed	link	reward
hatchback	AMT (production)	79am25	20	=	0	+	0	+	12	+	0	+	8	production	0	3		$0.50
MPG	AMT (production)	di44pb	20	=	0	+	0	+	10	+	0	+	10	production	0	2		$0.50
features	AMT (production)	73swjp	60	=	0	+	0	+	30	+	0	+	30	production	0	5		$0.50
wheel base	AMT (production)	dp63k7	20	=	0	+	0	+	20	+	0	+	0	production	0	5		$0.50

	Available as hatchback?	MPG, highway	Available with 4-wheel drive	Safety features	Seats 5 people comfortably?	Wheel base (inches)	score		
Volvo C30	Yes	30	Yes	0.52	0.42	103	1.83		Yes
Ford Fusion	not needed	not needed	not needed	not needed	not needed	112	0		No
Infiniti M37/M56	not needed	not needed	not needed	not needed	not needed	114	0		
Toyota Matrix	Yes	32	Yes	0.65	0.50	102	2.12		
Acura ILX	No	not needed	not needed	not needed	not needed	105	0		
Lexus ES 350	not needed	not needed	not needed	not needed	not needed	111	0		
Jeep Patriot	No	not needed	not needed	not needed	not needed	103	0		
Toyota Avalon	not needed	not needed	not needed	not needed	not needed	111	0		
Kia Optima	not needed	not needed	not needed	not needed	not needed	110	0		
Subaru Outback	Yes	30	Yes	0.09	0.25	107	1.24		
Hundai Tucson	Yes	29	Yes	0.91	0.91	103	2.67		
Lexus GS	not needed	not needed	not needed	not needed	not needed	112	0		
Volkswagen CC	Yes	31	Yes	0.24	0.76	106	1.94		
Honda CR-V	Yes	26	Yes	0.58	0.48	103	1.79		
Honda Accord	Yes	36	Yes	0.69	0.60	109	2.42		
Dodge Avenger	Yes	29	Yes	0.59	0.33	108	1.77		
Toyota Camry	Yes	35	Yes	0.46	0.32	109	1.87		
Lincoln MKS	not needed	not needed	not needed	not needed	not needed	112	0		
Subaru Legacy	Yes	27	Yes	0.67	0.09	108	1.54		
Hyundai Sonata	not needed	not needed	not needed	not needed	not needed	110	0		
							2.67		
						WINNER:	Hundai Tucson		

Figure 43. The decision result was "Toyota Camry" although this was different on each of the three times this model was run due to some differences in the inputs that were entered. The formulas are shown in Figure 25 on page 48.

References

Abraham, R., Burnett, M., & Erwig, M. (2008). Spreadsheet programming. In B. Wah (Ed.), *Wiley Encyclopedia of Computer Science and Engineering* (pp. 2804–2810). John Wiley & Sons, Inc.

Amershi, S., & Morris, M. R. (2008). CoSearch. In *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08* (p. 1647). New York, New York, USA: ACM Press. doi:10.1145/1357054.1357311

Ayalew, Y. (2001). *Spreadsheet Testing Using Interval Analysis*. Klagenfurt University.

Baker, K. R. (2012). *Optimization modeling with spreadsheets*. John Wiley & Sons.

Balakrishnan, N., Render, B., & Stair, R. M. (2006). *Managerial decision modeling with spreadsheets*. Prentice Hall.

Bernstein, M. S., Brandt, J., Miller, R. C., David, R., & Link, C. (2013). Crowds in two seconds : Enabling realtime crowd-powered Accessed Crowds in Two Seconds : Enabling Realtime Crowd-Powered Interfaces.

Bernstein, M. S., Little, G., Miller, R. C., Hartmann, B., Ackerman, M. S., Karger, D. R., … Panovich, K. (2010). Soylent: a word processor with a crowd inside. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology* (pp. 313–322).

Bodily, S. E. (1986). Spreadsheet Modeling as a Stepping Stone. *Interfaces*, *16*(5), 34–52.

Brown, P. S., & Gould, J. D. (1987). An experimental study of people creating spreadsheets. *ACM Transactions on Information Systems (TOIS)*, *5*(3), 258–272. doi:10.1145/27641.28058

Buchanan, L., & O Connell, A. (2006). A brief history of decision making. *Harvard Business Review*, *84*(1), 32. Retrieved from http://thegrcbluebook.com/wp-content/uploads/2013/02/A-Brief-History-of-Decision-Making-by-Leigh-Buchanan-and-Andrew-OConnell.pdf

Burnett, M. M., Atwood, J. W., Djang, R. W., Reichwein, J., Gottfried, H. J., & Yang, S. (2001). Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of Functional Programming*, *11*(2), 155–206.

Casella, G., & Berger, R. L. (2002). *Statistical Inference* (Second Edi.). Pacific Grover, CA: Duxbury.

Casimir, R. J. (1992). Real programmers don't use spreadsheets. In *ACM SIGPLAN Notices* (Vol. 27, pp. 10–16). ACM. doi:10.1145/130981.130982

Chan, K. T., King, I., & Yuen, M. (2009). Mathematical Modeling of Social Games Playing / Having Fun.

Chan, Y. E., & Storey, V. C. (1996). The use of spreadsheets in organizations: Determinants and consequences. *Information & Management*, *31*(3), 119–134. doi:10.1016/S0378-7206(96)00008-0

Cheatham, T. E., Holloway, G. H., & Townley, J. A. (1979). Symbolic evaluation and the analysis of programs. *Software Engineering, IEEE Transactions on*, (4), 402–417.

Cheng, R. C. H., & Amin, N. A. K. (1983). Estimating parameters in continuous univariate distributions with a shifted origin. *Journal of the Royal Statistical Society. Series B (Methodological)*, *45*(3), 394–403. Retrieved from http://www.jstor.org/stable/2345411

Chi, E. H., Pirolli, P., Chen, K., & Pitkow, J. (2001). Using information scent to model user information needs and actions and the Web. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 490–497). New York, New York, USA: ACM Press. doi:10.1145/365024.365325

Chilton, L. B., Little, G., Edge, D., Weld, D. S., & Landay, J. A. (2013). Cascade. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13* (p. 1999). New York, New York, USA: ACM Press. doi:10.1145/2470654.2466265

Cox, P. T., & Smedley, T. J. (1994). Using visual programming to extend the power of spreadsheet. In *Proceedings of the workshop on Advanced visual interfaces* (pp. 153–161). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=192343

Dai, P., Weld, D. S., & others. (2010). Decision-theoretic control of crowd-sourced workflows. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Eom, H. B., & Lee, S. M. (1990). A survey of decision support system applications (1971--April 1988). *Interfaces*, *20*(3), 65–79.

Eom, S. B., Lee, S. M., Kim, E. B., & Somarajan, C. (1998). A survey of decision support system applications (1988-1994). *Journal of the Operational Research Society*, *49*, 109–120. Retrieved from http://www.jstor.org/stable/3009977

Eom, S., & Kim, E. (2005). A survey of decision support system applications (1995–2001). *Journal of the Operational Research Society*, *57*(11), 1264–1278. doi:10.1057/palgrave.jors.2602140

Forman, E. H., Gass, S. I., & Smith, R. H. (2001). The Analytic Hierarchy Process – An Exposition 1, *49*(4), 469–486.

Franklin, M. J., Kossman, D., Kraska, T., Ramesh, S., & Xin, R. (2011). CrowdDB: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data* (pp. 61–72).

Frei, B. (2009). *Paid crowdsourcing: Current state & progress toward mainstream business use*. Retrieved from Smartsheet.com

Fujima, J., Yoshihara, S., & Tanaka, Y. (2007). Web application orchestration using Excel. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence* (pp. 743–749). Ieee. doi:10.1109/WI.2007.136

Fylstra, D., Lasdon, L., Watson, J., & Waren, A. (1998). Design and use of the Microsoft Excel Solver. *Interfaces*, *28*(5), 29–55.

Ginige, A., Paolino, L., Sebillo, M., Shrodkar, R., & Vitiello, G. (2010). User requirements for a web based spreadsheet-mediated collaboration. In *Proceedings of the International Conference on Advanced Visual Interfaces* (pp. 133–136).

Ginige, A., Paolino, L., Sebillo, M., Tortora, G., Romano, M., & Vitiello, G. (2010). A Collaborative Environment for Spreadsheet-Based Activities. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on* (pp. 273–274). Ieee. doi:10.1109/VLHCC.2010.55

Gladwell, M. (2007). *Blink: The power of thinking without thinking*. Hachette Digital, Inc.

Glueck, D., & Karimpour-Fard, A. (2008). Fast computation by block permanents of cumulative distribution functions of order statistics from several populations. *Communications in Statistics - Theory and Methods*, *37*(18). Retrieved from http://www.tandfonline.com/doi/abs/10.1080/03610920802001896#.U8V4h_nlqPU

Gottfried, H. J. H. J., & Burnett, M. M. M. (1997). Graphical definitions: making spreadsheets visual through direct manipulation and gestures. In *Proceedings of the 1997 IEEE Symposium onV isual Languages (VL)* (pp. 246–253). IEEE Comput. Soc. doi:10.1109/VL.1997.626590

Grossman, T. A., Mehrota, V., & Özluk, Ö. (2007). Lessons from Mission-Critical Spreadsheets. *Communications of the Association for Information Systems*, *20*(1), 1009–1042.

Hanus, M. (1997). A unified computation model for functional and logic programming. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 80–93).

Hendry, D. G., & Green, T. R. G. (1994). Creating, comprehending and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model. *International Journal of Human-Computer Studies*, *40*(6), 1033–1065.

Howard, R. (1966). Information Value Theory. *IEEE Transactions on Systems Science and Cybernetics*, *2*(1), 22–26. doi:10.1109/TSSC.1966.300074

Howe, J. (2006, June). The rise of crowdsourcing. *Wired Magazine*, (14). Retrieved from http://sistemas-humano-computacionais.wikidot.com/local--files/capitulo:redes-sociais/Howe_The_Rise_of_Crowdsourcing.pdf

Hu, C., Bederson, B. B., Resnik, P., & Kronrod, Y. (2011). MonoTrans2 : A New Human Computation System to Support Monolingual Translation, 1133–1136.

Ipeirotis, P. G., Provost, F., & Wang, J. (2010). Quality management on Amazon Mechanical Turk. *Proceedings of the ACM SIGKDD Workshop on Human Computation - HCOMP '10*, 64. doi:10.1145/1837885.1837906

Ipsilandis, P. G. (2008). Spreadsheet modelling for solving combinatorial problems: The vendor selection problem. In *Proceedings of European Spreadsheet Risks Interest Group (EuSpRIG)* (pp. 95–107).

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.

Kirkwood, C. W. (1997). *Strategic decision making*. Duxbury Press Belmont, CA.

Lai, K.-Y., Malone, T. W., & Yu, K.-C. (1988). Object lens: a "spreadsheet" for cooperative work. *ACM Transactions on Information Systems (TOIS)*, *6*(4), 332–353. doi:10.1145/62266.62276

Law, E. L. M., Von Ahn, L., Dannenberg, R. B., & Crawford, M. (2007). TagATune: A Game for Music and Sound Annotation. In *ISMIR* (Vol. 3, p. 2). Retrieved from http://www.cs.cmu.edu/~elaw/papers/ISMIR2007.pdf

Lawson, B. R., Baker, K. R., Powell, S. G., & Foster-Johnson, L. (2009). A comparison of spreadsheet users with different levels of experience☆. *Omega*, *37*(3), 579–590. doi:10.1016/j.omega.2007.12.004

Lenz, H. J., & Berlin, F. U. (2009). Spreadsheet Computation with imprecise and uncertain Data, 8421.

Lewis, C. (1985). Extending the spreadsheet interface to handle approximate quantities and relationships. In *ACM SIGCHI Bulletin* (Vol. 16, pp. 55–59).

Little, G., Chilton, L. B., Goldman, M., & Miller, R. C. (2010a). Exploring iterative and parallel human computation processes. In *Proceedings of the ACM SIGKDD Workshop on Human Computation - HCOMP '10* (pp. 68–76). New York: ACM. doi:10.1145/1753846.1754145

Little, G., Chilton, L. B., Goldman, M., & Miller, R. C. (2010b). Turkit: human computation algorithms on mechanical turk. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology* (pp. 57–66).

Lloyd, J. W. (1994). Practical advantages of declarative programming. In *Joint Conference on Declarative Programming, GULP-PRODE* (Vol. 94, p. 94). Retrieved from ftp://138.100.11.74/pub/papers/PARFORCE/second_review/D.WP3.1.M2.3.ps.Z

Marcus, A., Wu, E., Karger, D., Madden, S., & Miller, R. (2011a). Human-powered Sorts and Joins. *Proc. VLDB Endow.*, *5*(1), 13–24. Retrieved from http://dl.acm.org/citation.cfm?id=2047485.2047487

Marcus, A., Wu, E., Karger, D. R., Madden, S., & Miller, R. C. (2011b). Crowdsourced databases: Query processing with people.

Mather, D. (1999). A framework for building spreadsheet based decision models. *Journal of the Operational Research Society*, 70–74.

McGill, T. J., & Klobas, J. E. (2005). The role of spreadsheet knowledge in user-developed application success. *Decision Support Systems*, *39*(3), 355–369. doi:10.1016/j.dss.2004.01.002

Microsoft Corporation. (2003). *Microsoft Launches Visual Studio Tools for the Microsoft Office System*. Retrieved May 11, 2014, from http://www.microsoft.com/en-us/news/press/2003/oct03/10-13vstoofficelaunchpr.aspx?Search=true&mstLocPickShow=False

Morris, M. R., & Horvitz, E. (2007). SearchTogether. In *Proceedings of the 20th annual ACM symposium on User interface software and technology - UIST '07* (p. 3). New York, New York, USA: ACM Press. doi:10.1145/1294211.1294215

Morris, M. R., Lombardo, J., & Wigdor, D. (2010). WeSearch. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work - CSCW '10* (p. 401). New York, New York, USA: ACM Press. doi:10.1145/1718918.1718987

Myers, B. a. (1991). Graphical techniques in a spreadsheet for specifying user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 243–249). New York, New York, USA: ACM Press. doi:10.1145/108844.108903

Nardi, B. A., & Miller, J. R. (1990). An ethnographic study of distributed problem solving in spreadsheet development. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work* (pp. 197–208).

Nehzati, T., Ismail, N., & Rashidi-Bajgan, H. (2010). Developing Spreadsheet Based Decision Support System to Solve Warehouse Layout Problem.

Nunamaker, J. A. Y. F., Briggs, R. O., Mittleman, D. D., Vogel, D. R., & Balthazard, P. A. (1997). Lessons from a Dozen Years of Group Support Systems Research: A Discussion of Lab and Field Findings. *J. of Management Information Systems*, *13*(3), 163–207.

Nunamaker, J. F., Applegate, L. M., & Konsynski, B. R. (1988). Computer-Aided Deliberation: Model Management and Group Decision Support: Special Focus Article. *Operations Research*, *36*(6), 826–848.

Palmer, C. R., & Cormack, G. V. (1998). Operation transforms for a distributed shared spreadsheet. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work* (pp. 69–78).

Parameswaran, A., Teh, M. H., Garcia-Molina, H., & Widom, J. (2013). DataSift: An Expressive and Accurate Crowd-Powered Search Toolkit. In *First AAAI Conference on Human Computation and Crowdsourcing*. Retrieved from http://www.aaai.org/ocs/index.php/HCOMP/HCOMP13/paper/view/7500

Power, D. J., & Sharda, R. (2007). Model-driven decision support systems: Concepts and research directions. *Decision Support Systems*, *43*(3), 1044–1061. doi:10.1016/j.dss.2005.05.030

Quinn, A. J., & Bederson, B. B. (2011). Human Computation : A Survey and Taxonomy of a Growing Field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)* (pp. 1403–1412). ACM.

Quinn, A. J., Bederson, B. B., Yeh, T., & Lin, J. (2010). Crowdflow: Integrating machine learning with mechanical turk for speed-cost-quality flexibility. *Better Performance over Iterations*.

Ragsdale, C. T. (2014). *Spreadsheet Modeling and Decision Analysis: A Practical Introduction to Business Analytics* (7th ed.). Cengage Learning.

Ranneby, B. O. (2013). The Maximum Spacing Method . An Estimation Method Related to the Maximum Likelihood Method, *11*(2), 93–112.

Russell, D. M., Stefik, M. J., Pirolli, P., & Card, S. K. (1993). The cost structure of sensemaking. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems* (pp. 269–276).

Russell, S. J., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach* (Vol. 2, pp. 487–490). Prentice hall Englewood Cliffs.

Saaty, T. L. (1980). *The Analytic Hierarchy Process*.

Saaty, T. L., & Hall, M. (1999). Fundamentals of the analytic network process.

Scaffidi, C., Shaw, M., & Myers, B. (2005). Estimating the numbers of end users and end user programmers. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 207–214). IEEE. doi:10.1109/VLHCC.2005.34

Schwartz, B. (2003). *The Paradox of Choice*. Ecco.

Seref, M. H., Ahuja, R., & Winston, W. (2007). Developing spreadsheet-based decision support systems. *Dynamic Ideas*.

SERP (Spreadsheet Engineering Research Project). (2006). Retrieved from http://mba.tuck.dartmouth.edu/spreadsheet/product_pubs_files/SERPSurveyResults.doc

Sheng, V. S., & Provost, F. (2008). Get Another Label ? Improving Data Quality and Data Mining Using Multiple , Noisy Labelers Categories and Subject Descriptors.

Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological Review*, *63*(2), 129–138. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/13310708

Simon, H. A. (1972). Theories of Bounded Rationality. In C. B. McGuire & R. Radner (Eds.), *Decision and Organization* (Vol. 1, pp. 161–176). North-Holland Publishing Company.

Sipser, M. (1997). *Introduction to the theory of computation* (p. 396). PWS Pub. Co. Retrieved from http://books.google.com/books/about/Introduction_to_the_theory_of_computatio.html?id=cXcpAQAAMAAJ&pgis=1

Stadelmann, M. (1993). A spreadsheet based on constraints. In *Proceedings of the 6th annual ACM symposium on User interface software and technology* (pp. 217–224).

Stigler, & Stigler, G. J. (1961). The Economics Of Information. *Journal of Political Economy*, *69*(3), 213–225.

Streit, A., Pham, B., & Brown, R. (2008). A spreadsheet approach to facilitate visualization of uncertainty in information. *Visualization and Computer Graphics, IEEE Transactions on*, *14*(1), 61–72. doi:10.1109/TVCG.2007.70426

Tennent, J., & Friend, G. (2011). *Guide to business modelling* (Vol. 89). John Wiley & Sons.

Triantaphyllou, E. (2000). *Multi-criteria decision making methods*. Springer.

Troutt, M. D., Tadisina, S. K., & Clinton, R. J. (1991). Interactive optimization aspects of electronic spreadsheet models for design and planning. *Journal of the Operational Research Society*, *42*(5), 349–355.

Trushkowsky, B., Kraska, T., Franklin, M. J., & Sarkar, P. (2013). Crowdsourced Enumeration Queries. In *Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE)*.

US Bureau of Labor Statistics. (2001). *Current Population Study*. Retrieved May 11, 2014, from http://www.bls.gov/cps/ciuaw.pdf

US Bureau of Labor Statistics. (2003). Current Population Study. Retrieved from http://www.bls.gov/news.release/pdf/ciuaw.pdf

Vaidya, O. S., & Kumar, S. (2006). Analytic hierarchy process: An overview of applications. *European Journal of Operational Research*, *169*(1), 1–29. doi:10.1016/j.ejor.2004.04.028

Von Ahn, L. (2005). *Human Computation*. Carnegie Mellon University.

Von Ahn, L., & Dabbish, L. (2004). Labeling images with a computer game. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 319–326). New York, New York, USA: ACM Press. doi:10.1145/985692.985733

Vukovic, M., Lopez, M., & Laredo, J. (2010). PeopleCloud for the Globally Integrated Enterprise, 109–114.

Wang, J., Kraska, T., Franklin, M. J., & Feng, J. (2012). CrowdER: crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, *5*(11), 1483–1494. Retrieved from http://dl.acm.org/citation.cfm?id=2350229.2350263

Zanakis, S. H., Solomon, A., Wishart, N., & Dublish, S. (1998). Multi-attribute decision making: A simulation comparison of select methods. *European Journal of Operational Research*, *107*(3), 507–529. doi:10.1016/S0377-2217(97)00147-1

Zhang, H., Law, E., Miller, R., Gajos, K., Parkes, D., & Horvitz, E. (2012). Human computation tasks with global constraints. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12* (p. 217). New York, New York, USA: ACM Press. doi:10.1145/2207676.2207708