Dismantling Complicated Query Attributes with Crowd

Matan Laadan
Tel Aviv University
Tel Aviv, Israel
matanlaa@post.tau.ac.il

Tova Milo
Tel Aviv University
Tel Aviv, Israel
milo@cs.tau.ac.il

ABSTRACT

We study the problem of query evaluation with the help of the crowd, when the value of the queried attributes is not available in the database and is also hard for the crowd to estimate. Rather than asking users directly about these attributes, we propose a novel alternative approach that first uses the crowd to dismantle the query attributes into finer related ones (whose value estimation is easier), then assemble them to yield better estimation for the query attributes. We show that it is sometimes beneficial not to only dismantle the query attributes themselves, but rather to continue dismantling newly discovered attributes. We provide a careful statistical analysis to estimate the potential benefit (and cost) of dismantling each of the so-far-discovered attributes. Building on this analysis, we present an effective algorithm that balances between attributes dismantling and obtaining essential statistics about them (for estimating properties like "difficulty" and "contribution" of attributes) to decide how many crowd members should be asked about each attribute and how the answers should be assembled together. A thorough experimental analysis demonstrates the feasibility and effectiveness of the approach.

1. INTRODUCTION

We consider the problem of query evaluation with the help of the crowd, when the value of the query attributes is hard to estimate. Rather than asking users directly about these attributes, we propose a novel alternative approach that first uses the crowd to dismantle the query attributes into finer related ones (whose value estimation is easier), then assemble them to yield better estimation for the query attributes.

To illustrate, assume that we want to evaluate a query over a database of objects, testing and retrieving the values of certain attributes. This is a standard task when the attribute values are available explicitly in the database, but becomes challenging when they are not. For example, consider an imaginary cooking website CrowdCooking.com (CC) - a large recipes website where people can post their own recipes and other people can search and use them. Up until now, CC only allowed basic keyword search, but they now wish to upgrade their search capabilities to include more sophisticated searches, allowing people to search, e.g., for dessert recipes that are easy to make, have less than X calories and contain a certain amount of proteins. While NLP/text-analysis techniques could be used to evaluate some of these search criteria, this may be costly and inaccurate. An alternative approach that emerged in recent years is to use the crowd of web users for finding the value of the missing query attributes - Is a dish a dessert or not? How many calories/proteins does it contains? Etc. As crowd answers may be erroneous, a common approach is to ask multiple users about each missing query attribute and compute some aggregation (usually average/median) of the answers. The number of crowd members that need to be asked about a given attribute is typically determined by the difficulty of the question and the budget constraints[1]. For example, three users probably suffice to determine with a high probability that a certain dish is a dessert, but more are likely to be required to determine its number of calories. In fact, a key problem of this approach is that some attributes (like protein amount) are so difficult or un-intuitive for the crowd to evaluate, that the convergence to the final answer might be slow and thus require high budget [31]. Another disadvantage is that query attributes are handled separately and potential mutual information (for example between *dessert* and *calories*) which could be used to reduce the number of required questions, or to improve accuracy, is ignored.

A first solution to this problem was proposed in [27]. Instead of asking the crowd directly about the attributes mentioned in the query, it was suggested to also ask for the value of other (usually simpler) related attributes, and then derive the value of the query attributes from the answers. For example, to estimate the amount of protein in a certain dish one may ask what quantities of high protein ingredients (such as meat, dairy, eggs, nuts and soy) does it contains. In this solution, queries are processed in two steps: (1) An *offline* preprocessing phase that, given a query, determines which object's attributes should be asked about, how many users should estimate each of those attributes and how the obtained values should be assembled together, and (2) an *online* query evaluation phase, where each object in the database is processed using the scheme derived in the offline step. For example, consider a query about the protein

[1]In common crowdsourcing platforms, crowd questions have some (small) monetary cost, and thus the number of questions per object is typically bounded by the allocated budget

10.5441/002/edbt.2015.36

amount in recipes in CC. Assuming a budget of 20 questions per recipe, the preprocessing phase may derive a formula of the form: $0.5\mathit{protein\_amount}^{(10)} + 0.13\mathit{grams\_of\_meat}^{(3)} + 0.15\mathit{grams\_of\_dairy}^{(4)} + 3\mathit{number\_of\_eggs}^{(3)}$. In the formula, $\mathit{attribute\_name}^{(n)}$ denotes the average value of n users answers when asked about the given attribute. The formula indicates that rather than using the full budget to ask users directly about protein amount, a better estimation would be obtained by asking only 10 crowd members, then averaging the derived value with a linear combination of estimated values for three other related attributes - *grams_of_meat*, *grams_of_dairy*, *number_of_eggs* (computed by asking 3, 4 and 3 crowd members about them, respectively). We will explain later how such formulas are derived.

While this approach is shown to provide results superior to those obtained by asking the same overall number of questions only about the query attributes [27], a great obstacle is that it requires the use a *domain expert* that provides the list of related attributes for each query. This use of experts-in-the-loop limits the scalability of the approach and its applicability to fully-automated crowd-only platforms. In contrast, in the present paper, we provide a solution which is *entirely crowd-based.* Our goal is to replace the domain-expert by the Wisdom of Crowds, asking users to assist in "dismantling" difficult attributes and identifying those related attributes that can assist in query evaluation. Note that even harder related attributes may improve results they force different ways for estimations, which is one of the foundations of the wisdom of crowds principle.

As we will show, a successful solution will need to address two main challenges. The first is determining which object attributes should we best ask the crowd to dismantle. We show that it is sometimes beneficial to not only dismantle the query attributes themselves, but rather to continue dismantling newly discovered attributes. We provide a fine hypothetical analysis to estimate the potential benefit (and cost) of dismantling each of the so-far-discovered attributes and thereby determining which questions to ask the crowd. The second related challenge that we address is budget management. Given a budget (e.g., number of questions that can be asked to the crowd) for the offline preprocessing step, we need to use it both for dismantling the query attributes, as well as for obtaining some statistics about them (e.g., the distribution of user answers, the correlations between attributes value, etc.). Such statistics are required to estimate properties like "difficulty" and "contribution" of each attribute in order to decide how many crowd members should be asked about each attribute and how the answers should be assembled together. Our algorithm provides a careful analysis that allows to balance the budget between these two complimentary tasks.

We present in this article the following contributions:

1. We propose a simple and generic model for modeling a database of objects with infinite unknown attribute names and values, the type of questions that can be posed to the crowd and the characteristics of those answers.

2. Given an *online per-object budget* and an *offline preprocessing budget*, we use the model to present an algorithm that ideally uses the offline budget for deriving linear formulas (like the one illustrated above) that best exploit the online budget for deriving the values of query attributes. Our algorithm consists of five inter-related components for which we explain what type of information is required and what crowd questions may be used to obtain it. We provide a generic black-box description for each component (which allows to plug-in different implementations) and propose a concrete implementation.

3. Since the success of our framework depends on the discovery of relevant attributes, we focus our attention on this problem. We formally show how the potential gain (and cost) of each possible attribute dismantling question can be estimated and how this estimation can then be used to design an iterative algorithm that optimally chooses which crowd questions should be asked at each point. The estimation is based on a careful analysis of the already gathered information as well as on predictions about the potential effect of each question on the following algorithm components.

4. Of particular challenge are queries with more than one attribute. There is a fine tradeoff here between the gain that one may obtain by discovering underlying correlations between attributes, and the cost (in terms of crowd questions) required for such discovery. Our algorithm uses a fine analysis of the current data to predict potential contributions and to balance the two.

5. Finally, we present a thorough experimental analysis of our approach over two real-life data sets as well as synthetic data. We examine the various parts of our algorithm and its performance as a whole. We compare our algorithm to several existing/alternative approaches, showing that it consistently outperforms them in achieving lower average error for the same budget. Our experiments also demonstrate the necessity of different parts of the algorithm for accurate attribute dismantling, which later translates to accurate attribute value estimation.

The paper is organized as follows. The model and all relevant notations are presented in Section 2. To simplify the presentation we first consider in Section 3 the case where the query contains a single attribute. Queries with multiple attributes are then considered in Section 4. Our experimental study is presented in Section 5. We discuss related work in Section 6 and conclude in Section 7.

2. PRELIMINARIES

We start by describing our model and notations, then formally define the problem that we study.

Objects, attributes and queries. Our data set consists of a set of objects. We use $\mathcal{O}$ to denote the possibly infinite domain of objects, and o, o_i to denote an individual object in $\mathcal{O}$. In our running example, $\mathcal{O}$ is the set of all food recipes. An object may have attributes. We use $\mathcal{A}$ to denote the domain of attribute names and a,a_i to denote an individual attribute name in $\mathcal{A}$. We focus here on numerical attributes. Boolean attributes may be viewed here as numerical attributes with a value between 0 and 1, whereas multi-value attributes can be modeled by one such boolean attribute per value. In our running example, $\mathcal{A}$ includes all possible recipe properties such as *time_to_prepare*, *is_soup*, *is_tasty*, *protein_amount*, *is_brown*, *number_of_eggs*, etc.

For an object $o \in \mathcal{O}$, an attribute name $a \in \mathcal{A}$, a set of objects $O \subset \mathcal{O}$ and a set of attribute names $A \subset \mathcal{A}$, we use the following notations: (1) $o.a$ denotes the value of the attribute a of the object o, (2) $o.a^{(*)}$ denotes an estimation of that value (3) $o.A \equiv \{o.a \mid a \in A\}$ denotes the set of values for attributes in A of object o, (4) $O.a \equiv \{o.a \mid o \in O\}$ is the set of values of attribute a of the objects in O. We will sometimes abuse notations and consider these sets as *random variables* over objects in O (to be explained later). Finally (5) $D_{O\times A}$ denotes a data table with rows corresponding to objects in O and columns to attributes in A, along with some representation for each object.

Given a *query* Q over some data table D, we define $\mathcal{A}(Q)$ as the set of Q's query attributes. W.l.o.g one may think of Q as an SQL query and of $\mathcal{A}(Q)$ as the set of attribute names appearing in Q. In our running example Q might be, for instance, *select number_of_calories, protein_amount from CC where dessert=true.* In this example, $\mathcal{A}(Q) = \{$*is_dessert*, *number_of_calories*, *protein_amount*$\}$. As some attributes (and their values) may be missing from the data table D, we will need to learn them from the crowd.

Crowd questions. Crowd workers may be asked four types of questions:

Attribute Value Questions (for brevity, value questions) - Here a crowd member is asked to provide an estimation of the value of an *o.a*. An example of a value question in our running example is showing a worker a recipe and asking her for the value of *number_of_eggs*. For an object $o \in \mathcal{O}$ and an attribute $a \in \mathcal{A}$, $o.a^{(1)}$ denotes the random variable representing the estimation of one random worker's when asked about *o.a*. We use the $^{(1)}$ notation also for estimations of groups of values (for example, $O.A^{(1)} \equiv \{o.a^{(1)} \mid o \in O, a \in A\}$).

Attribute Dismantling Questions (for brevity, dismantling questions) - A crowd member is given here an attribute's name and requested to give another attribute's name that may provide some information about the value of the former. We assume (as later confirmed in our experiments) that workers are more likely to provide attributes that are correlative with the attribute in question. An example for a dismantling question may be *which recipe's attribute may help estimate its number_of_calories.* An answer may be *is_dietetic.* For simplicity we assume that answers that refer to the same property (like *large, big, grand*) can be reasonably identified and merged to a single representative. This may be done, e.g., using a common thesaurus/NLP tools. (We will show however that our technique can work even without this).

Dismantling Verification Questions (for brevity, verification questions) - Here we use crowd workers to verify that a previously suggested attribute a_i may indeed help in estimating the value of another attribute a_j. An example for a verification question is *does knowing if a dish is_black may help in determining its number_of_calories.* The likely crowd answer here is *No.*

Example Questions - Here workers are given some attributes' names and are asked to provide an example of an object $o \in \mathcal{O}$ along with its values for the attributes. An example for such a question is asking a user to upload a recipe along with its calorie value. For simplicity we will assume below that the given value is the correct one (otherwise the it can be estimated via value questions).

For all tasks we assume workers are independent and that spam filters are employed to avoid malicious workers.

Other notations. We further adopt and use some common notations. From statistics we use $\mathbb{E}_X[f(x)]$ for expectation, $\mathrm{Var}_X[f(x)]$ for variance, $\sigma_X(f(x))$ for standard deviation, $\mathrm{Cov}_{X,Y}(f(X), g(Y))$ for covariance and $\rho_{X,Y}(f(x), g(y))$ for correlation. The lower indexes specify the random variables and we omit them when they are clear from context. From algebra, we use M^T to denote a matrix M's transpose, M^{-1} to denote M's inverse and $\mathrm{Diag}(f(i))$ for diagonal matrix where the i'th value of the diagonal is equal to $f(i)$.

Problem definition. A user allocates a *per-object budget* B_{obj} which is the number of value questions that can be asked on a given object, in the online query evaluation phase, for estimating the value of the query attributes. To determine how to best use this budget, the user also allocates a *preprocessing budget* B_{prc} [2]. An (offline) preprocessing phase uses this to gather some information from the crowd (using the type of questions described above) and consequently derive a set of formulas of the form $o.a^{(*)} = \sum_{\mathcal{A}} l_a(a_i) o.a_i^{(b(a_i))}$ for each $a \in \mathcal{A}(Q)$, which determine how objects should be processed in the online query evaluation phase. The semantics of such formula is to first ask the crowd $b(a_i)$ value questions about each attribute $o.a_i$, then calculate the average answer for each $o.a_i$ (denoted $o.a_i^{(b(a_i))}$) and finally calculate an estimation for *o.a* using a *linear regression*[12] with predictors $l_a(a_i)$. An example of such a formula, for the attribute *protein_amount*, was given in the Introduction.

The function b in the formulas determines how many questions (if any) will be asked about each object's attribute, and intuitively reflects the "difficulty" of each attribute. Since the total value questions per object need to obey the B_{obj} budget constraint, b must satisfy $\sum_{a\in\mathcal{A}} b(a) \le B_{\mathrm{obj}}$. We call such function b a *budget distribution* of size B_{obj}.

For a given budget distribution function b and a linear regression formula l, we define an error in the estimation of a single attribute's value as $Er(o.a^{(*)} \mid b, l) = (o.a - \sum_{\mathcal{A}} l_a(a_i) o.a_i^{(b(a_i))})^2$. We then define the error of an attribute estimation as the mean square error over all objects $Er(\mathcal{O}.a^{(*)} \mid b, l) = \mathbb{E}_{\mathcal{O}}[Er(o.a^{(*)} \mid b, l)]$ and the query error as $Er(Q(D)^{(*)} \mid b, l) = \sum_{a\in\mathcal{A}(Q)} Er(\mathcal{O}.a^{(*)} \mid b, l)$. Note that for simplification we assume the errors of all attributes to be of equal importance. All our results also apply to a weighted error definition, as discussed later. Our goal here will be to minimize the query error $Er(Q(D)^{(*)})$. Namely, to find b and l which minimize it, and to do this using at most a budget B_{prc}.

3. OUR SOLUTION

We start by presenting a high-level informal description of our algorithm, what data do we collect and what is that data used for. Next, we provide a detailed formal description of the functionality of the different components, as well as references to existing solutions for some of them. We then return to the components that are in the heart of our contribution and provide concrete novel solutions for them.

To simplify the presentation we will assume below that the query contains only a single attribute, that we call the

[2]W.l.o.g. it is assumed that $B_{\mathrm{prc}} >> B_{\mathrm{obj}}$

target attribute, namely $\mathcal{A}(Q) = \{a_t\}$. We consider the general case afterwards. We also assume that no attributes are initially available for the objects in the queried data table. For instance, in our running example this means that we are only given the recipes and no explicit attributes for them. The algorithm can be naturally extended to the general setting.

```
   Data: $Q, B_{\text{obj}}, B_{\text{prc}}$
1  $E_B \leftarrow$GetExamples$(N_1, k)$;
2  while $CollectingAttributesCondition = True$ do
3  |   $a \leftarrow$GetNextAttribute$(A, S, B_{\text{obj}})$;
4  |   $A \leftarrow A \cup a$;
5  |   $S \leftarrow$UpdateStatistics$(S, a, E_B)$;
6  $b \leftarrow$FindBudgetDistribution$(S)$;
7  $E_L \leftarrow$GetExamples$(N_2, b)$;
8  $l \leftarrow$FindRegression$(b, E_L)$;
9  return $l, b$
```

Algorithm 1: The base case solution

Algorithm 1 depicts a general description of our solution. Table 1 shows the different information items being collected throughout its execution. It contains objects (first column), true values of objects' attributes (second column), and sets of workers' answers to value questions (denoted $\{o_i.a_j^{(1)}\}_1^n$ in all other columns where n is the answers set size). We use this table to illustrate *what* is done by Algorithm 1. We later explain *how* exactly this is done (what crowd tasks are involved, etc.). Note that some notations in Table 1 may not yet be clear at this point, but will be explained later.

At the beginning, the only information available is the name of the query attribute (Table 1's "True Values for $\mathcal{A}(Q)$" header). We first collect a set of example objects $E_B = \{e_1, \ldots, e_{N_1}\} \in \mathcal{O}$ along with their true value for a_t. Those objects and values are shown in Table 1a. We then iteratively add new attribute columns in Table 1a by dismantling existing attributes, thereby discovering new attribute names (the "Value Questions Answers for A_{final}" headers) and then obtaining values for them from workers. During this iterative process, we also use the crowd answers to calculate some statistics (not shown in the table) on the discovered attributes, which we use for deciding which attributes need to be dismantled next. When this collection process ends (we will explain later how this is determined), we use the statistics again to calculate a budget distribution b. Finally, to compute the linear regression l we collect a second set of examples E_L, along with their value for a_t (Table 1b's Objects and True Value columns). We use b to collect crowd answers for the remaining attributes, then use all the gathered information to learn the linear regression l. Now that we derived both b and l, the preprocessing phase ends.

Later, in the query evaluation phase, b and l are used to collect estimations about the objects in the queried data table D (the Answers in 1c) and use it to calculate and return $Q(D)$ (the "True Values" column in 1c).

3.1 The Algorithm Components

Algorithm 1 consists of five logical components: finding relevant attributes (lines 3-4), collecting statistics about them (lines 1 and 5), calculating a budget distribution (line 6), learning a linear regression (lines 7-8) and managing the preprocessing budget (line 2). We discuss them next.

objects	True Values for $\mathcal{A}(Q)$	Value Questions Answers for A_{final}			
	a_1	a_1	a_2	$\cdots$	a_l
e_1	$e_1.a_1$	$\{e_1.a_1^{(1)}\}_1^k$	$\{e_1.a_2^{(1)}\}_1^k$	$\ldots$	$\{e_1.a_l^{(1)}\}_1^k$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
e_{N_1}	$e_{N_1}.a_1$	$\{e_{N_1}.a_1^{(1)}\}_1^k$	$\{e_{N_1}.a_2^{(1)}\}_1^k$	$\ldots$	$\{e_{N_1}.a_l^{(1)}\}_1^k$

(a) Data used to calculate b

objects	True Values for $\mathcal{A}(Q)$	Value Questions Answers for A_{final}			
	a_1	a_1	a_2	$\cdots$	a_l
e_1	$e_1.a_1$	$\{e_1.a_1^{(1)}\}_1^{b(a_1)}$	$\{e_1.a_2^{(1)}\}_1^{b(a_2)}$	$\ldots$	$\{e_1.a_l^{(1)}\}_1^{b(a_l)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
e_{N_2}	$e_{N_2}.a_1$	$\{e_{N_2}.a_1^{(1)}\}_1^{b(a_1)}$	$\{e_{N_2}.a_2^{(1)}\}_1^{b(a_2)}$	$\ldots$	$\{e_{N_2}.a_l^{(1)}\}_1^{b(a_l)}$

(b) Data used to learn l

objects	True Values for $\mathcal{A}(Q)$	Value Questions Answers for A_{final}			
	a_1	a_1	a_2	$\cdots$	a_l
o_1	?	$\{o_1.a_1^{(1)}\}_1^{b(a_1)}$	$\{o_1.a_2^{(1)}\}_1^{b(a_2)}$	$\ldots$	$\{o_1.a_l^{(1)}\}_1^{b(a_l)}$
o_2	?	$\{o_2.a_1^{(1)}\}_1^{b(a_1)}$	$\{o_2.a_2^{(1)}\}_1^{b(a_2)}$	$\ldots$	$\{o_2.a_l^{(1)}\}_1^{b(a_l)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

(c) Data used in the online phase

Table 1: Data collected during the algorithm

Finding Attributes. We identify here a set of attributes that may assist in estimating the value of the query attribute. This is done using dismantling questions, followed by corresponding verification questions. A key observation here is that it is sometimes beneficial to not only ask users to dismantle the query attribute itself, but rather to continue dismantling newly discovered attributes. Indeed, since the human mind is associative, asking diverse questions is important for better learning a domain [7]. For example, in our CC example, when asking a user to dismantle *protein_amount*, we may get the attribute *meat_content* as an answer, but the distinction between *red_meat* and *white_meat* (which have different protein amounts) may be only obtained when asking users to dismantle *meat_content*.

We denote by A_m the set of known related attributes after m iterations (and respectively $A_0 = \mathcal{A}(Q)$ and A_{final} is the final subset). We denote by $A_{m+1|a_j} = A_m \cup ans_j$ the random variable representing this set, assuming the next dismantling question is for attribute a_j. Our goal will be to choose a_j such that $A_{m|a_j}$ allows minimal error. More formally, we wish to find

$$\operatorname{argmax} a_j \mathbb{E}_{A_{m|a_j}=A}[\min_{\substack{l,b \\ \forall a \notin A \ b(a)=0}} Er(Q|b,l)] \qquad (1)$$

As this choice obviously depends on how the budget distribution b and the linear regression l are selected, we leave the solution of expression 1 for section 3.2.1. For now we only note that after asking the selected dismantling question and getting a new attribute name for an answer, we use verification questions to ensure that the obtained new attribute name is indeed a relevant one. Here we use standard algorithms such as [25] to determine the required number of questions for making a decision.

Since a dismantling question in our setting is always followed by corresponding verifications questions, from here on whenever we use the term dismantling question we also refer to its following verification questions.

Collecting Statistics. As mentioned, we need to collect some information about A_m - the set of known related attributes after m iterations - and the way workers answer

			$\mathcal{A}(Q)$	A_m			
			a_1	a_1	a_2	$\cdots$	a_l
A_m	a_1	$S_c[1]$	$S_o[1]$	$S_a[1,1]$	$S_a[1,2]$	$\cdots$	$S_a[1,l]$
	a_2	$S_c[2]$	$S_o[2]$	$S_a[2,1]$	$S_a[2,2]$	$\cdots$	$S_a[2,l]$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	a_l	$S_c[l]$	$S_o[l]$	$S_a[l,1]$	$S_a[l,2]$	$\cdots$	$S_a[l,l]$

Table 2: Statistics calculated during the algorithm

them. Our main tool for finding those statistics is gathering samples of the crowd responses and analyzing them. We do so by asking value questions about a set of example objects collected using example questions. Formally, our goal is to find an accurate estimation for the trio $S_A = (S_{oA}, S_{aA}, S_{cA})$ (or just S when the index is clear from context), depicted in Table 2 and defined as follow (for reasons that will be clear later).

S_c - Statistics about agreement among crowd workers. More precisely, a vector of the average variances of workers answers to value questions. Formally, S_{cA} is a vector of size $|A|$ where $S_{cA}[a] = \mathbb{E}_{\mathcal{O}}[\text{Var}[o.a^{(1)}]])$. For instance, in our example we can expect $S_c[\textit{healthy}] > S_c[\textit{tomato}]$ as it is easier to identify if a recipe contains a tomato. This is the second column of Table 2.

S_o - Statistics about how informative are the attributes. More precisely, this is the covariance vector between workers answers to the different attribute and the query attribute. Formally, S_{oA} is a vector of size $|A|$ and $S_{oA}[a] = |\text{Cov}_{\mathcal{O}}(o.a^{(1)}, o.a_t)|$. For example, if $a_t = \textit{dessert}$, we can expect $\frac{S_o[sweet]}{\sigma(sweet)} > \frac{S_c[cheese]}{\sigma(cheese)}$ as most desserts are sweet (and most non-desserts are not) but cheese can be easily found both in desserts and in non-desserts. This is the third column of Table 2.

S_a - Statistics about how much distinctive are attributes (in comparison to the other attributes). More precisely, this is the covariance matrix over crowd's answers to different attributes. Formally, S_{aA} is a matrix of size $|A| \times |A|$ and $S_{aA}[a_i, a_j] = |\text{Cov}_{\mathcal{O}}(o.a_i^{(1)}, o.a_j^{(1)})|$. For example, we can expect $\frac{S_a[\text{Spicy, Sugar}]}{\sigma(\text{Spicy})\sigma(\text{Sugar})} > \frac{S_a[\text{Easy to make,Sugar}]}{\sigma(\text{Easy to make})\sigma(\text{Sugar})}$ as sugar usually indicates a non-spicy food but does not imply anything about the complexity of the recipe. This is the fourth column of Table 2.

Our goal is to get relatively good estimations of these measures for a low budget. We describe how this is done in section 3.2.2.

Calculating a budget distribution . Once the relevant attributes are identified and the relevant statistics are calculated we run a strictly computational algorithm to find b. As it was shown in [27], when applying the best linear regression to some table $D^{(b)}_{O\times A}$ (a notation that means $D^{(b)}_{O\times A}[o,a] = o.a^{(b(a))}$), the error is $\mathbb{E}[E.a_t]^2 - {S_o}^T(S_a + Diag(\frac{S_c(a)}{b(a)}))^{-1}S_o$. The first element is independent of b, so we have that the best budget distribution is

$$\underset{b}{\text{argmax}}\, {S_o}^T(S_a + Diag(\frac{S_c(a)}{b(a)}))^{-1}S_o \tag{2}$$

[27] also showed that finding this optimal b is Np-hard in B_{obj} and therefore an approximating algorithm is appropriate. They provide such algorithm, which is a variation of the well known greedy forward selection. We use this algorithm as the *FindQuestionsDistribution* method in Algorithm 1.

Learning a Linear Regression. The last part of the algorithm is deciding on a linear regression l. Since we can not find the overall best linear regression, we minimize the error over some training set representing the online phase data (meaning that the estimation of each attribute a is done according to $b(a)$). We get this set by using example questions (getting object and target values) and value questions (getting attribute values). To reduce costs we re-use previously collected data. When collecting objects we skip the first N_1 example questions, and when collecting estimations we only ask $b(a) - k$ value questions for each $e.a.$ This is line 7 in Algorithm 1 and how we collect Table 1b's data.

Once such training set exists, further computations are applied to find a linear regression l that minimizes the error over it. The problem of finding a linear regression that minimizes the mean square error is a well studied problem [12] and there are many algorithms for it that we can just use. Specifically, we used a singular value decomposition (SVD [15]) algorithm, but since it is used as a black box other algorithms can also fit. This is line 8 in Algorithm 1.

Managing the Preprocessing Budget. To fully understand where and how budget is spent, one needs to first see the actual implementation presented next. We thus postpone this discussion to section 3.2.3.

3.2 Concrete Solutions

Finally, we can focus on our implementations. We wish to remind that although they are described separately, all the components are in fact intertwined.

3.2.1 Finding Attributes

Recall that our objective is to solve expression 1. Knowing now, that the error behaves like expression 2 we can state a more specific objective - finding

$$\underset{a_j}{\text{argmax}}\,\underset{b}{\text{argmax}}\, S_{oA_{m|a_j}}^T (S_{aA_{m|a_j}} + Diag(\frac{S_{cA_{m|a_j}}(a)}{b(a)}))^{-1} S_{oA_{m|a_j}} \tag{3}$$

As an exact solution can only be made after asking all questions and calculating all $S_{A_{m|a_j}}$, we use the current statistics $S_{A_{m-1}}$ and estimate the next statistics $S_{A_{m|a_j}}$, for every a_j. We also calculate the probability of it remaining the same as only a first seen answer will effet it. We then use those estimations and solve expression 3. Described here are general schemes of the estimations. Full calculations can be found in the our paper[22].

Pr(new | a_j) - We need to estimate the probability to get a new answer. We do so by assuming it depends only on the number of questions asked so far and then using a simple Bernoulli-Bayesian model with the number of questions asked about a_j so far (n_j). The results are

$$\Pr(\text{new} \mid a_j) = \frac{n_j + 1}{n_j^2 + 3n_j + 2} \tag{4}$$

$\boldsymbol{S_{oA_{m|a_j}}}$ - We need to estimate the covariance of the next answer and the target ($S_o[ans_j]$). By definition of correlation we have $S_o[ans_j] = \frac{\rho(a_t, ans_j)}{\rho(a_t, a_j)} \frac{\sigma(ans_j)}{\sigma(a_j)} S_o[a_j]$. $\sigma(a_j)$ and $S_o[a_j]$ are known. $\sigma(ans_j)$ is assumed to be independent with a_j and can therefore be ignored. That leaves only the correlations' ratio. We previously assumed ans_j is highly correlated to a_j, we now approximate this as $\mathbb{E}[\rho(a_j, ans_j)] \approx 0.5$, which translates to $\frac{\rho(a_t, ans_j)}{\rho(a_t, a_j)} \approx 0.5$. We address this approximation later. This results in

$$S_{oA_{m|a_j}}[a] \approx \begin{cases} \frac{0.5}{\sigma(a_j)} S_{oA_{m-1}}[a_j] & a = ans_j \\ S_{oA_{m-1}}[a] & \text{otherwise} \end{cases} \tag{5}$$

$\boldsymbol{S_{cA_{m|a_j}}}$ - We need to estimate the variance of the next answer ($S_c[ans_j]$). Since there is no reason for it to change over different dismantling questions we can use the same distribution for every j. Because *FindRegression* is not analytic (as we will see later), instead of measuring an exact distribution (which will make it impossible to calculate) we take an 'optimism in the face of uncertainty' approach [20] and assume a very low constant value for it ($\forall j\ S_c(ans_j) \approx 0$). We then get

$$S_{cA_{m|a_j}}[a] \approx \begin{cases} 0 & a = ans_j \\ S_{cA_{m-1}}[a] & \text{otherwise} \end{cases} \tag{6}$$

$\boldsymbol{S_{aA_{m|a_j}}}$ - We need to estimate the covariances between the new answer and the previously discovered attributes ($S_a[a_i, ans_j],\ a_i \in A_{m-1}$). Again, since there is no reason for this to change over different dismantling questions we can just take the same distribution for every j. For similar reasons (calculations practicality), we again take the 'optimism in the face of uncertainty' approach. We (wrongfully) assume no correlation between the new and the existing attributes. This assumption cannot be taken for $S_a[ans_j, ans_j]$, but this factor cancelled anyway in the $Er(Q)$ calculation. We then get

$$S_{aA_{m|a_j}}[a_u, a_v] \approx \begin{cases} 1 & a_u = a_v = ans_j \\ 0 & ans_j \in \{a_u, a_v\} \\ S_{aA_{m-1}}[a_u, a_v] & \text{otherwise} \end{cases} \tag{7}$$

By putting results 4, 5, 6 and 7 into expression 3 we get that the best next dismantling question is

$$\underset{a_j}{\operatorname{argmax}} \Pr(\text{new} \mid a_j)[G(a_j) - L(A_{m-1}, B_{\text{obj}}, 1)] \tag{8}$$

where $\Pr(\text{new} \mid a_j)$ was described before, $G(a_j) = \frac{0.25 S_o[a_j]^2}{\sigma(a_j)^2}$ and $L(A, u, v) = \max_{b \text{ of size } u} S_{oA}(S_{aA} + Diag(\frac{S_{cA}}{b}))^{-1} S_{oA} - \max_{b \text{ of size } u-v} S_{oA}(S_{aA} + Diag(\frac{S_{cA}}{b}))^{-1} S_{oA}$.
Intuitively, when adding a new attribute, some of the record budget moves from the old attributes to the new one. G measures the gain from the new attribute and L measures the loss caused by decreasing budget from the old attributes. The reasons for those being the only changes are the low correlation assumptions we took while estimating $S_{A_{m|a_j}}$.

As all of those values can be calculated, this concludes the *GetNextAttribute* method in Algorithm 1. This is also how we get each "Value Questions Answers" header in Table 1. It is easy to see that one dismantling question at the end of each iteration is the only crowd task.

3.2.2 Collecting Statistics

As explained, our goal in this part is to find a good approximation for (S_{oA}, S_{aA}, S_{cA}) while using a minimal budget. Following our iterative process for finding attributes, we build S in an inductive way. Namely, for each new attribute we calculate S_{A_m} based on $S_{A_{m-1}}$ and questions about the new attribute.

For our current simplified case, there exists an approximation method in [27] that have proven itself before and that we can easily adapt. When we later discuss the general case, we will return to this part and refine the calculation. The ideas we take from [27] are to estimate $S_{A_{\text{final}}}$ (which is defined over $\mathcal{O}$) by calculating it over example set E_B and then, for each object, estimating the behavior of $o.a^{(1)}$ based on k sample answers (for a very small k). We use those ideas in an inductive way

A_{-1} - We leave S_c,S_a,S_o empty but collect a set of examples E_B with target a_t value by asking N_1 example questions (N_1 is a parameter studied in [27]). This is line 1 in Algorithm 1 and during it we collect Table 1a's objects and true values.

A_m - For the new attribute a we ask k values questions about $e.a$ for every $e \in E_B$. We then update S by keeping all previous values and adding (1) $S_o[a] = \mathbb{E}_{E_B}[e.a^{(k)} \cdot e.a_t]$, (2) $S_a[a, a_i] = S_a[a_i, a] = \mathbb{E}_{E_B}[e.a^{(k)} \cdot e.a_i^{(k)}]$ for every $a_i \in A_m$ and (3) $S_c[a] = \mathbb{E}_{E_B}[VarEst_k(e.a^{(1)})]$. This is the *UpdateStatistics* method in algorithm 1. During each step we get a sub-column in the answers column of Table 1a, a row in Table 2 and a sub-column in the right column of Table 2.

It should be easy to see how this algorithm is compliant with the ideas mentioned above. It should also be easy to see that during this part we use the crowd for N_1 example questions and $kN_1|A_{\text{final}}|$ value questions.

3.2.3 Management of the Preprocessing Budget

In our algorithm the preprocessing budget is used for

- Finding A_{final} by asking attributes and attributes verification questions. This costs n dismantle questions where n is the number of dismantling questions we choose to ask.
- Calculating the statistics $S_{A_{\text{final}}}$ by asking example and value questions. This costs N_1 example questions and $kN_1|A_{\text{final}}|$ value questions, where $|A_{\text{final}}|$ depends on n.
- Collecting a training set of N_2 samples for l's learning. This costs $(N_2 - N_1)$ example questions and $(N_2 - N_1)B_{\text{obj}} + N_1(B_{\text{obj}} - \sum_{\mathcal{A}} \min\{b(a), k\})$ value questions.

As N_1 and k are external parameters, the only variables are n and N_2. Therefore, the only open question is when to stop asking dismantling questions (line 2 in algorithm 1). We solve this tradeoff by applying a common simple linear lower bound for N_2 as a function of b ([16]). Note that as our only tradeoff is n vs. N_2, this mechanism is also appropriate when considering different costs for different crowd tasks.
As to the case of different cost that may apply for different questions of the same type (for example, numeric vs. binary), the appropriate coefficients need to be added. In this case, we also follow [27] idea and, during all components, divide each attribute's contribution by its cost.

Remarks. We conclude this section by commenting on the correctness and complexity of the algorithm. First, it is easy to see that the algorithm operates within the preprocessing budget B_{prc}. Second, it is also easy to see that the algorithm running time is polynomial with respect to the two budgets B_{prc} and B_{obj}.

4. EXTENDED SOLUTION

We focused so far on the simplified case where the query has a single attribute. We next consider the general case of multiple query attributes.

A naive solution is to equally split the online and offline budgets between the query attribute and solve the problem for each one separately. This however ignores possible correlation between the query attributes and their components. For example, consider a query with two attributes $\mathcal{A}(Q) = \{\textit{calories, is\_dessert}\}$. It is easy to see that many related attributes (e.g., *sugar, fat,...*) are good indicators for both target attributes, and budget would be saved if we reuse values. To address this we consider all the query attributes together, extending Algorithm 1 to handle multiple target attributes. We first present below a simple extension, then discuss its shortcomings, and then generalize it to overcome them. Our first extension generalized the algorithm components as follows.

GetExamples - Instead of asking the crowd for examples with one value for the single query attribute we now ask for examples with multiple attribute values - one per query attribute. (We later discuss what to do if users cannot provide all these values simultaneously.)

GetNextAttribute - Expression 8 is refined to consider all the attributes:

$$\underset{a_j}{\operatorname{argmax}} \sum_{a_t} \Pr(\text{new} \mid a_j)[G(a_t, a_j) - L(a_t, A_{m-1}, B_{\text{obj}}, 1)] \tag{9}$$

UpdateStatistic - Instead of updating the S_o statistics of a_{new} for a single query attribute, we now need to update $S_{oa_t}[a_{\text{new}}]$ for every query attribute $a_t \in \mathcal{A}(Q)$. Note that we added here the a_t notation to S_o since there are now several query attributes. S_a and S_c remained the same since they are independent of Q.

FindQuestionsDistribution - Instead of equation 2 we now have a refined version

$$\underset{b}{\operatorname{argmax}} \sum_{a_t \in \mathcal{A}(Q)} S_{o(a_t,A)}^{T} (S_{aA} + Diag(\frac{S_c[a]}{b(a)})^{-1} S_{o(a_t,A)} \tag{10}$$

FindRegression - Since l is now a set of linear regressions, we need to run this method $|\mathcal{A}(Q)|$ times. Since all l_{a_t} are independent this yields an optimal solution.

Note that for *GetExamples* we assumed above that it is possible to ask workers for examples with several attributes values. This may be problematic in practice: If the number of query attributes is too large, workers may not be willing to make the effort of providing all of their values; It may also be the case that a single crowd member does not know the value of all attributes, even for their own examples. To overcome this, instead of using just one set of examples E_B with all query attributes, we will collect multiple sets of examples $E_{B_{a_t}}$, one for each query attribute (or a small subset thereof). In this case, the collected data in Table 1a is replaced by the one depicted in Table 3.

Looking at this table we can see that although the information can be used to derive b, it comes with an additional cost: It is easy to see that the amount of data

	True Values for $\mathcal{A}(Q)$		Value Questions Answers for A_{final}		
	a_1	a_2	a_1	$\cdots$	a_l
e_1	$e_1.a_1$	?	$\{e_1.a_1^{(1)}\}_1^k$	$\dots$	$\{e_1.a_l^{(1)}\}_1^k$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
e_{N_1}	$e_{N_1}.a_1$	?	$\{e_{N_1}.a_1^{(1)}\}_1^k$	$\dots$	$\{e_{N_1}.a_l^{(1)}\}_1^k$
e_{N_1+1}	?	$e_{N_1+1}.a_2$	$\{e_{N_1+1}.a_1^{(1)}\}_1^k$	$\dots$	$\{e_{N_1+1}.a_l^{(1)}\}_1^k$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
e_{2N_1}	?	$e_{N_1+1}.a_2$	$\{e_{2N_1}.a_1^{(1)}\}_1^k$	$\dots$	$\{e_{2N_1}.a_l^{(1)}\}_1^k$

Table 3: Data collected in the general case

we need to collect now depends both on $|A_{\text{final}}|$ and on $|\mathcal{A}(Q)|$. Therefore, if we want to allow A_{final} to grow with $\mathcal{A}(Q)$, our cost will grow quadratically. It is also easy to see that most of this growth is due to cost of redundant value questions. For example, consider $\mathcal{A}(Q) = \{\textit{is\_dessert, number\_of\_calories,protein\_amount, easy\_to\_make}\}$. One can assume that although there is likely to be a correlation between *number_of_calories* and *is_dessert* this is not the case for *easy_to_make* and *protein_amount* so collecting statistics for all pairs is a waste of budget. To reduce this redundant overhead we take two steps. First, we choose carefully which data to collect (i.e., which $E_{B_{a_t}}.a$ value questions should we asked). Second, for pairs for which data had not been collected, we estimate $S_{oa_t}[a]$ based on the other collected data. We explain this next.

Collection. Our choice of which data to collect is based on the following observation. The two cases one wants to avoid are (1) missing a highly correlated attribute-target pair and (2) wasting budget on a poorly correlated attribute-target pair. Therefore, whenever we get a new attribute a_j we pair it with all query attributes a_t for which we have a reason to believe that $S_{oa_t}[a_j]$ is not negligible. In our heuristic, we define $S_{oa_t}[a_j]$ as negligible iff its value is less than a half of the maximal value $\max_{a \in \mathcal{A}(Q)} S_{oa}[a_j]$. Our estimation here for $S_{oa_t}[a_j]$ is done in the same way we described earlier in section 3.2.1. This results in the following rule - when asking a dismantling question about a_i and getting an answer a_j, ask value questions about $E_{B_{a_t}}.a_j$ iff $\rho(a_i, a_t) > 0.5 \max_{a \in \mathcal{A}(Q)} \rho(a_j, a)$.

Estimation. Finally, to estimate the missing S_o's values we use a graph model and define $G = (U, V, E)$ as a weighted bipartite graph with $U(G) = \mathcal{A}(Q)$ and $V(G) = A_m$. The idea is to make each edge's weight $w(a, a_t)$ represent the value of $S_{oa_t}[a]$ and then estimating missing edges by distances on a graph. Ideally, we would have defined $w(a, a_t) = S_{oa_t}[a]$. However, since S_o is not normalized and also not a distance function, this is impossible. To overcome this, we employ a method described in [29] and use angular distance as our weight function - $w(a_t, a_j) = \Gamma(O.a_t, O.a_j) = \arccos \frac{S_{oa_t}[a_j]}{\sigma(a_t)\sigma(a_j)}$. The idea behind angular distance is to consider an inner-product space where the vectors are random variables and the inner-product is covariance. This allows to prove that Γ is indeed a distance function that answer what we looked for. Using the fact that in the angular distance space $\Gamma_1 + \Gamma_2 = \arccos(\cos(\Gamma_1)\cos(\Gamma_2))$, we define our estimation of S_o as

$$S_{oa_t}[a_j] = \sigma(a_t) \cdot \sigma(a_j) \cdot \begin{cases} \cos(w(a_j, a_i)) & \text{edge exists} \\ \cos(\text{S.P}(a_t, a_j)) & \text{path exists} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where S.P stands for (multiplication) shortest path.

Weighted query attributes. To conclude, note that in our discussion so far we assumed the errors of all attributes to be of equal weight. In practice some normalization may be required. For example, *is_healthy* is on a scale of $[0, 1]$ whereas *number_of_calories* may reach thousands. In this case, each $a_t \in \mathcal{A}(Q)$ is associated with a weight ω_t and our goal is to minimize $\sum_{a_t \in \mathcal{A}(Q)} \omega_t \mathbb{E}[(O.a_t^{(*)} - O.a_t)^2]$. When following the previous calculation this simply results in adding weights to expression 9.

5. EXPERIMENTS

We analyze our solution experimentally along through dimensions. We start with a general proof of concept - an examination of our algorithm as a whole. We then move to an analysis of its components, their necessity and their quality. We conclude with an analysis of how different assumptions and parameters can influence the results.

5.1 Experiments Settings and Datasets

We used three datasets - two with real life objects and real crowd answers, and one synthetic. Crowd answers from value and attribute questions were gathered through CrowdFlower[3] - a platform for presenting small tasks to crowds. The answers collected in initial experiments was recorded in a database and reused in following experiments, so that results of multiple runs/algorithms may be compared in equivalent settings. To compare our performance to [27] that used experts to obtain relevant attributes, we also added to our database the data collected in that work. For example questions, in order to have a true 'gold standard' (known target answers), we used our lab members as crowd.

We designed our crowd interface and payment following the guidelines in [13] and the work of [27]. Our crowd tasks consist of a set of value (resp. dismantling) questions that a crowd member needs to answer. We set the payment for binary value question to 0.1¢ and to 0.4¢ for general numeric values. For dismantling and example questions, that were not studied in [27], we set the payment to 1.5¢ per answer, following our preliminary experiments that showed this to be the minimal price that kept workers' feedback positive, and set the price of an example question to 5¢ as this is a relatively hard task. (We will show however in the sequel that the trends in our results are robust to changes in these numbers). As for other parameters we used the following: the number of value samples k used for estimating statistics when deriving budget distributions was 2, as this is the recommended number for the corresponding black-box that we used[27]. The number of examples N_1 was set to 200 to keep our costs low while still having many examples. For learning the linear regression, the examples number N_2 was set to $50 + 8 * \#$attributes, a common practice in such tasks [16]. For attribute weights, unless otherwise stated, we gave each query attribute a weight in reverse proportion to its variance ($\omega_t = \frac{1}{Var(O.a_t)}$). This normalize all errors to a similar scale (standard deviations), so that no query attribute will be negligible. We will explicitly mention below where using other weights affects the results.

Human Pictures Data Set. In this set of experiments our objects are people and the only information available for them is their picture. The query attributes in the different experiments include *Weight, Height, Age, Bmi* (body mass index, defined as $\frac{\text{weight(kg)}}{\text{height(m)}^2}$) and *Attractiveness.* The objects O were taken from the publicly available Photographic Height/Weight Chart [4], where people post pictures of themselves announcing their own height and weight. We used reported value as the true values for *Height, Weight* and *Bmi.* For other target values, we used an average over many value question estimations.

Examples of answers received when asking workers to dismantle various attributes are depicted in Table 4. The first column depicts the attribute to dismantle, the second column contains some related attributes suggested by the crowd, the last column shows the percentage of all answers that each attribute name was returned. Examples of statistics gathered for the attributes are depicted in Table 5 (this is a concrete example of Table 2, but unlike Table 2 it shows attributes correlation and not covariance to make things more intuitive for the reader).

Recipes Data Set. In this set of experiments our objects are recipes and the data available for them in the database is the recipe's name, picture and unstructured ingredients-list. The query attributes in the different experiments include *Proteins, Calories, Good_for_kids, Easy_to_make* and *Healthy.* The objects are the 500 most popular recipes in `allrecipes.com`[1] website, normalized to one serving. We used nutritious values found in this website as true values for the matching query attributes. For other query attributes we again used average value derived from multiple value questions. Here again, examples of some answers obtained for dismantling questions and statistics on the attributes are depicted in tables 4 and 5 respectively.

Question	Answer	Frequency
Bmi	Weight	33%
	Height	33%
	Age	6%
	Attrctive	2%
Height	Age	22%
	Shoe Size	9%
	Taller Then You	7%
	Weight	6%
Age	Wrinkles	15%
	Gray Hair	10%
	Old	10%
	Children	3%
Attractive	Good Facial Features	17%
	Fat	6%
	Has Good Style	6%
	Works Out	1%

(a) Pictures Domain

Question	Answer	Frequency
Calories	Has Eggs	8%
	Low Calories	4%
	Dessert	2%
	Healthy	2%
Protein	Has Meat	13%
	Number of Eggs	4%
	High Protein	4%
	Vegetarian	2%
Healthy	Low Salt	8%
	Natural	8%
	Fat Amount	4%
	Bitter	4%
Easy To Make	Number of Ingredients	17%
	Fast	10%
	Tasty	5%
	Expensive	2%

(b) Recipes Domain

Table 4: Attribute dismantling questions and their answers

	S_c	$S_o\sigma(a_i)\sigma(a_j)$		$S_a\sigma(a_i)\sigma(a_j)$					
		Bmi	Age	Bmi	Weight	Heavy	Attractive	Works Out	Wrinkles
Bmi	30	0.88	0.63	1	0.94	0.86	0.48	0.4	0.26
Weight	189	0.86	0.7	0.94	1	0.82	0.53	0.39	0.28
Heavy	0.14	0.89	0.6	0.86	0.82	1	0.44	0.46	0.27
Attractive	0.13	0.45	0.44	0.48	0.53	0.44	1	0.32	0.28
Works Out	0.11	0.36	0.29	0.4	0.39	0.46	0.32	1	0.15
Wrinkles	0.16	0.25	0.52	0.26	0.28	0.27	0.28	0.15	1

(a) Pictures Domain

	S_c	$S_o\sigma(a_i)\sigma(a_j)$		$S_a\sigma(a_i)\sigma(a_j)$					
		Calories	Protein	Calories	Low Calorie	Desset	Healty	Vegetarian	Eggs
Calories	80707	0.41	0.34	1	0.2	0.07	0.15	0.18	0.03
Low Calorie	0.06	0.18	0.08	0.2	1	0.1	0.26	0.1	0.13
Desset	0.08	0.26	0.5	0.07	0.1	1	0.44	0.34	0.38
Healthy	0.2	0.02	0.16	0.15	0.26	0.44	1	0.06	0.27
Vegetarian	0.13	0.26	0.52	0.18	0.1	0.34	0.06	1	0.14
Eggs	0.05	0.11	0.26	0.03	0.13	0.38	0.27	0.14	1

(b) Recipes Domain

Table 5: Examples for statistics in the different domains

Synthetic Data. To neutralize our own subjectivity/belief w.r.t which object attributes are hard/easy, we also ran experiments on a synthetically generated domain. For this we automatically generated a set of objects and attributes (with some dependencies between them) and mocked crowd answers about them (in compliance with the assumptions on crowd's answers mentioned in the paper). The details of this process can be found in the full paper [22]. The experiment results are consistent with those for real-life data and are thus omitted here.

5.2 Proof of concept

We compared our algorithm, which we call *DisQ* (short for Dismantling queries), to existing practices. We use the following algorithms as baselines:

NaiveAverage - In this common approach, the online phase simply asks questions about the attributes in $\mathcal{A}(Q)$ and returns their average $o.a_t^{(B_{\text{obj}})}$. For $|\mathcal{A}(Q)| > 1$ we split the budget by the weights. This algorithm has no offline preprocessing phase.

SimpleDisQ - This is a simplified version of our algorithm, which captures the best that can be done today without using an expert. It runs similar to *DisQ*, but without the attribute dismantling phase.

We compared these two algorithms to our algorithm. We did so for all three data sets and for different query attributes and query sizes. We also tested with different preprocessing budgets B_{prc} and different per-object budgets B_{obj}. For B_{obj} we used the range of 0.4-10¢. The lower bound was set to match 1 numeric value-question. The upper bound was set as it is a fairly large amount and as most of the experiment graphs show stagnation after it. For B_{prc} we used the range of \$10-35. We have taken those values since the graphs stagnate outside those boundaries. For each value we executed 30 experiments and took the average result. Note that although we took the average, all observations are true in general as most results are very close to the average.

Varying B_{prc}. Figure 1a shows results for a query with $\mathcal{A}(Q) = \{Bmi\}$ (using the pictures data set) for varying preprocessing budgets. We will show more results later. We start with an example where $|\mathcal{A}(Q)| = 1$ to isolate different effects. We fixed B_{obj} to 4¢ and used different B_{obj} values. We used $B_{\text{obj}} = 4$¢ as it is over the graph's knee (as we will see later). Note that since *NaiveAverage* does not involve learning and since the number of examples in *SimpleDisQ* is always N_1 (since A_{final} is very small), *DisQ* is the only algorithm that changes with B_{prc}. One can easily see that for every B_{prc} value our algorithm has the lowest average error. The difference is especially significant for large B_{prc} values as for those ranges A_{final} is bigger. We can also see that the improvement is slowly stagnating which is the expected result if the "important" attributes are found quickly.
To ilustrate how an algorithm's output looks like, we provide here an example for one of the dismantles when $B_{\text{prc}} = \$25$. $Bmi^{(*)} = 0.6Bmi^{(5)} + 11.9Heavy^{(10)} + 0.4\,Works\_Out^{(1)} + 0.2Age^{(1)} - 2.7Attractive^{(3)} - 0.2\,Tall^{(2)} + 10.6$.

Varying B_{obj}. We continue with the same *Bmi* example but now considering varying online per-object budget. Figure 1d show the errors for this case. We used $B_{\text{prc}} = \$30$ as it is

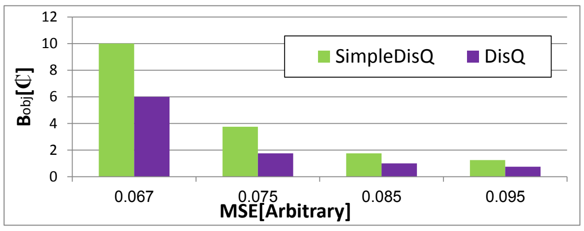
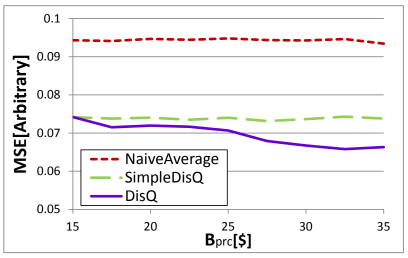


Figure 2: Necessary B_{obj} for achieving target errors

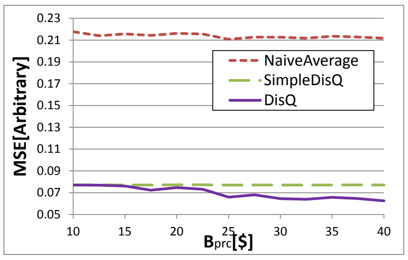
again over the graphs' knee, but similar behavior is shown for other values. First, note that all algorithms improve as B_{obj} increases and that this improvement is slowly decaying. This is what one could expect as a bigger B_{obj} means a bigger crowd (and should therefore mean better accuracy) and since it is known that every additional worker has declining marginal utility. Second, note that both *SimpleDisQ* and *DisQ* achieve lower error than *NaiveAverage*. This clearly shows how combining artificial intelligence with the wisdom-of-crowds leads to improved results. Finally, It is easy to see that the average results from our algorithm are superior to those of the other algorithms. This is especially noticeable for lower B_{obj} budget but is also true for higher B_{obj}. And again, although the we show results for the average case, this is true for most cases. One can see, for example, that in order to achieve an accuracy of less than 0.067 one needs to spend 10¢ per object in *SimpleDisQ* but only 6¢ per object in our algorithm. This statement is still true after including the extra budget for the preprocess phase, since the cost of learning a regression when $B_{\text{obj}} = 10$¢ exceed the \$30 used in our algorithm for $B_{\text{obj}} = 6$¢. In figure 2 we show more examples of the budget necessary for achieving different accuracies in the different algorithms.

Other Examples. Figures 1c and 1f show equivalent graphs for a case of two query attributes (*Bmi, Age*) and figures 1b and 1e show equivalent graphs in the recipes domain (for the query attribute *Protein*). It is easy to see that all of our observations about the first example (*Bmi*) are also true when adding to it a second attribute (*Age*). In the case of *Protein*, however, this is only partly true. Consider first figure 1e. At a first glance it looks different from figure 1d. However, a closer look shows that all our observation still hold. The only difference is that *NaiveAverage* performs much worse and this changes the proportions of the graph. We believe this is because *Protein* is much less intuitive than *Bmi*. Next, consider figure 1b. Here, in addition to the different proportions we also see a different trend. Unlike the previous cases where we saw that increasing B_{obj} always decreased the error, in the case of proteins we see increase in error for $B_{\text{obj}} > 4$¢. The reason for this lies in *CollectingAttributesCondition*. Since our stopping criteria for discovering new attributes depends on B_{obj}, for higher B_{obj} we have less budget for the dismantling which in turn result in less attributes and therefore in a larger error. The effect of a smaller attributes set A_{final} also exists for *Bmi*, but in that case its effect was smaller than the effect of the increased B_{obj}. A reasonable conclusion is that for large B_{obj} budget one should also provide a large B_{prc} budget.

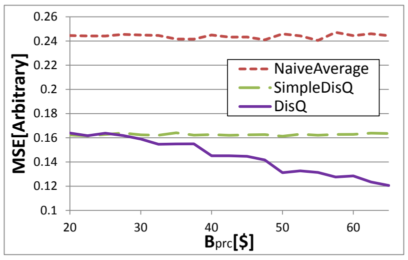
The experiments described above demonstrate the trends in all of our experiments: In all settings our algorithm outperforms the competing algorithms. Increasing improvements are observed when query attributes are difficult and



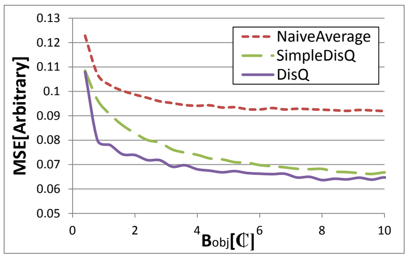
(a) $\mathcal{A}(Q) = \{Bmi\}$



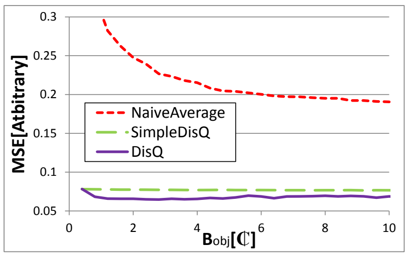
(b) $\mathcal{A}(Q) = \{Protein\}$



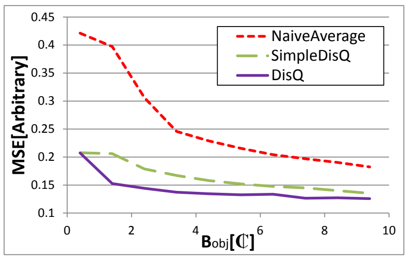
(c) $\mathcal{A}(Q) = \{Bmi,\ Age\}$



(d) $\mathcal{A}(Q) = \{Bmi\}$



(e) $\mathcal{A}(Q) = \{Protein\}$



(f) $\mathcal{A}(Q) = \{Bmi,\ Age\}$

Figure 1: Error in query estimation for varying B_{prc} (top row) and varying B_{obj} (bottom row)

the relative improvement normally grows with the budget allocated to the preprocessing increases, in particular in cases when the per-object online budget is small.

5.3 Algorithm Components

We next examine the individual algorithm components. In particular we analyze our attributes dismantling method and the processing of multiple query attributes.

5.3.1 Dismantling Attributes

We considered two dimensions here.

Finding Relevant Attributes. We first tested if the crowd can give good answers to attribute dismantling questions, and if so, then how. We created gold standard attributes sets for different data domains and query attributes, and tested the crowd coverage for these attributes (percentage of discovered attributes). We computed the performance of our dismantling process and of a naive approach that asks questions only about the attributes explicitly appearing in the query. For defining the gold standard in the pictures domain (for the query attributes *Height* and *Weight*) we used the expert-provided attributes from [27]. For the gold standard in the recipes domain (for the query attributes *Proteins* and textitCalories) we used an expert dietitian.

For all queries our algorithm yielded over 80% coverage, and we compensated for the missing attributes by other discovered attributes not mentioned by the experts. This shows that the crowd could indeed replace the experts for this task. In contrast, the coverage for the naive algorithm fell below 50%, demonstrating the necessity of our choice to dismantle additional attributes. We further validated these observations by considering two additional real-life attribute domains: house prices (using [18] as a gold standard), and laptop prices (using [9]), obtaining similar results.

The GetNextAttribute Method. We next compared our technique for choosing the next attribute to dismantle to a simpler alternative where the only attributes considered are the ones appearing explicitly in the query. We call this variant *OnlyQueryAttributes*. (We also considered variations of *OnlyQueryAttributes* and *DisQ* that chose questions at random, but since those variation are very naive and were consistently inferior to our algorithm we omitted those results). Two example experiment, for the recipes domain with and the query attribute protein are shown in figure 3b (for $B_{\text{prc}} = 30$ and varying B_{obj}) and figure 3a (from $B_{\text{obj}} = 4$¢and varying B_{prc}). First, it is easy to see that our previous observations on *DisQ* in figures 1b and 1e also hold for *OnlyQueryAttributes*. Second, *DisQ* consistently outperform *OnlyQueryAttributes* illustrating again the necessity of our approach. This intensifies as B_{prc} grows since there exist enough budget to learn many attributes so the low variety of answers to the dismantling question only about protein becomes apparent. Similar trends were observed in all settings - different query attributes, query length and domains. The only thing to note is that in some specific cases, when the answers to the dismantling questions about the query attributes were varied enough the difference between the algorithms became noticeable only for large B_{prc}.

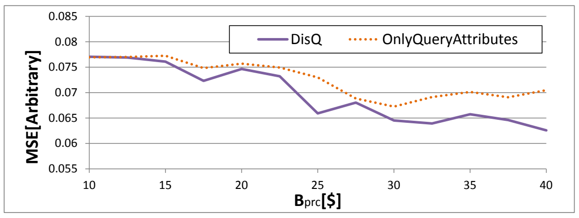
5.3.2 Statistic Estimation

We next examine our method for collecting partial statistics in queries with multiple attributes. As the main issues here are which attributes should be paired with which query attributes, and how to compensate for the missing pairs, we compared our solution to the following baselines.

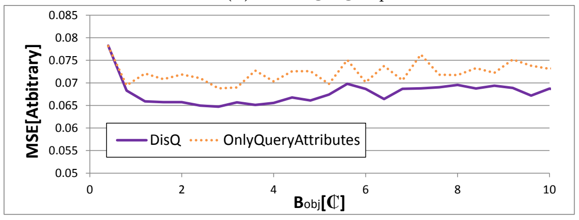
TotallySeparated - This is the naive solution that solves the problem separately for each query attribute, splitting the budget equally between them.

Full - This is a simplified variant of our algorithm that does not optimize the computation and simply gathers statistics for all attribute pairs.

OneConnection - This is another simplified variant that does consider only some of the pairs, but uses a more naive heuristic for choosing them: When a new attribute is discovered, it is paired only with one query attribute.

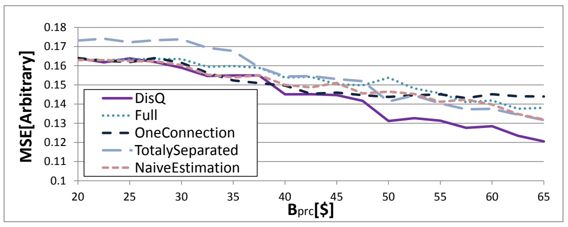


(a) Changing B_{prc}

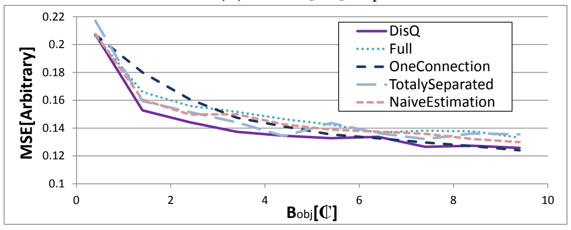


(b) Changing B_{obj}

Figure 3: Error in estimation for $\mathcal{A}(Q) = \{Protein\}$



(a) Changing B_{prc}



(b) Changing B_{obj}

Figure 4: Error in estimation for $\mathcal{A}(Q) = \{Bmi, Age\}$

NaiveEstimations - Finally, this variant selects the pairs using our technique, but rather than inferring individual values for the missing pairs, it assigns to all a default value that equal to the average S_o value.

A sample of the results, for the pictures domain and the query attributes *Bmi* and *Age* are shown in figures 4a ($B_{\text{obj}} = 4$¢, varying B_{prc}) and figure 4b ($B_{\text{prc}} = \$50$, varying B_{obj}). We set here B_{prc} to \$50 to highlight the trends, as we will shortly discuss. First, note that all the variations follow the general trends of our algorithm as discussed above, in regard for their dependencies in B_{prc} and B_{obj}. Second, it is easy to see the relatively bad performances of the *TotallySeperated* baseline (especially for lower B_{prc}) which demonstrates the advantage of asking about several query attributes together. Third, in comparison to *Full*, our algorithm consistently achieves better (or at least as good) results when considering reasonable B_{prc} (as in figure 4a). This effect was even more noticeable in the synthetic domain where we could test large queries. For some queries, however, these trends change for very high B_{prc} as the saved budget from the non-important pairs is wasted on even more non-important new attributes. Next, in compare to *OneConnection*, our algorithm achieves at least as good results for all budgets, and better results for high B_{prc}. The reason for that is that for large budgets *OneConnection* saves budget in the beginning on redundant connections, but that budget is then referred to even more redundant attributes. In some cases, however, and for low B_{prc}, *OneConnection* did get better results, but only very marginally. The reason is the tradeoff between B_{obj} and flexible-B_{prc} we discussed before which effects *DisQ* more. Finally, our algorithm consistently outperforms *NaiveEstimations*. This is true for every budget since our estimation method incures no crowd cost.

5.4 Dependency on Assumptions

Our last set of experiments examined the robustness of our algorithm to some changes in underlying our assumptions. We briefly discuss the assumptions considered and our conclusions. A detailed description of the experiments and results can be found in our full paper [22].
Attributes Quality: We tested resilience to receiving also some irrelevant attributes in the dismantling process. This did not affect the previous trends, but as expected required somewhat higher preprocessing budget (B_{prc}) for obtaining same error rates.
Normalization Mechanism: We tested the necessity of identifying different answers about the same property as one. Here again, our algorithm can work with imperfect (or even no) unification and the trends stay the same. Somewhat higher B_{prc} is again needed for obtaining same error rates.
Answer's Correlation Parameter: We used different constants (instead of just 0.5) for $\mathbb{E}[\rho(a_j, ans_j)]$ when estimating $S_{A_{m|a_j}}$. The results remained similar.
Crowd-Tasks Payment: We tested how a different pricing models for crowd tasks impact the results. Change in prices changed some of the gradients in the varying-B_{prc} graphs but the trends remained the same.

6. RELATED WORK

Using the crowd as a source of knowledge, and for solving problems, has attracted much research in recent years [11]. The crowd was shown to be a useful tool for many types of tasks, including, but not restricted to, value estimation [24], data filtering [25], information collection [5], natural languages processing [6] etc. However, to our knowledge, our work is the first to consider using the crowd for discovering query-related attribute names. There has also been much work dealing with the collection of data via various platforms (e.g., payment [2] or games [10]), and the effective collection of such data (e.g., how to best present questions [13], how to filter spam [19], when to stop asking [30] etc.). Our work exploits these platforms and previous results. The concept of removing experts from crowd processes was also researched before[14], but not in the context of query estimation.

Our work is also influenced by previous work in machine learning, and on the use of supervised learning for regression learning (e.g., [12]). A more specific problem related to our challenge is feature selection [17] - how to effectively narrow a set of attributes for some learning process. Two models that are particularly interesting are budget learning ([23]), where the issue is deciding which is the most valuable feature to measure next under a limited budget, and meta-features (e.g., [28]), where the issue is trying to predict unseen features behavior based on some properties and similarity to other features. All of those problems, however, focus on a given predefined set of attributes. They also do not consider the selection of the same attribute name more than once (required here due the uncertainty of crowd answers).

Previous work has also dealt with the combination of crowd and learning (e.g., [8]). The common combinations are to use the crowd to label a data set (e.g., [32]) or for filling attributes values (e.g., [21], [26]).Closest to our work is that of [27] which also deals with estimating one attribute value by asking about others, but, as explained in the Introduction, requires experts-in-the-loop. While we use some of their results as basic building blocks, a major contribution here is our crowd-based attribute dismantling along with the careful statistical analysis that allows for an effective experts-free algorithm.

7. CONCLUSION AND FUTURE WORK

We studied in this paper the problem of query evaluation when the value of the queried attributes is not available in the database and is also hard for the crowd to estimate. We proposed a novel approach that uses the crowd to dismantle the query attributes into finer related ones (whose value estimation is easier), then assembles them to yield better estimation for the query attributes. Given an online per-object budget and an offline preprocessing budget, we presented an algorithm that ideally uses the offline budget for dismantling the query attributes and deriving linear formulas that best exploit the online budget for deriving the values of query attributes. We have also demonstrated the effectiveness of the approach through experimental study on both real-life crowd and synthetic data.

We focus in the paper on the minimization of the expected-mean-square-error. Other error measures may also be of interest for future research. For example, a recall-precision measurement may fit more for boolean query attributes like *gluten_free* (for recipes), or for a categorical attribute like *cousin_type* where the number of possibilities may be large. We also considered only linear formulas for assembling attributes values. While this has proved to provide good experimental results, more general rules may be useful in certain situations and we intend to study this in future work. In our development we assumed that we are given an on-line per-object budget and an offline preprocessing budget and used the later to optimize the usage of the former. Determining automatically what these budgets should be and the ideal ratio between them is an intriguing future research.

Acknowledgements

This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, and by the Israel Ministry of Science.

8. REFERENCES

[1] AllRecipes.com. `http://allrecipes.com`.

[2] Amazon Mechanical Turk. `https://www.mturk.com`.

[3] Crowd Flower. `http://www.crowdflower.com`.

[4] The height and weight chart. `http://www.cockeyed.com/photos/bodies/heightweight.html`.

[5] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowd mining. In *SIGMOD Conference*, 2013.

[6] C. Biemann. Creating a system for lexical substitutions from scratch using crowdsourcing. *Language resources and evaluation*, 47(1), 2013.

[7] W. Bousfield and C. Sedgewick. An analysis of sequences of restricted associative responses. *The Journal of General Psychology*, 30(2), 1944.

[8] A. Brew, D. Greene, and P. Cunningham. The interaction between supervised learning and crowdsourcing. In *NIPS workshop on computational social science and the wisdom of crowds*, 2010.

[9] P. D. Chwelos, E. R. Berndt, and I. M. Cockburn. Faster, smaller, cheaper: an hedonic price analysis of pdas. *Applied Economics*, 40(22), 2008.

[10] D. Deutch, O. Greenshpan, B. Kostenko, and T. Milo. Declarative platform for data sourcing games. In *WWW*, 2012.

[11] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4), 2011.

[12] N. R. Draper and H. Smith. *Applied regression analysis 2nd ed.* New York New York John Wiley and Sons 1981., 1981.

[13] C. Eickhoff and A. P. de Vries. Increasing cheat robustness of crowdsourcing tasks. *Inf. Retr.*, 16(2), 2013.

[14] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD Conference*, 2014.

[15] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5), 1970.

[16] S. B. Green. How many subjects does it take to do a regression analysis. *Multivariate behavioral research*, 26(3), 1991.

[17] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3, 2003.

[18] D. Harrison Jr and D. L. Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1), 1978.

[19] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, 2010.

[20] L. P. Kaelbling. *Learning in embedded systems.* MIT press, 1993.

[21] E. Kamar, S. Hacker, and E. Horvitz. Combining human and machine intelligence in large-scale crowdsourcing. In *AAMAS*, 2012.

[22] M. Laadan. Dismantling complicated query attributes with crowd. Master's thesis, Tel-Aviv University, 2014.

[23] D. J. Lizotte, O. Madani, and R. Greiner. Budgeted learning of naive-bayes classifiers. *CoRR*, abs/1212.2472, 2012.

[24] J. Noronha, E. Hysen, H. Zhang, and K. Z. Gajos. Platemate: crowdsourcing nutritional analysis from food photographs. In *UIST*, 2011.

[25] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. CrowdScreen: algorithms for filtering data with humans. In *SIGMOD Conference*, 2012.

[26] G. Patterson and J. Hays. SUN attribute database: Discovering, annotating, and recognizing scene attributes. In *CVPR*, 2012.

[27] S. Sabato and A. Kalai. Feature multi-selection among subjective features. In *ICML (3)*, 2013.

[28] B. Taskar, M. F. Wong, and D. Koller. Learning on the test data: Leveraging unseen features. In *ICML*, 2003.

[29] A. Towsley, J. Pakianathan, and D. H. Douglass. Correlation angles and inner products: Application to a problem from physics. *ISRN Applied Mathematics*, 2011.

[30] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, 2013.

[31] A. Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2), 1945.

[32] P. Ye and D. Doermann. Combining preference and absolute judgements in a crowd-sourced setting. In *ICML (3)*, 2013.