Exploiting Correlations for Evaluating Complex Queries

Manas Joglekar
Stanford
manasrj@cs.stanford.edu

Hector Garcia-Molina
Stanford
hector@cs.stanford.edu

Aditya Parameswaran
U. Illinois and MIT
adityagp@illinois.edu

Christopher Re
Stanford
chrismre@cs.stanford.edu

ABSTRACT

User Defined Function(UDFs) are used increasingly to augment query languages with extra, application dependent functionality. UDFs tend to be expensive, either in terms of monetary cost or latency. In this paper, we study ways to efficiently evaluate selection queries involving UDFs. We provide a family of techniques for processing queries at low cost while satisfying user-specified precision and recall constraints. Our techniques are applicable to a wide variety of scenarios, such as when selection probabilities of tuples are available beforehand, when this information is available but noisy, or when no such prior information is available. We also generalize our techniques to more complex queries. Finally, we test our techniques on real datasets, and show that they achieve significant savings in cost of upto 80%, while incurring only a small reduction in accuracy.

1. INTRODUCTION

User defined functions (UDFs) provide query languages with extra, application-dependent functionality. UDFs are especially important in data science, enabling data scientists to augment their workflows with complex computation, including calls to machine learning algorithms or external APIs. Overall, UDFs come in many flavors: they could invoke external services (e.g., calls to an up-to-date weather monitoring service, or a credit check service), expensive algorithms (e.g., image, video or text analysis algorithms), or even crowdsourced workers (e.g., workers on Mechanical Turk in CrowdDB or Deco [14, 28]). In all of these cases, UDFs are costly to evaluate, either (i) in terms of time, for instance in the expensive algorithm scenario, or (ii) in terms of monetary cost, for instance in the crowdsourcing or external service call scenario.

In this paper we study techniques for reducing the number of UDF invocations during query execution, provided the user tolerates some reduction in accuracy. In our case we define accuracy in information retrieval terms (and not in terms of errors of individual values). That is, say C (a set of tuples) is the correct result of a query (performing all UDF calls), and R is the approximate result. Then the precision is $|R \cap C|/|R|$ and the recall is $|R \cap C|/|C|$. The user specifies bounds on precision and recall to indicate what

accuracy is acceptable.

EXAMPLE 1.1. *To motivate our approach, consider the following simple example. A user wants to contact customers with a very good credit rating to offer a special promotion. The following query describes his needs:*

$$Q : \text{SELECT} * \text{FROM R(A, ID) WHERE f(ID)} = 1.$$

Here f *is a UDF, say, a call to a credit check service —returning 1 (for "good") and 0 (for "bad")—taking as input ID the identity of the customer.*

If the user requires perfect precision and recall, then the system must retrieve every R *record and check if its* f(ID) *value is* 1. *The cost, either in terms of time taken or money paid to the credit bureau, will be high.*

Instead say the user is willing to tolerate a slightly lower value of precision and/or recall, say 90% for each. (The precise meaning of this bound will be discussed later.) In other words, the user is willing to pitch the promotion to a few customers with a bad credit rating, and to miss a few good customers, in order to get the query results significantly faster.

Then, one straightforward query execution strategy is to simply evaluate f *on as many tuples as necessary in sequence until the recall and precision constraints are met. Thus, we will end up making close to* $90\% \times 3000 = 2700$ *calls to the credit check* f *function.*

Another approach, which we will explore in depth in this paper, is to estimate and then exploit correlations. In our example, let us assume there is an attribute A *of* R *that is correlated to the output of* f. *For instance,* A *could be an income attribute or a housing status that could indicate a higher likelihood for having a good credit score. (We discuss later how we can discover one or more attributes that may be correlated to the UDF.)*

Step 1: Estimating Correlations. *One way to estimate correlations is by sampling some* R *tuples. For example, after reading 5% of the tuples with each* A *value, the system has the following estimates:*

- *If* A=1, f(ID) = 1 *with probability 0.9*

- *If* A=2, f(ID) = 1 *with probability 0.5*

- *If* A=3, f(ID) = 1 *with probability 0.1.*

(As we discuss later, sampling is not the only way to obtain the statistics needed by our next step.) The system also knows that there are 1000 tuples with each of the possible A *values, for a total of 3000* R *records.*

Step 2: Executing Query. *Then, a better query execution strategy is to handle tuples with each* A *value differently: Tuples with* A=1 *are very likely to satisfy the predicate, so we can add them to*

the result without evaluating f. *Similarly, tuples with* A = 3 *and unlikely to match, so we can simply discard them. Tuples with* A = 2 *can go either way, so we evaluate* f *for all of them and only add to the result matching tuples. With this strategy we only need to evaluate* f *1000 times (plus the evaluations needed for sampling), not the 2700 calls of the straightforward approach, and we still get an expected precision and recall of (approximately) 90%.*

Note that the execution strategy illustrated above is one of many: for every group of tuples that shares the same A *value, we can choose to (a) directly add all of them to the result without evaluating* f, *(b) discard all of them, or (c) evaluate all of them and then decide whether to add it to the result or not. Thus, if* n *is the number of distinct values of* A, *we have* 3^n *possible alternative execution strategies. We could also employ probabilistic execution strategies, where we toss a biased coin for each tuple in a group of tuples sharing the same* A *value, and decide what to do based on the outcome. Overall, it is not immediately obvious which of these strategies we should use in order to meet our precision and recall guarantees. Indeed, as we will see in the following, designing optimized execution strategies can be shown to be NP-Hard.*

To summarize, our approach involves avoiding UDF evaluations by exploiting *correlations* between UDF calls and values of other attributes. In our example we have swept a number of details and possible generalizations "under the rug" in order to present the essence of our approach. The details will be formalized in the sections that follow, but for now we make a few clarifications:

- What we called attribute A in our example, could be several attributes or even a virtual attribute representing the output of a machine learning algorithm that predicts the f output based on values in a tuple. That is, with appropriate training data we may develop a less expensive function f′ that predicts f (and also provides confidences for its predictions). With such a function we can skip Step 1, and in step 2 use the f′ values to decide whether to discard a tuple, output it immediately, or evaluate the expensive f function. In this paper we will focus on the sampling approach because it makes it easier to account for the cost of obtaining correlations.

  That said, our techniques for step 2 are agnostic to where the confidences come from, and can still be used. As a result, we can certainly use confidences generated by say, logistic regression or least squares estimation, or any other statistical techniques. We experimentally evaluate the impact of where the confidences come from (including sampling and logistic regression) in Section 6.

- We assume that f values have not been cached in advance. If the values are available, then the expense of evaluating f on every tuple has been paid in advance, and conventional query processing strategies can be used. Our techniques can be applied even if some of the values are available in advance. Note however that in many cases f outputs are time sensitive and should not be cached. For example, weather predictions or credit ratings change continuously.

We reiterate that our approach hinges on two assumptions: (a) that the user is willing to accept lower precision and/or recall, and (b) that there is an attribute (possibly virtual, possibly multiple attributes) that is correlated to the UDF output. For (a), we believe that users are familiar with document retrieval models where it is not practical to get the complete and exact answer to a query. Often, the output of the UDF itself is subjective or approximate (e.g., whether a patient is prone to a particular illness, or an image is inappropriate), so errors cannot be ruled out no matter what we do. For (b), if correlated attributes are not known in advance, there are well known techniques for learning what attributes are good pre-

dictors, and for combining these attributes into a virtual attribute that can be used by our solution.

When these assumptions hold, we will show (with real data sets) that queries can be evaluated very efficiently. These savings will grow in importance as data sets grow in size and UDFs become more and more popular. In our experiments we will also study various performance related questions: For example, how much sampling should one do in Step 1? As we sample more, our statistics improve, but our cost savings decrease. How sensitive is our approach to the number of A values? As the number of values grows, the number of tuples with a given value shrinks, possibly making it harder to get estimates.

Prior work has addressed the optimal placement of UDF evaluation in query plans [11, 12, 17, 19], assuming users have provided hints for costs of UDFs; in our work, we aim to avoid UDF evaluations entirely by exploiting correlations. Our work is also similar to the work on exploiting correlations between sensors to reduce evaluations in the sensor networks domain [13], however, the scenario is very different, necessitating different approaches. Related work is surveyed in more detail in Section 7.

Contributions: We propose a novel approach to optimize expensive UDF optimization by exploiting correlation information. Specifically,

- In Section 2, we introduce the relevant notation for the setup of the problems we consider.
- In Section 3, we study the case where correlations are known, and can be exploited — we consider three scenarios where we can exploit existing correlations, and we describe essentially optimal algorithms for each scenario.
- In Section 4, we study the case where correlations are not known in advance, and we must jointly estimate correlations, and exploit them. We describe a method of obtaining correlation information, how to use this information adaptively, and provide a rule of thumb to use in practice.
- In Section 6, we experimentally evaluate the performance of our algorithms on two real datasets, and also demonstrate the correctness of the algorithms.

In Section 5, we mention some extensions of our methods to more advanced SQL queries that involve more than one table and/or UDF predicate.

2. PRELIMINARIES

We begin by considering queries like our example query Q in Section 1. We reproduce the query here:

Q : SELECT $*$ FROM R(A, ID) WHERE f(ID) = 1

The relation R contains an attribute A, the values of which are correlated with the output of f(ID) across tuples. It is straightforward to generalize to queries with more correlated attributes, with joins or projects, or with categorical predicates for f with operators other than equality. (We discuss generalizations in Section 5.)

To assist the description of our terminology, we provide example data for R in Table 1. Naturally, f(ID) is not known in advance (and instead must be computed using UDF evaluations), but is also shown along with the table. The tuples in R that satisfy the predicate (i.e., f(ID) equals 1) are called *correct*, while those that do not are called *incorrect*. Thus, in Table 1, tuples 1-4, 6, and 12 are correct, while the rest are incorrect.

Groups of Tuples: We use A to denote both the correlated attribute, as well as the set of distinct values that appear in R.A. Therefore, we use $a \in A$ to denote a value that attribute A can take. In Table 1, a can be 1, 2, or 3.

Tuple No.	A	ID	f(ID)
1	1	999-999-999	1
2	1	913-418-777	1
3	1	719-334-111	1
4	1	999-999-999	1
5	2	913-418-737	0
6	2	719-334-113	1
7	2	999-999-299	0
8	3	913-418-737	0
9	3	719-334-121	0
10	3	999-999-959	0
11	3	913-418-727	0
12	3	719-334-311	1

Table 1: Example Data for R

The set of tuples that share the same value of A is called a *group*. We let t_a be the number of tuples in the group that has A=a. Thus, $t_1 = 4, t_2 = 3$, and $t_3 = 5$ in our example. In a group corresponding to A=a, we let c_a denote the number of correct tuples (i.e., tuples satisfying the predicate), and w_a be the number of incorrect tuples. Thus, for $a = 2$, $c_2 = 1$, while $w_2 = 2$.

Note that while we assume c_a, w_a to be known in our example (and also in our initial setting in Section 3.1), typically these values are not known, and are treated as random variables. In such cases, we use upper case C_a and W_a to denote the random variables. We assume t_a for all $a \in A$ is always known.

Actions and Costs: For our simple query Q, we must decide if each tuple in R is in the result. We have three alternative *actions* we could take:

- First, we could *discard* the tuple, i.e., no action is taken on the tuple, and the tuple does not contribute to the output. In this case, we are predicting that the tuple is not correct.
- Next, we could *retrieve but not evaluate* the tuple, i.e., retrieve the tuple from R, and add it to the result without actually evaluating the UDF. In this case, we are predicting that the tuple is correct.
- Last, we could *retrieve and evaluate* the tuple, i.e., retrieve the tuple from R, evaluate the UDF on the tuple, and add the tuple to the result if it satisfies the condition f(ID)=1. In this case, we are certain to be accurate in our assessment of the tuple.

We assume that we incur a cost of c_e for every tuple evaluated, and a cost of c_r for every tuple retrieved from storage. Since UDFs are expensive, c_e is likely to be much greater than c_r. Thus, the cost of discarding a tuple is 0, the cost of retrieving and not evaluating is c_r, and the cost of retrieving and evaluating is $c_r + c_e$. Note that this cost model implies we have some type of index on A so we can reach the examined tuples with constant cost independent of the discarded tuples.

We now present expressions denoting costs across tuples. We first define the following terms for convenience:

- R_a^+ denotes the total number of correct tuples that we retrieved from group a, while R_a^- denotes the total number of incorrect tuples that we retrieved from group a.
- E_a^+ denotes the total number of correct tuples that we evaluated from group a. E_a^- denotes the total number of incorrect tuples we evaluated from group a.

Then, our overall cost, which we will aim to minimize, is:

$$\mathcal{C} = \sum_{a \in A} c_r(R_a^+ + R_a^-) + c_e(E_a^+ + E_a^-) \quad (1)$$

Metrics and Constraints: Our output must meet a user-specified *precision* and *recall* constraint. As defined earlier, precision is the fraction of tuples in the output that are correct. Using our notation, $\sum_{a \in A} R_a^+$ is the number o correct tuples that we return, whereas $\sum_{a \in A} R_a^+ + R_a^- - E_a^-$ is the total number of tuples we return (incorrect tuples that are retrieved and evaluated get discarded). Hence,

$$\mathcal{P} = \frac{\sum_{a \in A} R_a^+}{\sum_{a \in A} R_a^+ + R_a^- - E_a^-}$$

Analogously, recall is the fraction of the correct tuples that are present in the output, i.e.,

$$\mathcal{R} = \frac{\sum_{a \in A} R_a^+}{\sum_{a \in A} C_a}$$

The user specifies a precision lower-bound $\alpha \in [0, 1]$, and a recall lower-bound $\beta \in [0, 1]$. Our output must then meet the following criteria:

$$\mathcal{P} \geq \alpha; \quad \mathcal{R} \geq \beta$$

A special case of interest is the *browsing* scenario, where 100% precision is required, and thus we have to evaluate every tuple we retrieve, and our objective is to minimize expected cost while satisfying the recall constraint.

Probabilistic Constraints: The above bounds are strict, but if we are making discard and evaluate decisions based on imprecise statistics, there is a (hopefully small) chance that the output will violate the bounds. Thus, we also allow the used to define a *satisfaction probability* ρ (which we expect to be very close to 1). The system should then guarantee that with probability ρ the precision and recall constraints are met.

3. EXPLOITING CORRELATIONS

In the example of Section 1 we described the two steps of our approach: Step 1 obtains correlations, and Step 2 takes this correlation information as input to process the query efficiently. We consider three forms of such correlations::

- Perfect Information: In this case, Step 2 receives both W_a and C_a (the number of wrong and correct tuples, respectively) for all a. This case is included simply as a baseline, so we can compare the other more realistic cases against it.
- Perfect Selectivities: in this case, we know the probability of each tuple in a group being correct, i.e., s_a for each group a. If we take a random tuple from group a, then f(ID)=1 for that tuple with probability s_a independent of other tuples.
- Estimated Selectivities: in this case we do not know the selectivity s_a precisely. That is, we have a random variable S_a that represents the selectivity of group a. Selectivity s_a is now the mean of S_a and v_a is the variance of S_a.

Exploiting these different types of correlations is the technical focus of this section.

3.1 Warm-up: Perfect Information

We begin by considering the *perfect information* case, i.e., we know, in advance, the precise number of correct and incorrect tuples in each group. That is, for each $a \in A$, we know the values of C_a, i.e., the number of correct tuples, and W_a, the number of incorrect tuples. Thus, in this section, these values are constants. While these assumptions are not completely realistic (we will rarely have such information available in advance), this section will demonstrate that even when perfect information is available, the problem of selecting which groups to retrieve and evaluate, which groups to

retrieve but not evaluate, and which groups to discard is already intractable. Furthermore, the section will act as a introduction to the more complex schemes that we describe in subsequent sections.

Query Optimization and Execution: Since the exact number of correct and incorrect tuples is known, our processing of this query proceeds in two steps:

- **Optimization:** We determine, for each group, whether to (a) retrieve and evaluate, (b) retrieve but not evaluate, or (c) directly discard the tuples in that group. In this section, we focus on *deterministic* schemes; that is, we pick one of (a) (b) or (c) for each group to minimize the cost objective $\mathcal{C}$ while meeting precision and recall constraints. (We will consider probabilistic schemes in subsequent sections.) No query execution happens at this point.
- **Execution:** Then, we execute the query by retrieving all the tuples in the groups for which we either chose to do (a) or (b) in the previous step, then we evaluate all the tuples in the groups for which we chose to do (a).

Optimization Problem: Given the values of C_a and W_a for each a, the goal of our optimization step is to output boolean decision variables R_a and E_a to satisfy recall and precision constraints while minimizing cost. The boolean variable R_a is set to 1 if the tuples in that group with $A = a$ are to be retrieved during query execution (and 0 if the tuples are to be discarded), and the boolean variable E_a is set to 1 if the tuples in that group with $A = a$ are to be evaluated during query execution (and 0 if they are not). Note that $R_a \geq E_a$ for each $a \in A$ since we may only evaluate the UDF on a tuple if it is retrieved first.

We now rewrite the constraints in the previous section in terms of the new boolean variables R_a, E_a. The recall constraint becomes:

$$\sum_{a \in A} C_a R_a \geq \beta \sum_{a \in A} C_a \qquad (2)$$

Note that given C_a values, the right hand side of the inequality is a constant $\gamma = \beta \sum_{a \in A} C_a$. The precision constraint becomes:

$$\frac{\sum_{a \in A} C_a R_a}{\sum_{a \in A} C_a R_a + W_a(R_a - E_a)} \geq \alpha$$

$$\Leftrightarrow \sum_{a \in A} ((\frac{1}{\alpha} - 1)C_a - W_a)R_a + W_a E_a \geq 0 \qquad (3)$$

Then, the optimization problem can be restated as:

PROBLEM 1 (PERFECT-INFORMATION). *Given $C_a, W_a, t_a \forall a$, identify $R_a, E_a \in \{0, 1\}; R_a \geq E_a$ to minimize $\mathcal{C} = \sum_{a \in A}(W_a + C_a)(R_a c_r + E_a c_e)$, such that Constraints 2 and 3 are satisfied.*

We demonstrate that Problem 1 is NP-hard, which motivates the more relaxed semantics throughout this section.

Hardness of Special Case. We can show that Problem 1 is NP-hard even when there is no precision constraint. Since Problem 1 can be cast as an Integer Linear Program (ILP), overall, the problem is NP-Complete.

THEOREM 3.1. *Problem 1 is NP-Complete in $|A|$ (i.e., the number of distinct values of A).*

PROOF. (Sketch) Let $W_a = C_a, \alpha = 0, \beta = 1/2$, then Constraint 3 is trivially satisfied, and the objective becomes ($E_a = 0$ since we don't have any other constraint on E_a):

$$\text{Minimize} \sum_{a \in A} C_a R_a, \text{ such that } \sum_{a \in A} R_a C_a \geq \sum_{a \in A} \frac{1}{2} C_a$$

We may reduce the NP-Hard Partition problem to this special case, with the integers in the partition problem being mapped to C_a values. □

Complete proofs of this as well as other results in the paper can be found in the appendix of the extended technical report [24].

The hardness result continues to hold when
- C_a is any constant multiple of W_a (for all a)
- β is set to any arbitrarily small or large number

As a result, even very simple versions of the problem are already intractable. On the other hand, the hardness result does not hold when there is only a precision constraint (and no recall constraint)—the precision constraint can be trivially satisfied by retrieving no tuples.

3.2 Perfect Selectivity Information

In the previous section, we demonstrated that the optimization problem to generate deterministic execution schemes meeting precision and recall guarantees exactly, was NP-Hard. In this section, we make the following modifications to the problem in the previous section:

- **Selectivities:** We now assume that $W_a, C_a, \forall a \in A$ are not provided to us. Instead, we assume that we know the selectivity s_a of each group $a \in A$, i.e., for a tuple with $A = a$, the probability of it being correct is independently true with probability s_a. These selectivities could be learned using historical data. We assume that the selectivity values are *perfect*. (In the next section, we consider selectivity values that may be *imperfect*.)
- **Satisfaction Probability:** Since we are dealing with selectivities which are probabilities, rather than deterministic W_a, C_a values, there are always adversarial inputs on which any query execution strategy would perform poorly. Thus, in this section (and subsequent ones) we assume that the user has given us a probability value ρ such that our precision and recall constraints must be met with probability ρ. We expect ρ to be very close to 1.
- **Probabilistic Schemes:** Since we are dealing with probabilities for both the quantities mentioned previously, it is more natural to consider probabilistic execution strategies rather than deterministic ones.

We now present the problem formally.

Query Optimization and Execution: As before, our processing proceeds in two steps:

- **Optimization:** We determine, for each group, whether to (a) retrieve and evaluate, (b) retrieve but not evaluate, or (c) directly discard the tuples in that group. We do this by generating probabilities $0 \leq E_a, R_a \leq 1, \forall a \in A$. The objective of this step is to choose R_as and E_as such that the results after execution (described next) satisfy the precision and recall constraints with probability at least ρ, while minimizing expected cost $\mathcal{C}$. No query execution happens at this point.
- **Execution:** We then execute the query by retrieving each of the tuples in group a with probability R_a, independent of all other tuples, and evaluate each of the retrieved tuples from group a with probability $\frac{E_a}{R_a}$, again independently of all other tuples. (Note that these two probabilities when multiplied together give E_a, which is precisely the probability we evaluate a tuple from group a.) Tuples that are retrieved but not evaluated are returned in the output (that is, we assume those tuples are correct).

Optimization Problem: For each $a \in A$, we are given the selectivity s_a of group a, i.e., each tuple in group a is correct with probability s_a independent of other tuples.

Our constraints can be expressed as follows: We have the basic constraint $1 \geq R_a \geq E_a \geq 0$. Recall that R_a^+ denotes the random variable corresponding to the number of correct tuples retrieved from group a (similarly for R_a^-, E_a^+, E_a^-.) The precision constraint, which must be satisfied with probability ρ, can be expressed as:

$$\sum_{a \in A} R_a^+ - \alpha \sum_{a \in A} (R_a^+ + R_a^- - E_a^-) \geq 0 \qquad (4)$$

The recall constraint, which again must be satisfied with probability ρ, can be expressed as:

$$\sum_{a \in A} R_a^+ - \beta \sum_{a \in A} C_a \geq 0 \qquad (5)$$

Since our solution is now probabilistic, we try to minimize the expected value of the objective.

$$\mathcal{C} = E\left[\sum_{a \in A} c_r(R_a^+ + R_a^-) + c_e(E_a^+ + E_a^-) \right]$$
$$= \sum_{a \in A} (W_a + C_a)(c_r R_a + c_e E_a) = \sum_{a \in A} t_a(c_r R_a + c_e E_a)$$

PROBLEM 2 (PERFECT-SELECTIVITIES). *Given $s_a, t_a \forall a$, identify $0 \leq E_a \leq R_a \leq 1$ to minimize $\mathcal{C} = \sum_{a \in A} t_a(c_r R_a + c_e E_a)$, such that Constraints 4 and 5 are satisfied with probability greater than ρ.*

In the next few subsections, we will (a) describe a LP-based solution for Problem 2 (b) demonstrate that the LP-based solution is asymptotically optimal (c) describe how we may speed up the computation of the LP, by leveraging the special structure of the LP.

3.2.1 LP-Based Solution

First, notice that the expected value of the LHS of Constraint 4 can be rewritten as follows:

$$G^p = \sum_{a \in A} t_a s_a (1 - \alpha) R_a + t_a(1 - s_a)\alpha(E_a - R_a) \qquad (6)$$

We get the above equation by substituting $E[R_a^+] = s_a t_a R_a$, $E[R_a^-] = (1 - s_a)t_a R_a$ (the E_a equations are similar). Similarly, the expected value of the LHS of Constraint 5 can be rewritten as follows:

$$G^r = \sum_{a \in A} t_a s_a R_a - \sum_{a \in A} \beta t_a s_a \qquad (7)$$

Our approach will be to ensure that G^p and G^r, i.e., the expected values of quantities closely related to precision and recall, are greater than some carefully chosen thresholds t_ρ^p and t_ρ^r respectively, to ensure that the corresponding constraints 4 and 5 are satisfied with probability ρ.

We define t_ρ^p, t_ρ^r as follows:

$$t_\rho^p = \sqrt{\frac{\log(1 - \rho)\sum_{a \in A} t_a}{2}} \qquad (8)$$

$$t_\rho^r = \sqrt{\frac{\log(1 - \rho)\sum_{a \in A} t_a(1 - \beta)}{2}} \qquad (9)$$

Then, consider the following linear program:

LINEAR PROG. 3.2 (PERFECT-SELECTIVITIES). *Minimize* $\sum_{a \in A} t_a(R_a c_r + E_a c_e)$ *subject to:*

$$\sum_{a \in A} t_a s_a(1 - \alpha)R_a + t_a(1 - s_a)\alpha(E_a - R_a) \geq t_\rho^p \qquad (10)$$

$$\sum_{a \in A} t_a s_a R_a - \sum_{a \in A} \beta t_a s_a \geq t_\rho^r \qquad (11)$$

$$\forall a \in A; 1 \geq R_a \geq E_a \geq 0$$

It can be shown that solving the linear program above gives us R_a and E_a that satisfy the precision and recall constraints with probability at least ρ. Our proof uses repeated applications of Hoeffding's inequality, union bounds [30], as well as other standard results in probability. The proof of the following theorem, along with those for other theorems, can be found in the appendix.

THEOREM 3.3. *Solving Linear-Prog. 3.2 provides a feasible solution for Problem 2.*

The theorem above only states that the solution will be feasible, but not how close the resulting cost $\mathcal{C}$ is to the optimal cost. Thankfully, as we will show below, our solution is close to optimal.

Tightness. We state the tightness bound on our result below.

THEOREM 3.4. *Let s_a^{min} be the smallest non-zero value of s_a. Then, the difference between the cost of the optimal solution to Linear-Prog. 3.2, and the cost of the optimal solution to Problem 2, is at most*

$$(c_e + c_r)\frac{1}{s_a^{min}}\max(t_\rho^r + t_{1-\rho}^r, \frac{t_\rho^p + t_{1-\rho}^p}{1 - \alpha})$$

This cost difference described in the previous theorem is $O(\sqrt{n})$ where n is number of tuples, because the t^p, t^r values are $O(\sqrt{n})$. Now consider what happens as $n \to \infty$ for fixed s_a's, α, β. This means we have the same constraints, and groups with the same selectivities, but the group sizes are scaled up. As $n \to \infty$, the ratio of our cost to optimal cost $\to 1$, as indicated by the theorem below:

THEOREM 3.5. *As $n \to \infty$, the ratio of the cost of the optimal solution to Linear-Prog. 3.2, to the cost of the optimal solution to Problem 2 approaches 1.*

Thus, our solution for Problem 2 is asymptotically optimal.

3.2.2 Greedy Solution for the linear problem

In this section, we describe an efficient way to solve Linear-Prog. 3.2 without even using a linear solver, in $O(|A| \log |A|)$ time:

THEOREM 3.6. *If the following constraints hold:*

$$t_\rho^p < \sum_{a \in A} \max(t_a(s_a - \alpha), 0)$$

$$t_\rho^r < \sum_{a \in A} (1 - \beta)t_a s_a$$

then the solution to Linear-Prog 3.2 can be found in $O(|A| \log |A|)$

The first constraint simply states that the variance term t_ρ^p is not too large (the constraint is trivially satisfied if the term is 0). It ensures that the precision constraint can be satisfied without evaluating any tuple with selectivity $\geq \alpha$, while the second constraint simply ensures that the problem has a solution.

Our algorithm is the following:

- Initialize all R_a and E_a to 0. Sort the a's based on s_a.
- Then start increasing the values of R_a in a greedy fashion. That is, increase R_a for the highest selectivity a until it becomes 1, then increase R_a for the next highest selectivity a, and so on. Keep doing repeating this step until the recall constraint Equation 11 is satisfied.
- After that, keep the R_as fixed, and start increasing the E_as in a greedy fashion, but in reverse order. That is, start by increasing E_a for the a with lowest selectivity and non-zero R_a, until it reaches R_a, then move to the a with next lowest selectivity, and so on. Keep repeating this step until the precision constraint Equation 10 is satisfied.

Thus the approach is to round up the values of the R_a in the order of decreasing s_a, and then round up the values of the values of the E_a in the order of increasing s_a. The intuition is simple; we want to retrieve all the groups where the selectivity is high, in order to meet our recall constraint. Once we know we're retrieving enough groups to meet our recall constraint, we then ensure that we're evaluating enough groups to meet the precision constraint. Naturally, we'd rather evaluate UDFs for the *most incorrect groups among those we're already retrieving* — i.e., those with the lowest s_a first, so that we can have the maximum impact on precision.

The proof that Algorithm GREEDY-LP provides the solution to Linear-Prog. 3.2 uses the following Lemmas, whose proofs can be found in the appendix:

LEMMA 3.7. *There exists an optimal solution where, for every a_1 and a_2 such that $s_{a_1} > s_{a_2}$: either $E_{a_2} = R_{a_2}$ or $E_{a_1} = 0$*

LEMMA 3.8. *There exists an optimal solution where, for every a_1 and a_2 such that $s_{a_1} > s_{a_2}$: either $R_{a_1} = 1$ or $R_{a_2} = 0$*

LEMMA 3.9. *In an optimal solution, either the precision constraint must be satisfied tightly (with equality) or all the Es must be 0. In addition, the recall constraint must be satisfied tightly.*

3.3 Estimated Selectivity Information

In the previous section, we developed probabilistic execution strategies when the selectivity of each group is precisely known in advance. In reality, however, we are unlikely to know the precise selectivity of each group, and must instead rely on *estimates of selectivity* (found using sampling or some other method). In this section, we consider the case where we only have an estimate of the selectivity.

As in the previous section, we focus on probabilistic execution strategies that ensure that the precision and recall constraints are met with satisfaction probability ρ close to 1. Further, as in the previous section, our query processing proceeds in two steps, one, that solves an optimization problem, and second, that actually executes the query using the solution to the optimization problem.

Optimization Problem: Suppose the tuple selectivity for each value $a \in A$ is given by a value s'_a unknown to us. What is known to us instead is an estimate s_a of s'_a, which is an instance of a random variable S_a. Let

$$E[S_a] = s'_a \quad \text{and} \quad \text{Var}(S_a) \approx v_a$$

where v_a is our estimate of the variance of S_a. Note that in this section s_a is an estimate of S_a, and not the actual selectivity like in the previous section.

We are given α, β, ρ, and we want to choose probabilities $1 \geq R_a \geq E_a \geq 0, \forall a \in A$, such that when execution strategy retrieves and evaluates tuples probabilistically as described in Section 3.2, the eventual result satisfies the precision and recall constraints with probability at least ρ, while minimizing cost:

PROBLEM 3 (ESTIMATED-SELECTIVITIES). *Given s_a, v_a, identify $0 \leq E_a \leq R_a \leq 1$ to minimize $\mathcal{C} = \sum_{a \in A} t_a(c_r R_a + c_e E_a)$, such that Constraints 4 and 5 are satisfied with probability greater than ρ.*

3.3.1 Convex Optimization-Based Solution

Until now, we have not specified the correlations between S_as for different as. We solve the problem for two cases, one where the correlations are unknown (and hence we assume the worst case of maximum correlation between S_as) and the other case where S_as for different a's are independent of each other. Our solutions for the two cases are:

CONVEX PROG. 3.10 (UNKNOWN-CORRELATIONS). *Minimize* $\sum_{a \in A} t_a(R_a c_r + E_a c_e)$ *such that*

$$\sum_{a \in A} (1-\alpha) t_a R_a s_a - t_a \alpha (R_a - E_a)(1 - s_a) \geq X$$

$$\sum_{a \in A} t_a R_a s_a - \beta t_a s_a \geq Y$$

$$c_\rho \sum_{a \in A} \sqrt{v_a} t_a (R_a - \alpha E_a) + 0.5\sqrt{t_a} = X$$

$$c_\rho \sum_{a \in A} \sqrt{v_a} t_a |R_a - \beta| + 0.5\sqrt{t_a} = Y$$

$$R_a, E_a \in \{0, 1\}; R_a \geq E_a$$

where $c_\rho = \frac{1}{\sqrt{1-\rho}}$

Notice that the first two constraints and the objective of the problem above are identical to those in Linear-Prog. 3.2. However, the next two constraints (regarding X, Y) are highly non-linear, forcing the problem to be a convex problem. Solving this problem gives a solution which has precision $\geq \alpha$ and recall $\geq \beta$ with probability at least ρ each.

CONVEX PROG. 3.11 (INDEPENDENT GROUPS). *Minimize* $\sum_{a \in A} t_a(R_a c_r + E_a c_e)$ *such that*

$$\sum_{a \in A} (1-\alpha) t_a R_a s_a - t_a \alpha (R_a - E_a)(1 - s_a) \geq X$$

$$\sum_{a \in A} t_a R_a s_a - \beta t_a s_a \geq Y$$

$$c_\rho \sqrt{\sum_{a \in A} t_a^2 v_a (R_a - \alpha E_a)^2 + 0.25 t_a} = X$$

$$c_\rho \sqrt{\sum_{a \in A} t_a^2 v_a (R_a - \beta)^2 + 0.25 t_a} = Y$$

$$R_a, E_a \in \{0, 1\}; R_a \geq E_a$$

where $c_\rho = \frac{1}{\sqrt{1-\rho}}$

Once again, the first two constraints and the objective of the problem above are identical to those in Linear-Prog. 3.2, while the next two constraints make it highly non-linear.

Solving this problem gives a solution which has precision $\geq \alpha$ and recall $\geq \beta$ with probability at least ρ each.

THEOREM 3.12. *Solving Convex-Prog. 3.10 provides a feasible solution for Problem 3 when correlations between S_as are unknown.*

THEOREM 3.13. *Solving Convex-Prog. 3.11 provides a feasible solution for Problem 3 when S_as are independent.*

Both our solutions make use of Chebyshev's inequality, which says that for any random variable X

$$\text{Prob}\left[|X - E[X]| > t\right] \leq \frac{\text{Var}(X)}{t^2}$$

This implies that

$$\text{Prob}\left[|X - E[X]| \leq \frac{\text{Dev}(X)}{\sqrt{1-\rho}}\right] \geq \rho$$

Thus in order to satisfy a constraint $X \geq 0$ with probability ρ, it is sufficient to ensure that $E[X] \geq \frac{\text{Dev}(X)}{\sqrt{1-\rho}}$. We use this to transform the precision and recall constraints 'with probability ρ' into non-probabilistic constraints. The X and Y in Convex-Prog. 3.10 and Convex-Prog. 3.11 are the deviation times $\frac{1}{1-\rho}$ terms, and the lhs in the first two constraints say that expected value is greater than X or Y.

4. JOINT ESTIMATION & EXPLOITATION

In the previous section, we developed probabilistic execution strategies when selectivity estimates are already available. However, in many cases, selectivity estimates are not usually available beforehand, and need to be gathered on-the-fly. In this section, we focus on the problem of jointly estimating and exploiting selectivity information. Our estimation of selectivities will happen using sampling (i.e., retrieving and evaluating a small fraction of the tuples).

Our key idea will be to adapt the technique from Section 3.3 to work with sampling-based estimates of selectivities. We first examine how we can map knowledge from samples to the setting from Section 3.3 in Section 4.1. We then describe our extension of the solution from Section 3.3 to this new scenario, assuming samples are given, in Section 4.2. Lastly, we consider the problem of deciding how much to sample from each group in Section 4.3.

4.1 Sampling-based Estimates

For the purposes of this subsection, we assume that some number of tuples per group have been sampled, i.e., for each of those tuples, we have retrieved and evaluated them. We now describe how the sampled UDF evaluations lead to selectivity estimates.

For each $a \in A$, we use random variable S_a to represent the distribution of selectivity estimates we may obtain for tuples from group a. We assume that the S_as themselves are independent of each other for different as, so evaluating a tuple for $a = a_1 \in A$ tells us nothing about the selectivity S_{a_2}, for $a_2 \neq a_1$.

Suppose for each $a \in A$, we have evaluated the UDF on F_a tuples for that a, and found F_a^+ of them to satisfy the predicate, and F_a^- that don't (thus $F_a = F_a^+ + F_a^-$). Then at that point, the probability density function of our estimate at x will be given by us a Beta distribution [30]: $\text{Beta}(F_a^+ + 1, F_a^- + 1)$ Therefore

$$s_a = E\left[\text{Beta}(F_a^+ + 1, F_a^- + 1)\right] = \frac{F_a^+ + 1}{F_a + 2};$$

$$v_a = \text{Var}(\text{Beta}(F_a^+ + 1, F_a^- + 1)) = \frac{s_a(1 - s_a)}{F_a + 3}$$

where s_a and v_a are defined as per Section 3.3.

4.2 Solution given Sampling-based Estimates

As described above, we can use sampling to obtain selectivity estimates. The estimates can then be used to solve the problem in a way similar to Section 3.3.

However, there is a small wrinkle that needs to be dealt with: in addition to giving us selectivity estimates, the sampled tuples will themselves have been evaluated already. So, among these tuples, those that are correct (based on the UDF invocation on that tuple) can be simply returned as part of the query result without re-evaluating them.

Expressing the mean and variance of selectivities in terms of F_a^+, F_a^- and F_a, and taking into account the tuples that have already been sampled (and hence retrieved and evaluated), we can rephrase the optimization problem from Section 3.3 as below:

CONVEX PROG. 4.1 (SAMPLING SELECTIVITIES). *Minimize* $\sum_{a \in A}(t_a - F_a)(R_a c_r + E_a c_e) + F_a(c_e + c_r)$ *such that*

$$\sum_{a \in A} F_a^+(1 - \alpha) + (1 - \alpha)(t_a - F_a)R_a s_a -$$

$$(t_a - F_a)\alpha(R_a - E_a)(1 - s_a) \geq X$$

$$\sum_{a \in A} F_a^+ + (t_a - F_a)R_a s_a - \beta(t_a - F_a)s_a\beta F_a^+ \geq Y$$

$$c_\rho\sqrt{\sum_{a \in A}(t_a - F_a)^2 v_a(R_a - \alpha E_a)^2 + 0.25(t_a - F_a)} = X$$

$$c_\rho\sqrt{\sum_{a \in A}(t_a - F_a)^2 v_a(R_a - \beta)^2 + 0.25(t_a - F_a)} = Y$$

$$R_a, E_a \in \{0, 1\}; R_a \geq E_a$$

where $c_\rho = \frac{1}{\sqrt{1-\rho}}$

and we have:

THEOREM 4.2. *When S_as are been obtained by sampling F_a tuples from group a for each $a \in A$, of which F_a^+ tuples turn out to be correct (based on UDF evaluations for each of the F_a tuples), solving Convex-Prog. 4.1 provides a feasible solution for Problem 3.*

Note that nothing prevents us from going back-and-forth between estimating selectivities and exploiting them: that is, we may continuously update our selectivity estimates as we evaluate more tuples. We can start off with certain selectivity estimates, apply CONVEXPROG 4.1, decide whether to retrieve and/or evaluate for each group, and as we evaluate more tuples per group, we can go back to Section 4.1, derive new estimates for selectivities, and then apply CONVEXPROG 4.1 again: Thus, our algorithms can be used multiple times in an adaptive fashion.

4.3 Deciding how much to Sample

We now consider the question of how many tuples F_a to sample from each group a. One simple baseline, is to fix a constant c and then sample c tuples from each group. However, since we know the sizes of the different groups, we can use that to significantly reduce overall cost.

We first state a property, and then use that to derive a rule of thumb for how many tuples to sample per group. Let n be the total number of tuples in the table. Then, if the precision and recall thresholds are fixed, it can be shown that a desirable F_a should be

$$O(t_a n^{-\frac{1}{3}})$$

The justification for this statement can be found in the extended technical report [24].

Therefore, our rule of thumb is:

> For a suitably chosen parameter *num*, F_a, the number of tuples sampled from group a for each $a \in A$ should be
>
> $$F_a = num \times t_a n^{-\frac{1}{3}}$$

We can use this rule of thumb in conjunction with the previous sections to decide how much to sample from each group. Naturally, the rule of thumb (as stated) cannot be applied if the value of the parameter *num* is not known to us. As we will see in the experimental section, the rule of thumb is not very sensitive to the value of *num*, and will work even for values of *num* in a fairly large range in a variety of experimental scenarios.

As it turns out (as we will see in the experimental section), the optimal value of *num* is proportional to α, the desired precision threshold. In fact, we find that a value $z\alpha$, with $2 \le z \le 5$ usually works well. But even if it doesn't, we can guess the optimal value of z using adaptive sampling as follows: We start with a small value of $z\alpha$ for *num*, and keep increasing it. We also keep solving the convex optimization problem (CONVEXPROG. 4.1) for each value of *num* and keep estimating the cost of the solution as we increase *num*. The cost will initially fall as *num* increases, and will later start to rise as *num* crosses its optimal value. When cost starts to rise, we can stop further sampling and proceed to solve the problem using our technique from Section 4.2.

5. EXTENSIONS

In this section, we mention some variations of the problem that can be handled using small extensions to the techniques discussed so far. Further details on the extensions and their solution can be found in the appendix of the technical report [24].

- **Alternate Objective Functions** : There may be scenarios where the user has a fixed cost budget, and wishes to maximize the number of tuples returned while ensuring a lower bound on precision. Minor modifications of our techniques can be used to handle this variation.
- **Multiple Predicates** : Another variation of the problem is where the query might have multiple chained UDF selects on a table (which is equivalent to a conjunction of multiple UDF predicates). Since precision and recall constraint are specified by the user for the final output, for this variation it may be possible to trade-off accuracy in one predicate for higher accuracy in the other at the same cost. This problem can be handled by introducing one decision variable per mapping of UDF to decisions. For example, there may be one decision variable which is true if and only if we retrieve for the first predicate and retrieve and evaluate the second predicate.
- **Single Predicate with Join** : Another variation we can consider is where the table T being selected on is later going to be joined with another table T'. Each tuple of the table T may match with a different number of tuples of T', possibly making it worthwhile for us to evaluate a tuple with low correctness-probability that matches with a large number of tuples from T', over a tuple with higher correctness probability that joins with fewer tuples from T'. This variation can also be solved by a minor modification of our techniques.

6. EXPERIMENTS

Our experimental evaluation studies the following three items

- In Experiment 1, we discuss the performance of our query evaluation algorithms versus both baseline algorithms and algorithms that have perfect knowledge. On both our data sets, we show that our algorithm achieves $40 - 80\%$ reduction in cost compared to baseline approaches, and is competitive with (unrealistic) approaches that have perfect knowledge. We also verify that our approaches meet the specified precision and recall constraints. For this evaluation, we assume a fixed sampling procedure.
- In Experiment 2, we discuss the sensitivity of our approach to the choice of sampling procedure and its parameters. We also demonstrate that our method can be used with more sophisticated sampling models, such as a logistic regressor.
- In Experiment 3, we discuss the sensitivity of the performance of our approach to the precision, recall, and probability constraints in the query.

6.1 Experimental Setup

We use two datasets for our experiments both based on loan data.

Lending Club The first dataset, denoted LC, consists of loan data from the peer-to-peer lending website, the Lending Club [1]. The data consists of two csv files, which together contain about 53000 labelled tuples. Each tuple corresponds to a single loan application, and includes attributes like applicant's id, loan amount, term, applicant's grade (a grade assigned by the lending club to reflect borrower quality), employment title, housing status, annual income, purpose of loan. One of the columns is Loan Status, which is the current status of the loan. Values for this column include 'Current', 'Charged Off' (which means that the loan is considered unlikely to be repaid), 'Fully Paid', 'Default' and 'Late'. We assign a value of 'good' to 'Fully Paid' and a value of 'bad' to 'Charged Off', 'Late', and 'Defaulted'.

Prosper The second dataset consists of loan data from the website Prosper [2]. It consists of about 30000 tuples, each tuple corresponds to an individual loan application. It has columns such as amount borrowed, debt to income ratio, and grade. The Loan Status column has values which we classify as 'good' and 'bad'. 'good' values correspond to loans which were paid back on time, while 'bad' ones are for loans which were either paid late or not repaid at all.

Protocol. We then assume that the loan status attribute ('good' vs. 'bad') is the UDF f to be selected on. That is, we consider the query on a table of loan applications, that selects loan applications that will be repaid on time. Overall, we have 53000 tuples from one dataset and 30000 from the other for which the value of the UDF f is known precisely for the purposes of evaluation, but assumed to be unknown to any of the query evaluation algorithms initially. We can then simulate 'sampling' by revealing the value of f for any tuple requested by the query evaluation algorithm. Moreover, we can check the precision and recall obtained by any query evaluation algorithm using our knowledge of f. Unless otherwise specified, we set the default value for the query constraints to be $\alpha = 0.8$ (precision), $\beta = 0.8$ (recall), $\rho = 0.8$ (satisfaction probability). We experimented with different costs for evaluation versus retrieval, but found our results were largely insensitive to these parameters. Hence we set $c_e = 3$ and $c_r = 1$, which implies that evaluating the UDF is a factor of three more expensive than retrieving the tuple.

6.2 Experiment 1 : Performance Comparison

In this experiment, we show that our algorithm has significantly better performance compared to baseline approaches, and is competitive with (unrealistic) approaches that have perfect knowledge. We compare four different query evaluation algorithms:

- Full : This is a baseline algorithm in which one retrieves and evaluates every tuple. Full is the current standard of process-
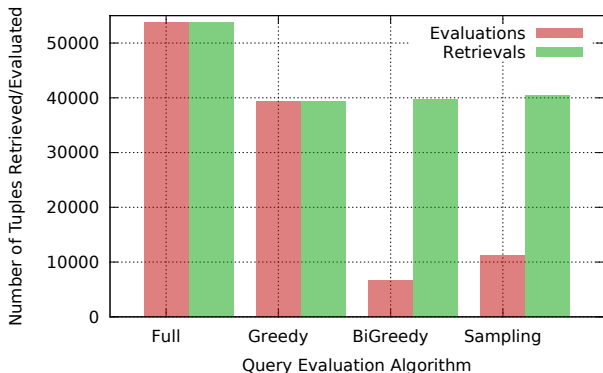
Figure 1: Number of retrievals and evaluates for various Query Evaluation Algorithms

ing with UDFs.

- BiGreedy: Here we use the technique from Section 3.2. The technique is unrealistic as it requires that selectivities be known exactly, so we use our knowledge of all values of f to compute the selectivities and provide them as input to the technique. Thus, this algorithm acts as a cost lower bound to the first algorithm (since it has perfect knowledge of selectivities). We again form groups using grade as the correlated column.

- Greedy: This algorithm also receives perfect selectivities as input (and so is unrealistic). It then greedily retrieves tuples in decreasing order of selectivity until the recall constraint is met in expectation, and evaluates every tuple it retrieves. This algorithm is used as a baseline.

- Sampling: In this algorithm, we form groups based on some correlated column, sample to get selectivity estimates, and then use the technique from Section 3.3 to decide what to retrieve and evaluate. For this experiment, we keep the sample size fixed at 5% of the data, and fix the correlated column to grade.

We include Greedy and BiGreedy as they represent unrealistic baselines that have perfect information. Our method, Sampling, obtains the selectivity information and exploits it. The cost of sampling tuples to estimate the selectivity will be included in the total cost of the Sampling algorithm. Thus, by comparing Sampling with Greedy and BiGreedy, we can understand how effectively Sampling deal with imperfect selectivity information. We present experiments on the LC dataset as the results are similar on the Prosper dataset.

6.2.1 Comparison of Costs

We estimate the average number of tuples retrieved and evaluated by each algorithm over 50 iterations, and plot them in the bar chart in Figure 1. We see that while the number of retrievals is almost the same for all algorithms (except Full), the number of evaluations is far smaller for our two algorithms than for the baselines. Specifically, our sampling based algorithm ends up evaluating around 11000 tuples, which is almost one-fourth of the 39000 tuples evaluated by greedy. The Sampling algorithm is not too much worse than BiGreedy, even though BiGreedy has the unrealistic advantage of knowing exact selectivities beforehand.

6.2.2 Satisfaction of Precision and Recall Constraints

While our algorithms avoid several evaluations, they do so at the cost of slightly lower precision and recall. In this experiment, we

show that the precision and recall guarantees are met by our algorithm. Our algorithms are supposed to guarantee that the precision and recall constraints will be satisfied with probability at least ρ. We now test if this guarantee is being met.

We first fix a value of ρ. For that value of ρ, we execute the Sampling algorithm 100 times (all the way from sampling, to finally retrieving and evaluating and returning tuples), and for each execution, we note whether the precision and recall constraints were satisfied or not. Then we compute the accuracy i.e. the fraction of times the constraints were satisfied for this ρ, and repeat this procedure for different values of ρ. The accuracy is then plotted vs. ρ, in Figures 2(a) (Precision Accuracy) and 2(b) (Recall Accuracy). The $x = y$ line in the figures is the minimum level of accuracy required to satisfy our guarantee. As the figures show, our accuracy is consistently above the line, which means our algorithms do satisfy the precision and recall constraints more than ρ fraction of the time, as desired.

6.3 Experiment 2: Robustness of Estimation

As we saw in the previous section, the Sampling algorithm does not involve any unrealistic assumptions like knowing exact selectivities, and still performs far better than the baselines. However, the Sampling algorithm contains several free parameters, such as the correlated column to be used to estimate selectivities of UDF evaluations, and the number of tuples to be sampled to obtain selectivity estimates. In this section, we will attempt to explore how the parameters may be chosen so as to obtain good results.

Sampling Algorithms . We compare two different sampling algorithms. Each sampling algorithm tells us how many tuples to sample from each group, before finalizing the selectivity estimates. Subsequently, techniques from Section 3.3 (denoted 'Sampling' above) may be used, using selectivity estimates as input. The sampling algorithms are:

- Constant(c): This algorithm samples c tuples from each group.
- Two-Third-Power(*num*): This algorithm uses our result from Section 4 which says that if a table has n tuples, and group a has t_a tuples, then the optimal number of tuples to sample from group a is proportional to $t_a n^{-\frac{1}{3}}$. Hence this algorithm samples $num \, t_a n^{-\frac{1}{3}}$ tuples from group a.

6.3.1 Sampling Schemes and Sample Sizes

We now explore the question of how many tuples to sample per group. When we have formed groups but not sampled any tuples, the only information we have is the sizes of the groups.

The schemes of Section 6.3 tell us the proportion in which to sample tuples from each group. Each scheme has an additional free parameter (c or *num*), which we use to determine the exact number of tuples to sample from each group. We study the average cost of evaluating tuples as a function of this parameter, for each scheme.

Specifically, we first fix the value of the parameter (c or *num*) and use the parameter value to compute the number of tuples to sample from each group. Then we randomly sample and evaluate the corresponding number of tuples from the groups, and form estimates about the selectivity of each group. We then solve our convex optimization problem as in Section 3.3 to decide how many tuples to retrieve and evaluate from each group, and compute the expected number of tuples we needed to evaluate to determine cost (the number of tuples retrieved does not vary much for different sampling schemes or sizes). We do this for 100 iterations, and take the average the number of evaluations across iterations. We find average evaluations for several different values of the parameter, and using different correlated columns, and plot the results. The cor-
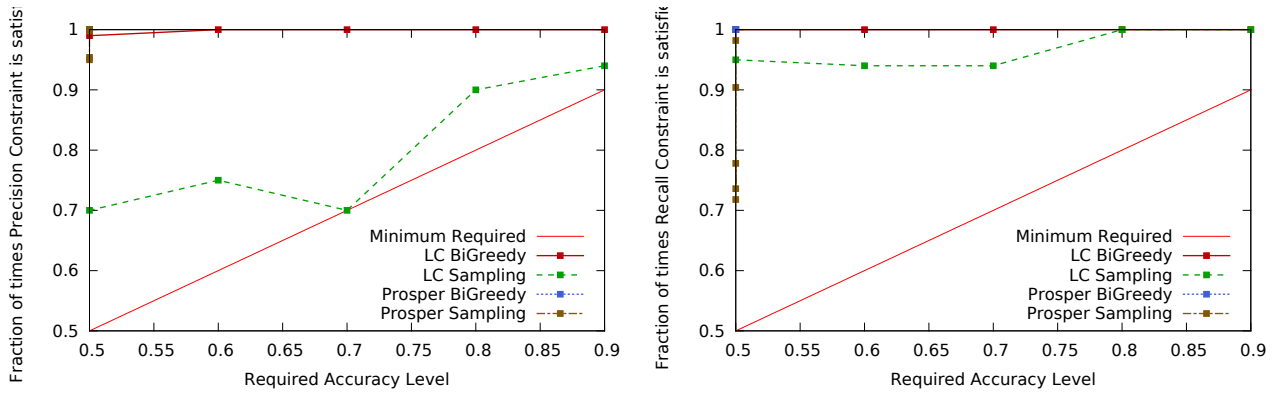
Figure 2: (a) Fraction of times precision constraint was satisfied, for different values of ρ (b) Fraction of times recall constraint was satisfied, for different values of ρ

related columns we use are Loan Term, Housing Status and Grade for LC, and grade for Prosper. The evaluations vs. parameter c for the **Constant** Sampling scheme is plotted in Figure 3(a), while evaluations vs. parameter *num* for the **Two-Third-Power** scheme is in Figure 3(b).

We make three observations about Figures 3(a) and 3(b). The first observation is that sampling too little or too much leads to higher cost; too little because our selectivity estimates are bad, and too much because of the cost of sampling itself. There is a region in between that gives low cost. The second observation is that the **Two-Third-Power** scheme generally leads to lower optimal costs than the **Constant** scheme. For instance, the optimal cost for predictor variable grade on LC is about 10500 for the **Constant** scheme, while it is about 9400 for the **Two-Third-Power** scheme. Similarly, for predictor variable Term on LC, the optimal cost for scheme **Constant** is about 17300, while for scheme **Two-Third-Power**, the optimal cost is about 16100. The third observation is that a value of *num* between 2α and 5α (where α is the precision constraint) leads to near optimal cost for all three predictors on LC as well as grade on Prosper, while for the **Constant** scheme, the optimal number of tuples to sample varies quite a bit for different predictors (We discuss further the dependence of optimal *num* on α in Section 6.4). Thus the **Two-Third-Power** scheme is also robust to small changes in the parameter values, or changes in the predictor or dataset.

While a *num* parameter value between 2α and 5α works fine for the current datasets, it may not necessarily be optimal on other datasets. Thankfully, we do not even need to know the optimal value of the parameter a priori, because we can find the value by running our algorithm adaptively as described in Section 4.

Another thing to note from the results on using different correlated columns is that any one of them is much cheaper than using a naive greedy algorithm that evaluates β fraction of the tuples. Such a greedy algorithm would evaluate about 43000 tuples for LC and about 24000 tuples from Prosper, so using any of the predictors (even the ones whose correlatedness with the predicate doesn't seem obvious a priori) with our method leads to a huge savings in cost.

6.3.2 Using logistic regression to estimate selectivity

In the introduction, we had mentioned that the "correlated column" used to form groups does not have to be a single column from the table. It can be a combination of columns, or even a new virtual column that we create using available data. Specifically, if we have a machine learning algorithm that uses available columns to make a prediction about the UDF f, then we can think of the output of that

algorithm as a new column, and use that column as our correlated column.

In this experiment, we use logistic regression to provide us with such a new column. Specifically, we randomly sample and evaluate UDF f on 1% of the rows of the table (later, we also try this for 2% of the rows). The resulting set of rows is used as training data to learn a logistic regressor from available columns to the predicate f. We only learn the regressor over columns that are either numeric, or nominal with < 50 different values (to avoid overfitting).

After learning the regressor, we apply it to the table to get a probability score for each tuple (The training tuples are included in the table as well, and the cost of evaluating them before training is taken into account in our cost graphs). Then we split the tuples into 10 buckets (groups) based on the probability score. The bucket ranges are chosen so as to get equal sized buckets. For example, if 10% of the training data tuples got a probability score between 0 and 0.55, and the remaining probability scores were uniformly distributed between 0.55 and 1, then we may create one bucket for tuples with score in $[0, 0.55)$, one for $[0.55, 0.6)$ and so on. Then, we proceed as earlier for these buckets, sampling from them to estimate their selectivities. We do not directly use the selectivity estimates output by the logistic regressor, because we do not have guarantees on the correlation between probability scores of different tuples. (Sampling seems to confirm this lack of correlation; for instance, the fraction of tuples having probability score ≈ 0.5 which actually satisfy predicate f is not close to 0.5). Thus we make buckets and treat them as groups instead, sample from them to estimate selectivities, and apply the convex optimization from Section 3.3. We use the **Two-Third-Power** scheme to decide how many tuples to sample per group, and we plot the average number of evaluations across 100 iterations, versus the sampling parameter, in Figure 4. The number of tuples evaluated as shown in the plot includes the tuples evaluated to generate training data for the logistic regressor. Note that in spite of this, the number of evaluations required by our algorithm is far lower than that required by the Greedy algorithm baseline. On LC, we evaluate less than 10000 tuples, while on Prosper, we evaluate about 18000 tuples, while Greedy evaluates about 40000 and 24000 respectively.

The optimal cost in the plot is less than the one we get if we use Grade as a correlated column for LC, while it is a bit more for Prosper. But more importantly, using logistic regression to create the correlated column gives us the advantage of not having to know beforehand which column to use.

6.4 Experiment 3: Query Parameters

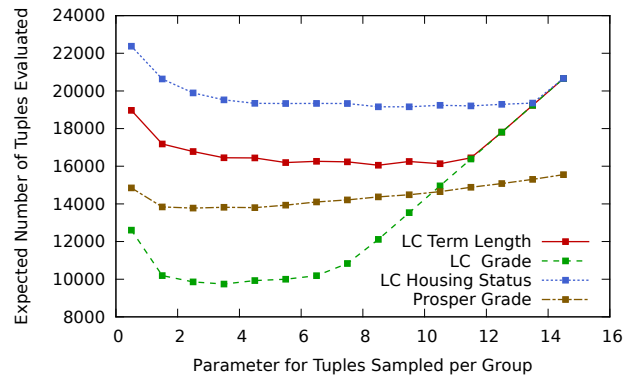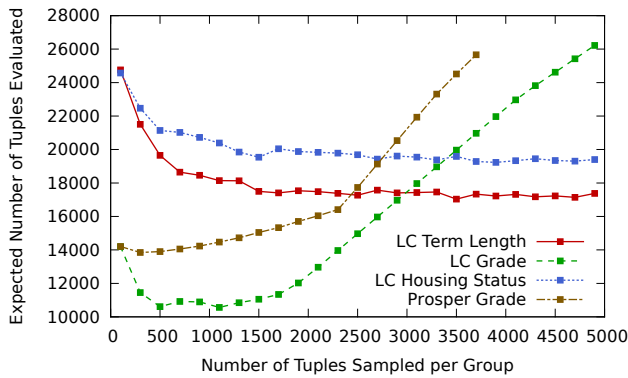In this experiment, we study the effect of precision and recall

*Figure 3: (a) Expected number of tuples evaluated for different levels if sampling per group for the **Constant** sampling scheme (b) Expected number of tuples evaluated for different levels of sampling per group for the **Two-Third-Power** sampling scheme*
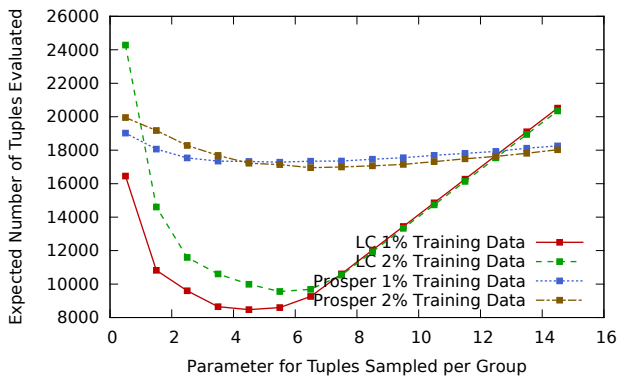

Figure 4: Expected number of tuples evaluated for different levels of sampling per group

constraints α and β on the cost of our algorithm. We demonstrate why some approximation can result in a large cost saving.

We consider LC with Grade as the fixed predictor, we vary α and β from 0.2 to 0.9 each, and we plot average retrieval and evaluation costs over 50 iterations for each combination of α and β. We keep ρ fixed at 0.5 and use the **Two-Third-Power** sampling scheme for this experiment. The results are similar for other predictors, other ρ values, and the other dataset. Figure 5(a) compares the number of evaluations vs α for $\beta = 0.8$, Figure 5(b) compares the number of evaluations vs. β for $\alpha = 0.8$, and Figure 5(c) compares the number of retrieves vs β for $\alpha = 0.8$.

When α and β are equal to 1 (100%), we must necessarily retrieve and evaluate every tuple, whereas when they're 0, we don't need to retrieve or evaluate any tuple. But when they are both 0.8 we don't need to evaluate 80% of the tuples, but much fewer. The figures show that cost as a function of either α or β looks like a convex function. This tells us why approximating results can be a performance win: a small loss in accuracy can lead to a much larger saving in cost.

As we vary α and β, we observe that the optimal value of the *num* parameter is approximately proportional to α. So we set the value of *num* to 2.5α, 3.5α, and so on for the plots. The plots show that $num = 2.5\alpha$ performs slightly better than other values of *num* consistently (for all values of α), which reflects our observation that the optimal value of *num* is proportional to α. Number of retrieves vs α is not plotted because the two are nearly independent, with number of retrieves depending almost entirely on β and ρ.

7. RELATED WORK

The prior work related to our paper can be divided into a few categories. We describe each one in turn below:

Sensor Networks: There has been a lot of work on data processing applied on sensor networks [6]. We could regard taking additional readings for a sensor as an expensive operation (which only must be done as needed, for example, to answer a user query), much like the expensive UDFs we consider in this paper. There has been some work on query processing for sensor networks while minimizing the amount of data acquired (i.e., readings taken) [27]. Further, there has been work on using correlated attributes to avoid acquiring data [13], which is very similar to our setting. However, the underlying scenarios and assumptions are very different, necessitating different techniques.

Approximate Query Processing: Over the past decade, there has been a lot of work on approximate query processing; as examples, see [10, 16, 18, 23]. Garofalakis et al. [15] provides a good survey of the area. There are multiple systems that support approximate query processing, including BlinkDB [5] and Aqua [4], allowing users to trade-off accuracy for performance. Often, these schemes achieved the desired level of accuracy by running the query on a sample of the database. There has also been work on deciding how to choose appropriate samples for query processing [3, 8]. All of these systems and algorithmic papers focus on a notion of approximation that is slightly different from the one used in our paper: in particular, these papers only approximate numerical aggregate quantities, focusing on operators such as SUM, AVG, VAR, as well as frequency moments [7]. On the other hand, our paper focuses on approximating selection (i.e., set-valued) queries, as a result, our work is complementary.

Reference [22] uses histograms to approximate set-valued queries, but their notion of approximation involves approximating values within tuples, rather than approximating the set of tuples themselves, within a certain desired precision and recall.

Query Processing with UDFs: UDFs occur often in database queries. The work on optimizing queries with UDFs has focused primarily on identifying the best placement of UDFs within query plans [12, 20], and not on using correlations to avoid UDF evaluations in the first place.

Recently, crowd-powered database systems, such as CrowdDB [14] and Deco [29] have been treating calls to human workers for data processing and data gathering on-the-fly, as expensive UDFs that should be used sparingly. Our optimization techniques could easily be applied to these systems.
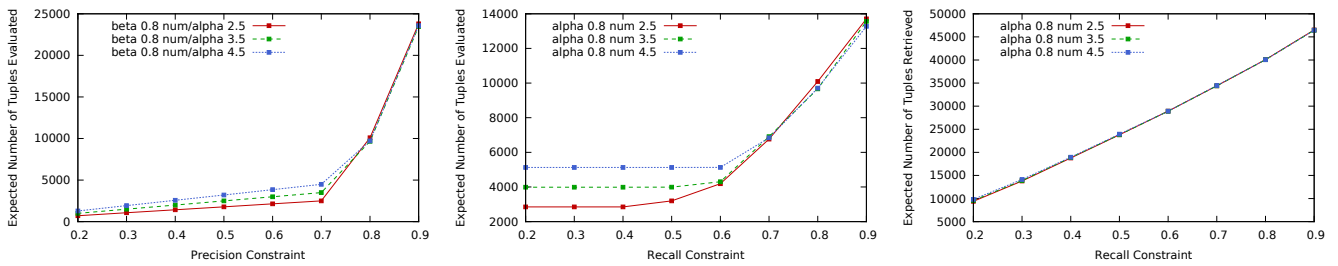
Figure 5: (a) Expected number of tuples evaluated for different levels of alpha (b) Expected number of tuples evaluated for different levels of beta (c) Expected number of tuples retrieved for different levels of beta

Correlation-aware Query Processing: There has been some recent work on finding correlations between attributes of a relation [9, 21, 25] — these papers could be used to identify the correlated attributes to bootstrap our techniques for estimating and exploiting correlations. Correlations have also been used for traditional query optimization, allowing for more accurate estimates of intermediate result sizes, and therefore of query-plan costs. Recent work has also looked at appropriately choosing which indexes and materialized views to maintain taking correlations into account [26]. While these papers mainly focus on correlations between attributes within a relation (and therefore there is no cost to acquiring data since it is already provided), our paper exploits correlations between attributes in a relation and an external UDF.

8. CONCLUSIONS

In this paper, we introduced techniques for efficiently approximating select queries with expensive UDFs. We found that even though the basic problem for even the simplest scenario is NP-Hard, there are efficient techniques that allow us to achieve near-optimality. Our techniques apply to a variety of different scenarios, based on what information is available to us, and allow us to reduce cost while satisfying precision and recall constraints specified by the user. As UDFs become increasingly ubiquitous and datasets get larger, the cost savings can become quite significant. Moreover, our methods integrate well with machine learning techniques such as logistic regression, allowing us to leverage these techniques to reduce cost even further. Our techniques can also be generalized to other classes of queries, such as selects followed by joins, or multiple selection conditions. Finally, our experiments on real data show that the cost savings we obtain are quite large relative to the small loss in accuracy. As future work, we plan to explore generalizing our techniques to complete select-from-where queries, as well as queries involving nesting and aggregation.

9. REFERENCES

[1] http://www.lendingclub.com.
[2] http://www.prosper.com.
[3] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 487–498, New York, NY, USA, 2000. ACM.
[4] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 574–576, New York, NY, USA, 1999. ACM.
[5] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42, 2013.
[6] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.
[7] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996.
[8] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 539–550, New York, NY, USA, 2003. ACM.
[9] P. G. Brown and P. J. Hass. Bhunt: Automatic discovery of fuzzy algebraic constraints in relational data. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 668–679. VLDB Endowment, 2003.
[10] K. Chakrabarti et al. Approximate query processing using wavelets. In *VLDB*, pages 111–122, 2000.
[11] S. Chaudhuri and K. Shim. Query optimization in the presence of foreign functions. In *VLDB*, pages 529–542, 1993.
[12] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. *ACM Trans. Database Syst.*, 24(2):177–228, 1999.
[13] A. Deshpande, C. Guestrin, W. Hong, and S. Madden. Exploiting correlated attributes in acquisitional query processing. In *ICDE*, pages 143–154, 2005.
[14] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, 2011.
[15] M. N. Garofalakis and P. B. Gibbon. Approximate query processing: Taming the terabytes. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 725–, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
[16] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, pages 541–550, 2001.
[17] J. M. Hellerstein. Practical predicate placement. In *SIGMOD Conference*, pages 325–335, 1994.
[18] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In J. Peckham, editor, *SIGMOD 1997*, pages 171–182. ACM Press, 1997.
[19] J. M. Hellerstein and M. Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *SIGMOD Conference*, pages 267–276, 1993.
[20] J. M. Hellerstein and M. Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 267–276, New York, NY, USA, 1993. ACM.
[21] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 647–658, New York, NY, USA, 2004. ACM.
[22] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 174–185, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
[23] C. Jermaine et al. Scalable approximate query processing with the dbo engine. *ACM Trans. Database Syst.*, 33(4), 2008.
[24] M. R. Joglekar, H. Garcia-Molina, A. Parameswaran, and C. Re. Optimally exploiting correlations for udf evaluation. Technical report, 2014.
[25] H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik. Correlation maps: A compressed access method for exploiting soft functional dependencies. *Proc. VLDB Endow.*, 2(1):1222–1233, Aug. 2009.
[26] H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik. Coradd: Correlation aware database designer for materialized views and indexes. *Proc. VLDB Endow.*, 3(1-2):1103–1113, Sept. 2010.
[27] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 491–502, New York, NY, USA, 2003. ACM.
[28] A. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: Declarative crowdsourcing. Technical report, Stanford University, 2012.
[29] H. Park, H. Garcia-Molina, R. Pang, N. Polyzotis, A. Parameswaran, and J. Widom. Deco: A system for declarative crowdsourcing. *Proc. VLDB Endow.*, 5(12):1990–1993, Aug. 2012.
[30] L. Wasserman. *All of Statistics*. Springer, 2003.

10. APPENDIX

10.1 Correctness Proof for our solution to Perfect Selectivities Case

R_a^+, R_a^-, E_a^+, E_a^-, C_a and W_a are all binomial random variables, since they're a sum of Bernoulli random variables. Moreover, since each tuple with $A = a$ is retrieved randomly with probability R_a independently of other tuples, and correct with probability s_a independently of other tuples, we must have

$$E\left[R_a^+\right] = t_a s_a R_a$$

$$Var(R_a^+) = t_a s_a R_a (1 - s_a R_a)$$

and

$$E\left[R_a^-\right] = t_a(1 - s_a)R_a$$

$$Var(R_a^-) = t_a(1 - s_a)R_a(1 - (1 - s_a)R_a)$$

Similarly,

$$E\left[E_a^+\right] = t_a s_a E_a$$

$$Var(E_a^+) = t_a s_a E_a(1 - s_a E_a)$$

and

$$E\left[E_a^-\right] = t_a(1 - s_a)E_a$$

$$Var(E_a^-) = t_a(1 - s_a)E_a(1 - (1 - s_a)E_a)$$

.

Thus for the precision, we have:

$$E\left[\sum_{a \in A} R_a^+ - \alpha \sum_{a \in A}(R_a^+ + R_a^- - E_a^-)\right]$$
$$= \sum_{a \in A} t_a s_a R_a (1 - \alpha) + \alpha t_a (1 - s_a)(E_a - R_a)$$

For recall, we have,

$$E\left[\sum_{a \in A} R_a^+ - \beta \sum_{a \in A} C_a\right] = \sum_{a \in A} t_a s_a R_a - \beta t_a s_a$$

To convert the probabilistic satisfaction constraints for precision/recall, into an absolute constraint in terms of the R_as and E_as, we need the general form of Hoeffding's inequality:

Hoeffding's inequality: Let X_1, X_2, ... X_n be independent random variables, which are almost surely bounded i.e.

$$Pr(a_i \leq X_i \leq b_i) = 1$$

Let there sum be

$$S = \sum_{i=1}^{n} X_i$$

Then:

$$Pr(S - E\left[S\right] \leq -t) \leq e^{-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}}$$

$$Pr(S - E\left[S\right] \geq t) \leq e^{-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}}$$

□

For a tuple t in the table that satisfies the predicate, consider the variable Ip_t which takes value:

1. $1 - \alpha$ if the tuple is correct and retrieved

2. $-\alpha$ if the tuple is incorrect and retrieved but not evaluated

3. 0 otherwise

Then the lhs of the precision constraint, is the sum of Ip_t over all t in the table.

Similarly, for every tuple t let Ir_t be an indicator variable which takes value:

1. $1 - \beta$ if the tuple is correct and retrieved

2. 0 otherwise

Then the lhs of the recall constraint is the sum of Ir_t over all t.

All the Ip variables are independent of each other, and all the Ir variables are independent of each other. Moreover, the variables are all bounded in the following ranges:

1. Ip in $[-\alpha, 1 - \alpha]$

2. Ir in $[0, 1 - \beta]$

Now we can apply Hoeffding's inequality to get a concentration bound on the lhs of the precision and recall constraints. Specifically,

$$Pr(|(\sum_{a \in A} R_a^+ - \alpha \sum_{a \in A}(R_a^+ + R_a^- - E_a^-)) -$$
$$(\sum_{a \in A} t_a s_a (1 - \alpha)R_a + t_a(1 - s_a)\alpha(E_a - R_a))|$$
$$\geq t) \leq e^{-\frac{2t^2}{\sum_{a \in A} t_a}}$$

and

$$Pr(|(\sum_{a \in A} R_a^+ - \beta C_a) - (\sum_{a \in A} t_a s_a R_a - \beta t_a s_a)| \geq t) \leq e^{-\frac{2t^2}{\sum_{a \in A} t_a(1 - \beta)}}$$

We have

$$T_\rho^p = \sqrt{\frac{log(1 - \rho)\sum_{a \in A} t_a}{2}}$$

and

$$T_\rho^r = \sqrt{\frac{log(1 - \rho)\sum_{a \in A} t_a(1 - \beta)}{2}}$$

Then with probability $\geq \rho$ the lhs of the precision constraint will be within T_ρ^p of it's expectation, and the lhs of the recall constraint will be within T_ρ^r of its expectation.

Our R_as and E_as satisfy

$$\sum_{a \in A} t_a s_a (1 - \alpha)R_a + t_a(1 - s_a)\alpha(E_a - R_a) \geq T_\rho^p$$

So with probability $\geq \rho$, the value of $\sum_{a \in A} R_a^+ - \alpha \sum_{a \in A}(R_a^+ + R_a^- - E_a^-)$ will be greater than it's expectation minus T_ρ^p, that is, $\sum_{a \in A} t_a s_a (1 - \alpha)R_a + t_a(1 - s_a)\alpha(E_a - R_a) - T_\rho^p$ which is greater than 0.

Our R_as and E_as also satisfy

$$\sum_{a \in A} t_a s_a R_a - \sum_{a \in A} \beta t_a s_a \geq T_\rho^r$$

So with probability $\geq \rho$, the value of $\sum_{a \in A} R_a^+ - \beta C_a$ will be greater than it's expectation minus T_ρ^r, i.e. $(\sum_{a \in A} t_a s_a R_a - \beta t_a s_a) - T_\rho^r$ which is greater than 0.

Thus both precision and recall constraints are satisfied with probability ρ.

10.2 Proof of Tightness Bound for solution to Perfect Selectivities Case

Suppose an optimal solution (where precision and recall constraints are satisfied with probability ρ and expected cost is minimised) is given by R_a^o, E_a^o for each $a \in A$. The expected cost for this solution is $\sum_{a \in A} t_a(R_a^o c_r + E_a^o c_e)$. Consider the expected values of lhs of their precision and recall constraints. For precision, it is given by:

$$P^o = \sum_{a \in A} t_a s_a (1 - \alpha) R_a^o + t_a (1 - s_a) \alpha (E_a^o - R_a^o)$$

and for recall, it is given by:

$$R^o = \sum_{a \in A} t_a s_a R_a^o - \beta t_a s_a$$

Let the solution produced by our algorithm by given by R_a', E_a' for each $a \in A$. And let:

$$P' = \sum_{a \in A} t_a s_a (1 - \alpha) R_a' + t_a (1 - s_a) \alpha (E_a' - R_a')$$

and for recall, it is given by:

$$R' = \sum_{a \in A} t_a s_a R_a' - \beta t_a s_a$$

Hence $P' \geq T_\rho^p$, $R' \geq T_\rho^r$

Moreover, we must have $P^o \geq -T_{1-\rho}^p$, since if the expected value of the lhs of precision constraint for the optimal solution is less than $-T_{1-\rho}^p$, then by the chernoff bound, the lhs of the precision constraint will be ≤ 0 with probability $\geq 1 - \rho$, and hence the precision constraint will be satisfied with probability $\leq \rho$. For a similar reason, we must have $R^o \geq -T_{1-\rho}^r$.

We will now show two tightness bounds, one in terms of the cost, and one in terms of the precision and recall constraints.

First we will find a bound in terms of expected cost $\sum_{a \in A} (t_a s_a + t_a(1 - s_a))(R_a c_r + E_a c_e)$. We know that

$$\sum_{a \in A} R_a^o t_a s_a - \sum_{a \in A} \beta t_a s_a \geq -T_{1-\rho}^r$$

and

$$\sum_{a \in A} (1 - \alpha) t_a s_a R_a^o + t_a (1 - s_a) \alpha (E_a^o - R_a^o) \geq -T_{1-\rho}^p$$

. We will increase some values of R_a, and increase the corresponding values of E_a by the same amount, so as to get the resulting R_a and E_a values to satisfy, $\sum_{a \in A} R_a^{o2} t_a s_a \geq T_\rho^r + \sum_{a \in A} \beta t_a s_a$ and $\sum_{a \in A} (1 - \alpha) t_a s_a R_a^o + t_a (1 - s_a) \alpha (E_a^o - R_a^o) geq T_\rho^p$. We choose which R_as value to raise, in a greedy manner, as follows: We choose the a with highest value of selectivity ($\frac{t_a s_a}{t_a s_a + t_a(1 - s_a)}$) for which $R_a < 1$, and keep increasing that R_a (and E_a) till R_a becomes 1 (or both the constraints are satisfied), then move on to the next best a, and so on. When a R_a and E_a are increased by d, the value of $\sum_{a \in A} R_a t_a s_a$ goes up be $dt_a s_a$, the value of $\sum_{a \in A} (1 - \alpha) t_a s_a R_a + t_a (1 - s_a) \alpha (E_a - R_a)$ goes up by $d(1 - \alpha) t_a s_a$ while cost goes up by $(c_r + c_e) d(t_a s_a + t_a(1 - s_a))$. If we keep raising the values till both constraints $\sum_{a \in A} R_a^{o2} t_a s_a \geq T_\rho^r + \sum_{a \in A} \beta t_a s_a$ and $\sum_{a \in A} (1 - \alpha) t_a s_a R_a^o + t_a (1 - s_a) \alpha (E_a^o - R_a^o) geq T_\rho^p$ are satisfied, then the cost increase will be at most

$$(c_e + c_r) \frac{1}{s_a} Max(T_\rho^r + T_{1-\rho}^r, \frac{T_\rho^p + T_{1-\rho}^p}{1 - \alpha})$$

where s_a is the smallest value of selectivity from among the as whose R_a, E_a we raised. Now since the resulting R_a and E_a values satisfy these constraints, the total cost $\sum_{a \in A} (t_a s_a + t_a(1 -$

$s_a))(R_a c_r + E_a c_e)$ has to be more than the cost of our solution (R_a', E_a') (which was optimal for the linear problem with the given constraints). Thus the difference between the cost of our solution, and the cost of the optimal solution to the original problem, is at most

$$(c_e + c_r) \frac{1}{s_a} Max(T_\rho^r + T_{1-\rho}^r, \frac{T_\rho^p + T_{1-\rho}^p}{1 - \alpha})$$

. This is our tightness bound on cost. If table size is n, and the minimum s_a is constant as n grows, the cost difference is $O(\sqrt{n})$ because the T^p, T^r values are $O(\sqrt{n})$. Since the cost of the solution is expected to be linear in n, an extra cost of $O(\sqrt{n})$ should be a small fraction of the actual cost.

For the precision/recall tightness bound, let Let $\beta^\star = \beta + \frac{T_\rho^r + T_{1-\rho}^r}{\sum_{a \in A} t_a s_a}$. And let $\alpha^\star = \alpha + \frac{T_\rho^p + T_{1-\rho}^p}{\sum_{a \in A} \beta^\star t_a s_a}$ And let the optimal solution for precision $\alpha^\star$ (with probability ρ) and recall $\beta^\star$ (with probability ρ) be $R_a^\star$, $E_a^\star$ for each $a \in A$. Then that optimal solution must satisfy:

$$\sum_{a \in A} (1 - \alpha^\star) t_a s_a R_a + \alpha^\star t_a (1 - s_a)(E_a - R_a) \geq -T_{1-\rho}^p$$

and

$$\sum_{a \in A} R_a t_a s_a \geq -T_{1-\rho}^r + \sum_{a \in A} \beta^\star t_a s_a$$

which implies (because of our choice of $\alpha^\star$ and $\beta^\star$)

$$\sum_{a \in A} (1 - \alpha) t_a s_a R_a + \alpha t_a (1 - s_a)(E_a - R_a) \geq T_\rho^p$$

and

$$\sum_{a \in A} R_a t_a s_a \geq T_\rho^r + \sum_{a \in A} \beta t_a s_a$$

Hence the optimal solution for $\alpha^\star$, $\beta^\star$ satisfies the constraints of the linear problem we used to find R_a', E_a'. Since we had the optimal solution to our linear problem, it's cost is necessarily less than the cost of $R_a^\star$, $E_a^\star$. Thus our solution has cost less than the optimal cost for $\alpha^\star$ and β^{star}. Note that for table size n, the T^p, T^r terms grow as $O(\sqrt{n})$, while the $\sum_{a \in A} t_a s_a$ would probably be linear in n. Thus the difference between β and $\beta^\star$, and the difference between α and $\alpha^\star$ are in $O(\frac{1}{\sqrt{n}})$.

10.3 Proof for Greedy solution to the linear problem

For purposes of this proof, let's use c_{a_i} to denote $t_{a_i} s_{a_i}$ and w_{a_i} to denote $t_{a_i}(1 - s_{a_i})$ for all i.

Lemma: Suppose we have a_1 and a_2 such that $s_{a_1} > s_{a_2}$. Then in an optimal solution, one of the following must be true:

1. $E_{a_2} = R_{a_2}$
2. $E_{a_1} = 0$

Proof: Suppose $E_{a_2} < R_{a_2}$ and $E_{a_1} > 0$. Suppose we replace E_{a_2} by $E_{a_2} + \epsilon$, and E_{a_1} by $E_{a_1} - \epsilon \frac{w_{a_2}}{w_{a_1}}$ for an epsilon small enough so as to not violate the $0 \leq E_a \leq R_a$ bound for either a_1 or a_2. Then the recall constraint and rhs of precision constraint aren't affected at all, while lhs of precision constraint is increased by $\epsilon \alpha (w_{a_2} - \delta w_{a_1}) = 0$. Thus both the constraints continue to be satisfied as before. On the other hand, cost increases by $c_e \epsilon (w_{a_2} + c_{a_2} - \frac{(w_{a_2})(w_{a_1} + c_{a_1})}{w_{a_1}}) = c_e \epsilon (c_{a_2} - \frac{c_{a_1} w_{a_2}}{w_{a_1}}) < 0$. Thus this transformation keeps the constraints satisfied and reduces cost, contradicting the optimality of the solution. Hence, at least one the the conditions

1. $E_{a_2} = R_{a_2}$
2. $E_{a_1} = 0$

must be satisfied.

This lemma shows why once the Rs are fixed, the Es must be filled in increasing selectivity order. According to the lemma, E for at most one selectivity value can be strictly between 0 and R, while Es must be zero for all higher selectivities, and R for all lower selectivities.

Lemma: Suppose we have a_1 and a_2 such that $s_{a_1} > s_{a_2}$. Then in an optimal solution, one of the following must be true:

1. $R_{a_1} = 1$
2. $R_{a_2} = 0$

Proof: Suppose $R_{a_1} < 1$ and $R_{a_2} > 0$. We consider two cases:

1. **Case 1:** $E_{a_2} = R_{a_2}$. Reduce R_{a_2}, E_{a_2} by ϵ, and raise R_{a_1}, E_{a_1} by $\epsilon \frac{c_{a_2}}{c_{a_1}}$. This keeps the rhs and lhs values of both the constraints unchanged, but reduces cost by $(w_{a_2} + c_{a_2})(c_r + c_e)\epsilon - (w_{a_1} + c_{a_1})(c_r + c_e)\epsilon \frac{c_{a_2}}{c_{a_1}} = \epsilon(c_r + c_e)(w_{a_2} - w_{a_1}\frac{c_{a_2}}{c_{a_1}}) > 0$. This contradicts the optimality of the solution, proving the lemma in case 1.

2. **Case 2:** $E_{a_2} < R_{a_2}$. Let $l = max(\frac{c_{a_2}}{c_{a_1}}, \frac{(1-\alpha)c_{a_2} - \alpha w_{a_2}}{(1-\alpha)c_{a_1} - \alpha w_{a_1}})$ Reduce R_{a_2} by ϵ, and raise R_{a_1} by $l\epsilon$. The rhs of both constraints remain the same, while the lhs either remain the same or increase. Hence the constraints continue to be satisfied. The cost decreases by $\epsilon c_r(w_{a_2} + c_{a_2} - lw_{a_1} - lc_{a_1})$. If $l = \frac{c_{a_2}}{c_{a_1}}$, then cost decreases as shown in case 1. If $l = \frac{(1-\alpha)c_{a_2} - \alpha w_{a_2}}{(1-\alpha)c_{a_1} - \alpha w_{a_1}}$, then cost decreases by

$$\epsilon c_r \frac{1}{(1-\alpha)c_{a_1} - \alpha w_{a_1}}$$
$$((w_{a_2} + c_{a_2})((1-\alpha)c_{a_1} - \alpha w_{a_1}) - (w_{a_1} + c_{a_1})((1-\alpha)c_{a_2} - \alpha w_{a_2}))$$
$$= \epsilon c_r \frac{c_{a_1}w_{a_2} - c_{a_2}w_{a_1}}{(1-\alpha)c_{a_1} - \alpha w_{a_1}}$$
$$> 0$$

This contradicts the optimality of the solution, thus proving the lemma.

The second lemma means that there is at most one selectivity for which $0 < R < 1$, and R is 0 for all lower selectivities and 1 for all higher selectivities.

Lemma: In an optimal solution, either the precision constraint must be satisfied tightly (with equality) or all the Es must be 0. In addition, the recall constraint must be satisfied tightly.

Proof: Suppose the precision constraint is not tight in our solution, and at least one E is > 0. Then we can decrease that E slightly, while still satisfying the precision and recall constraints, and reducing cost. This proves the first part of the lemma.

We next prove that the recall constraint must also be tight. Let a_r be the lowest selectivity a with $R > 0$. Suppose the recall constraint isn't tight, suppose all Es are 0. If the precision-constraint-coefficient $(1-\alpha)c_{a_r} - \alpha w_{a_r} < 0$, then decreasing R_{a_r} slightly will continue to satisfy the precision and recall constraints and reduce cost, which is not possible. So recall constraint must be tight if all Es are 0.

That leaves the case where not all Es are zero (so precision constraint is tight) and recall constraint isn't tight. Let a_r be the lowest selectivity a with $R > 0$ and let a_e be the highest selectivity a with $E < R$. Because of condition $X < \sum_{a \in A} max((1 - $

$\alpha)t_a s_a - \alpha t_a(1 - s_a), 0)$, the selectivity of a_e must be $<= \alpha$. For a small ϵ, decreasing R_{a_r} and E_{a_r} by ϵ and increasing E_{a_e} by $\epsilon \frac{(1-\alpha)c_{a_r}}{\alpha w_{a_e}}$ will continue to satisfy the precision and recall constraints. Cost will reduce by $\epsilon c_r(c_{a_r} + w_{a_r}) + \epsilon c_e(w_{a_r} + c_{a_r} - \frac{(w_{a_e} + c_{a_e})(1-\alpha)c_{a_r}}{\alpha w_{a_e}})$. Clearly, $\epsilon c_r(c_{a_r} + w_{a_r}) > 0$. And $(\alpha w_{a_e})(w_{a_r} + c_{a_r}) - (1 - \alpha)c_{a_r}(w_{a_e} + c_{a_e})$ $\geq (w_{a_e} + c_{a_e})(w_{a_r} + c_{a_r})(\alpha(1-\alpha) - (1-\alpha)\alpha) = 0$. Thus the transformation keeps constraints satisfied and reduces cost, which contradicts the optimality of the solution, proving the lemma.

Thus the recall constraint must be tight. This, along with the lemma on greedy assignment of Rs, proves that increasing Rs in decreasing order of selectivity, until the recall constraint is satisfied, is optimal. After that, the greedy assignment of Es lemma shows that increasing Es in order of increasing selectivity gives the optimal solution.

10.4 Correctness Proof for our solution to the Estimated Selectivities Case

As before, we have for each $a \in A$:

1. R_a^+ is the number of correct tuples ($f(B) = 1$) that we end up retrieving for $A = a$.

2. R_a^- is the number of incorrect tuples that we end up retrieving for $A = a$.

3. E_a^+ is the number of correct tuples we end up evaluating (and hence accepting) for $A = a$.

4. E_a^- is the number of incorrect tuples we end up evaluating (and hence rejecting) for $A = a$.

In addition, we have

1. C_a is the number of tuples satisfying the predicate for $A = a$.

2. W_a is the number of tuples not satisfying the predicate for $A = a$.

Then we must satisfy

$$\sum_{a \in A} R_a^+ - \alpha \sum_{a \in A}(R_a^+ + R_a^- - E_a^-) \geq 0$$

must be satisfied with probability ρ, and

$$\sum_{a \in A} R_a^+ - \beta C_a \geq 0$$

must be satisfied with probability ρ.

Let

$$P = \sum_{a \in A} R_a^+ - \alpha \sum_{a \in A}(R_a^+ + R_a^- - E_a^-)$$

and let

$$R = \sum_{a \in A} R_a^+ - \beta C_a$$

To satisfy the constraints $P \geq 0$ and $R \geq 0$ with probability ρ each, we want to satisfy

$$E(P) \geq c_\rho \text{Dev}(P)$$

and

$$E(R) \geq c_\rho \text{Dev}(R)$$

for an appropriate c_ρ determined by Chebyshev's inequality. Thus we want to find the expectation and variances of P and R in terms of R_a, E_a, and fixed quantities.

For a random variable U (from among R_a^+, R_a^-, E_a^+, E_a^-, C_a, W_a), and a tuple number i, let $I_{U,i}$ denote the indicator variable

that is 1 if the ith tuple adds one to U, and 0 otherwise. For example, $I_{R_a^+,2}$ is 1 if tuple number 2 for $A = a$ is retrieved and satisfies the predicate, and 0 otherwise. Thus $U = \sum_{i=1}^{t_a} I_{U,i}$. The I's are all Bernoulli random variables. Both P and R can be expressed as weighted sums of these Bernoulli random variables. Thus to find expectation and variance of P and R, we first try to find the expectation, variance, and covariances of the I variables. Each tuple with $A = a$ is retrieved with probability R_a, and evaluated with probability E_a. Moreover, it satisfies the predicate with probability S_a. Thus this tuple contributes 1 to R_a^+ with probability $R_a S_a$. So for each i, $I_{R_a^+,i}$ is a Bernoulli random variable with

$$E\left[I_{R_a^+,i}\right] = R_a S_a$$

. Similarly,

$$E\left[I_{R_a^- - E_a^-,i}\right] = (R_a - E_a)(1 - S_a)$$

$$E\left[I_{C_a,i}\right] = S_a$$

For all U, i,

$$Var(I_{U,i}) = E\left[I_{U,i}\right]\left(1 - E\left[I_{U,i}\right]\right)$$

For each U, i, $Var(I_{U,i} = E\left[I_{U,i}\right]\left(1 - E\left[I_{U,i}\right]\right)$

To find covariances, we note that

$$E\left[S_a^2\right] = E\left[(1 - S_a)^2\right] = s_a^2 + v_a$$

and

$$E\left[(1 - S_a)S_a\right] = s_a(1 - s_a) - v_a$$

Thus for $i \neq j$

$$
\begin{aligned}
Cov(I_{R_a^+,i}, I_{R_a^+,j}) &= E\left[I_{R_a^+,i}I_{R_a^+,j}\right] - E\left[I_{R_a^+,i}\right]E\left[I_{R_a^+,j}\right] \\
&= E\left[R_a S_a R_a S_a\right] - R_a s_a R_a s_a \\
&= R_a^2(v_a + s_a^2) - R_a^2 s_a^2 \\
&= R_a^2 v_a
\end{aligned}
$$

Similarly,

$$Cov(I_{R_a^- - E_a^-,i}, I_{R_a^- - E_a^-,j}) = (R_a - E_a)^2 v_a$$

and

$$Cov(I_{C_a,i}, I_{C_a,j}) = v_a$$

Moreover,

$$
\begin{aligned}
Cov(I_{R_a^+,i}, I_{R_a^- - E_a^-,j}) &= E\left[I_{R_a^+,i}I_{R_a^- - E_a^-,j}\right] - E\left[I_{R_a^+,i}\right]E\left[I_{R_a^- - E_a^-,j}\right] \\
&= E\left[R_a S_a(R_a - E_a)(1 - S_a)\right] \\
&\quad - R_a s_a(R_a - E_a)(1 - s_a) \\
&= R_a(R_a - E_a)(s_a(1 - s_a) - v_a) \\
&\quad - R_a(R_a - E_a)s_a(1 - s_a) \\
&= - R_a(R_a - E_a)v_a
\end{aligned}
$$

and similarly,

$$Cov(I_{R_a^+,i}, I_{C_a,j}) = R_a v_a$$

Finally,

$$
\begin{aligned}
Cov(I_{R_a^+,i}, I_{R_a^- - E_a^-,i}) &= E\left[I_{R_a^+,i}I_{R_a^- - E_a^-,i}\right] - E\left[I_{R_a^+,i}\right]E\left[I_{R_a^- - E_a^-,i}\right] \\
&= E\left[0\right] - R_a s_a(R_a - E_a)(1 - s_a) \\
&= - R_a(R_a - E_a)s_a(1 - s_a)
\end{aligned}
$$

and

$$
\begin{aligned}
Cov(I_{R_a^+,i}, I_{C_a,i}) &= E\left[I_{R_a^+,i}I_{C_a,i}\right] - E\left[I_{R_a^+,i}\right]E\left[I_{C_a,i}\right] \\
&= E\left[R_a S_a\right] - R_a s_a s_a \\
&= R_a s_a(1 - s_a)
\end{aligned}
$$

Now that we know the relevant expectations, variances and covariances, we can compute the expectation and variance of P and R. We have

$$P = \sum_{a \in A} \sum_{i=1}^{t_a} (1 - \alpha)I_{R_a^+,i} - \alpha I_{R_a^- - E_a^-,i}$$

and

$$R = \sum_{a \in A} \sum_{i=1}^{t_a} I_{R_a^+,i} - \beta I_{C_a,i}$$

Hence,

$$E\left[P\right] = \sum_{a \in A} t_a((1 - \alpha)R_a s_a - \alpha(R_a - E_a)(1 - s_a))$$

$$E\left[R\right] = \sum_{a \in A} t_a(R_a s_a - \beta s_a)$$

Let $P^a = (1 - \alpha)R_a^+ - \alpha(R_a^- - E_a^-)$ and $R_a = R_a^+ - \beta C_a$ Thus $P = \sum_{a \in A} P^a$ and $R = \sum_{a \in A} R^a$

$$
\begin{aligned}
Var(P^a) =&Var(\sum_{i=1}^{t_a}(1 - \alpha)I_{R_a^+,i} - \alpha I_{R_a^- - E_a^-,i}) \\
=&t_a(Var((1 - \alpha)I_{R_a^+,1}) + Var(-\alpha I_{R_a^- - E_a^-,1}) \\
&+ 2Cov((1 - \alpha)I_{R_a^+,1}, -\alpha I_{R_a^- - E_a^-,1})) \\
&+ t_a(t_a - 1)(Cov((1 - \alpha)I_{R_a^+,1}, (1 - \alpha)I_{R_a^+,2}) \\
&+ Cov(-\alpha I_{R_a^- - E_a^-,1}, -\alpha I_{R_a^- - E_a^-,2}) \\
&+ 2Cov((1 - \alpha)I_{R_a^+,1}, -\alpha I_{R_a^- - E_a^-,2})) \\
=&t_a((1 - \alpha)^2 Var(I_{R_a^+,1}) + (\alpha)^2 Var(I_{R_a^- - E_a^-,1}) \\
&- 2\alpha(1 - \alpha)Cov(I_{R_a^+,1}, I_{R_a^- - E_a^-,1})) \\
&+ t_a(t_a - 1)((1 - \alpha)^2 Cov(I_{R_a^+,1}, I_{R_a^+,2}) \\
&+ (\alpha)^2 Cov(I_{R_a^- - E_a^-,1}, I_{R_a^- - E_a^-,2}) \\
&- (1 - \alpha)\alpha 2Cov(I_{R_a^+,1}, I_{R_a^- - E_a^-,2})) \\
=&t_a((1 - \alpha)^2 R_a s_a(1 - R_a s_a) \\
&+ \alpha^2(R_a - E_a)(1 - s_a)(1 - (R_a - E_a)(1 - s_a)) \\
&+ 2\alpha(1 - \alpha)R_a s_a(R_a - E_a)(1 - s_a)) \\
&+ t_a(t_a - 1)((1 - \alpha)^2 R_a^2 v_a \\
&+ \alpha^2(R_a - E_a)^2 v_a + 2\alpha(1 - \alpha)R_a(R_a - E_a)v_a) \\
=&t_a((1 - \alpha)^2 R_a s_a(1 - R_a s_a) + \\
&\alpha^2(R_a - E_a)(1 - s_a)(1 - (R_a - E_a)(1 - s_a)) \\
&+ 2\alpha(1 - \alpha)R_a s_a(R_a - E_a)(1 - s_a)) \\
&+ t_a(t_a - 1)v_a(R_a - \alpha E_a)^2
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Var}(R^a) =& \mathrm{Var}\left(\sum_{i=1}^{t_a} I_{R_a^+,i} - \beta I_{C_a,i}\right)\\
=& t_a\left(\mathrm{Var}(I_{R_a^+,1}) + \mathrm{Var}(-\beta I_{C_a,1}) + 2\mathrm{Cov}(I_{R_a^+,1}, -\beta I_{C_a,1})\right)\\
&+ t_a(t_a-1)\left(\mathrm{Cov}(I_{R_a^+,1}, I_{R_a^+,2}) + \mathrm{Cov}(-\beta I_{C_a,1}, -\beta I_{C_a,2})\right.\\
&\left.+ 2\mathrm{Cov}(I_{R_a^+,1}, -\beta I_{C_a,2})\right)\\
=& t_a\left(\mathrm{Var}(I_{R_a^+,1}) + (\beta)^2\mathrm{Var}(I_{C_a,1}) - 2\beta\mathrm{Cov}(I_{R_a^+,1}, I_{C_a,1})\right)\\
&+ t_a(t_a-1)\left(\mathrm{Cov}(I_{R_a^+,1}, I_{R_a^+,2}) + (\beta)^2\mathrm{Cov}(I_{C_a,1}, I_{C_a,2})\right.\\
&\left.- 2\beta\mathrm{Cov}(I_{R_a^+,1}, I_{C_a,2})\right)\\
=& t_a\left(R_a s_a(1 - R_a s_a) + \beta^2 s_a(1 - s_a) - 2\beta R_a s_a(1 - s_a)\right)\\
&+ t_a(t_a-1)(v_a R_a^2 + v_a\beta^2 - 2\beta R_a v_a)\\
=& t_a\left(R_a s_a(1 - R_a s_a) + \beta^2 s_a(1 - s_a) - 2\beta R_a s_a(1 - s_a)\right)\\
&+ t_a(t_a-1)v_a(R_a - \beta)^2
\end{aligned}
$$

Until now, we hadn't assumed anything about the correlation between S_as for different as. It's known that for any set of random variables X_i over some values of i,

$$
\mathrm{Dev}\left(\sum_i X_i\right) \le \sum_i \mathrm{Dev}(X_i)
$$

. Equality holds in the worst case where all X_is are fully correlated. If X_is are independent, then we have $\mathrm{Var}(\sum_i X_i) = \sum_i \mathrm{Var}(X_i)$. We now consider two cases.

Unknown Correlations Case:
If we know nothing about the correlations between S_as, the best bounds we can place on $\mathrm{Dev}(P)$ and $\mathrm{Dev}(R)$ are $\sum_{a\in A}\mathrm{Dev}(P^a)$ and $\sum_{a\in A}\mathrm{Dev}(R^a)$ respectively.

When we write the constraints $E[P] \le c_\rho\mathrm{Dev}(P)$ and $E[P] \le c_\rho\mathrm{Dev}(R)$, we want the resulting optimization problem to be easy to solve. Specifically, we would like the constraints to be linear, and if that's not easily possible, then we'd like them to be convex. Expectations of P and R are already linear in decision variables R_a and E_a, and so is the objective function. We will try to upper bound the rhs of both constraints (the deviations) so as to make it linear, while using a fairly tight upper bound. The rhs of both constraints are going to be a sum of standard deviations over $a \in A$. The standard deviation is a square root of the variance, which consists of a t_a factor and a $t_a(t_a - 1)$ factor. The key observation is that t_a is the only large quantity in the expression, where large means that is scales with the size of the table. The other terms, like R_a, E_a, s_a, α, β are all between 0 and 1. Thus only the $t_a(t_a - 1)$ factor under the square root can contribute an $O(n)$ factor to the rhs (where $n = \sum_{a\in A} t_a$ is the size of the table), while the t_a factor can contribute at most $O(\sqrt{n})$. Moreover, since the cost of our final solution is likely to be $O(n)$, it is ok if the rhs is made linear by adding to it a factor that is at most $O(\sqrt{n})$.

Specifically, we have

$$
\begin{aligned}
&(1-\alpha)^2 R_a s_a(1 - R_a s_a)\\
&+ \alpha^2(R_a - E_a)(1 - s_a)(1 - (R_a - E_a)(1 - s_a))\\
&+ 2\alpha(1-\alpha)R_a s_a(R_a - E_a)(1 - s_a)\\
\le&\, 0.25(1-\alpha)^2 + 0.25\alpha^2 + 0.25*2\alpha(1-\alpha)\\
=&\, 0.25
\end{aligned}
$$

$$
\begin{aligned}
&R_a s_a(1 - R_a s_a) + \beta^2 s_a(1 - s_a) - 2\beta R_a s_a(1 - s_a)\\
=&\,(R_a s_a(1 - R_a s_a) - R_a^2 s_a(1 - s_a)) + (R_a^2 s_a(1 - s_a)\\
&+ \beta^2 s_a(1 - s_a) - 2\beta R_a s_a(1 - s_a))\\
=&\, s_a R_a(1 - R_a) + (R_a - \beta)^2 s_a(1 - s_a)\\
\le&\, s_a R_a(1 - R_a) + \max(R_a^2, (1 - R_a)^2)s_a(1 - s_a)\\
=&\,\max(s_a R_a(1 - R_a) + R_a^2 s_a(1 - s_a),\\
&\quad s_a R_a(1 - R_a) + (1 - R_a)^2 s_a(1 - s_a))\\
=&\,\max(s_a R_a(1 - R_a + R_a - R_a s_a),\\
&\quad s_a(1 - R_a)(R_a + 1 - R_a - (1 - R_a)s_a))\\
=&\,\max(s_a R_a(1 - s_a R_a), s_a(1 - R_a)(1 - (1 - R_a)s_a))\\
\le&\,\max(0.25, 0.25)\\
=&\, 0.25
\end{aligned}
$$

And hence

$$
\begin{aligned}
\mathrm{Var}(P^a) =& t_a((1-\alpha)^2 R_a s_a(1 - R_a s_a)\\
&+ \alpha^2(R_a - E_a)(1 - s_a)(1 - (R_a - E_a)(1 - s_a))\\
&+ 2\alpha(1-\alpha)R_a s_a(R_a - E_a)(1 - s_a))\\
&+ t_a(t_a - 1)v_a(R_a - \alpha E_a)^2\\
\le&\, t_a^2 v_a(R_a - \alpha E_a)^2 + 0.25t_a
\end{aligned}
$$

We can then use

$$
\sqrt{x^2 + y^2} \le |x| + |y|
$$

to get

$$
\begin{aligned}
\mathrm{Dev}(P^a) =&\sqrt{\mathrm{Var}(P^a}\\
\le&\sqrt{v_a}t_a(R_a - \alpha E_a) + 0.5\sqrt{t_a}
\end{aligned}
$$

Hence

$$
\mathrm{Dev}(P) \le \sum_{a\in A}\sqrt{v_a}t_a(R_a - \alpha E_a) + 0.5\sqrt{t_a}
$$

giving us a linear upper bound on the rhs of the constraint as needed. Similarly, we can get

$$
\mathrm{Dev}(R) \le \sum_{a\in A}\sqrt{v_a}t_a|R_a - \beta| + 0.5\sqrt{t_a}
$$

. Thus we get the convex optimization problem:

$$
\text{minimize} \sum_{a\in A} t_a(R_a c_r + E_a c_e) \text{ such that}
$$

$$
\sum_{a\in A}(1-\alpha)t_a R_a s_a - t_a\alpha(R_a - E_a)(1 - s_a)
$$

$$
\ge c_\rho\sum_{a\in A}\sqrt{v_a}t_a(R_a - \alpha E_a) + 0.5\sqrt{t_a}
$$

$$
\sum_{a\in A} t_a R_a s_a - \beta t_a s_a \ge c_\rho\sum_{a\in A}\sqrt{v_a}t_a|R_a - \beta| + 0.5\sqrt{t_a}
$$

$$
R_a, E_a \in \{0, 1\}; R_a \ge E_a
$$

And solving this problem gives a solution which has precision $\ge \alpha$ and recall $\ge \beta$ with probability at least ρ each.

Independent S_a's case:
Now we move to the case where S_as are independent of each other. This case is worth considering because it's common. If we groups tuples by their value in a column, and evaluate a sample of tuples

from each group, then we get selectivity estimates for each group, with some variance. But tuple evaluations from one group don't give us information about selectivity of other groups. Thus in case of sampling, we have independent random variables S_a for each $a \in A$.

Because of the independence, we have

$$\text{Var}(P) = \sum_{a \in A} \text{Var}(P^a)$$

and

$$\text{Var}(R) = \sum_{a \in A} \text{Var}(R^a)$$

Thus in the optimization problem constraints are

$$E[P] \leq c_\rho \sqrt{\sum_{a \in A} \text{Var}(P^a)}$$

*

$$E[R] \leq c_\rho \sqrt{\sum_{a \in A} \text{Var}(R^a)}$$

Again, the objective function as well as lhs of both constraints are linear in decision variables R_a and E_a. The rhs has a square root. Like we did before, we would like to upper bound the rhs so as to make the optimization problem linear or convex. In this case, there does not seem to be a way to make the rhs linear without increasing it significantly. But it is possible to make the problem convex instead, with a slight increase in the rhs.

As in the last section, we note that

$$\text{Var}(P^a) \leq t_a^2 v_a (R_a - \alpha E_a)^2 + 0.25 t_a$$

Thus

$$\text{Dev}(P) \leq \sqrt{\sum_{a \in A} t_a^2 v_a (R_a - \alpha E_a)^2 + 0.25 t_a}$$

The rhs is a square root of a sum of squares and a positive constant. Suppose we have a $|A| + 1$ dimensional vector $\vec{R}$ whose components are the $R_a \sqrt{t_a^2 v_a}$ for each $a \in A$ and the constant $\sqrt{\sum_{a \in A} 0.25 t_a}$ and a $|A| + 1$ dimensional vector $\vec{E}$ whose components are $\alpha E_a \sqrt{t_a^2 v_a}$ for each $a \in A$ and 0. Then the rhs is given by $\left\|\vec{R} - \vec{E}\right\|_2$. For vectors $\vec{R_1}, \vec{E_1}, \vec{R_2}, \vec{E_2}$, by the triangle inequality, we have

$$\left\|\vec{R_1} - \vec{E_1}\right\|_2 + \left\|\vec{R_2} - \vec{E_2}\right\|_2 \geq 2 \left\|\frac{\vec{R_1} - \vec{E_1}}{2} + \frac{\vec{R_2} - \vec{E_2}}{2}\right\|_2$$

$$= 2 \left\|\frac{\vec{R_1} + \vec{R_2}}{2} - \frac{\vec{E_1} + \vec{E_2}}{2}\right\|_2$$

Thus the function $\left\|\vec{R} - \vec{E}\right\|_2$ is convex over the R_as and E_as.

Now consider the precision constraint

$$E[P] \geq c_\rho \left\|\vec{R} - \vec{E}\right\|_2$$

Since the lhs is linear in R_as and E_as, while the rhs is convex, the constraint is convex. Similarly, we can show that upper bounding $\text{Dev}(R)$ by $\sqrt{\sum_{a \in A} t_a^2 v_a (R_a - \beta)^2 + 0.25 t_a}$ makes the recall constraint convex. Thus we reduce our problem to the following convex optimization problem

minimize $\sum_{a \in A} t_a (R_a c_r + E_a c_e)$ such that

$$\sum_{a \in A} (1 - \alpha) t_a R_a s_a - t_a \alpha (R_a - E_a)(1 - s_a)$$

$$\geq c_\rho \sqrt{\sum_{a \in A} t_a^2 v_a (R_a - \alpha E_a)^2 + 0.25 t_a}$$

$$\sum_{a \in A} t_a R_a s_a - \beta t_a s_a \geq c_\rho \sqrt{\sum_{a \in A} t_a^2 v_a (R_a - \beta)^2 + 0.25 t_a}$$

$$R_a, E_a \in \{0, 1\}; R_a \geq E_a$$

which can be solved with any local optimization technique.

10.5 Justification for Rule of Thumb from Section 4

We will justify our rule of thumb, by making some simplifying assumptions in our equations, and then optimizing our problem locally. To start with, we ignore the exact values of R_as, E_as and s_as and instead write the precision and recall constraints approximately as

$$\sum_{a \in A} F_a \tau_1 - \sqrt{\sum_{a \in A} \frac{\tau_2 t_a^2}{F_a}} \geq 0$$

for some constants τ_1 and τ_2.

Then, the derivative of the lhs with respect to F_a is:

$$\tau_1 - \frac{t_a^2 \tau_2}{2 F_a^2 \sqrt{\sum_{a \in A} \frac{\tau_2 t_a^2}{F_a}}}$$

The derivative is negative when $F_a \approx 0$, and thus increasing F_a helps satisfy the constraint better (compared to increasing other Fs or Es with larger derivative). As F_a increases, the derivative decreases, and the marginal value of increasing F_a decreases. In the locally optimal solution, the derivative with respect to each F_a will be equal. Suppose for the locally optimal solution,

$$Y = 2 \sqrt{\sum_{a \in A} \frac{\tau_2 t_a^2}{F_a}}$$

, and the derivative with respect to each F_a is τ_3. Then, we have

$$\tau_1 - \tau_3 = \frac{t_a^2}{F_a^2 Y}$$

and hence

$$F_a \approx \frac{t_a}{\sqrt{Y(\tau_1 - \tau_3)}}$$

Let $n = \sum_{a \in A} t_a$ be the total number of tuples in the table. Now we want to find the order of magnitude value of Y. We have

$$\frac{t_a}{F_a} = \sqrt{Y((\tau_1 - \tau_3))}$$

and thus

$$Y = 2\sqrt{\sum_{a \in A} \frac{\tau_2 t_a^2}{F_a}}$$

$$= 2\sqrt{\sum_{a \in A} \sqrt{Y(\tau_1 - \tau_3)} \tau_2 t_a}$$

$$\Rightarrow Y^{\frac{3}{4}} \approx 2\sqrt{\tau_2 \sqrt{(\tau_1 - \tau_3)} n}$$

$$\Rightarrow Y = \left(4\tau_2 \sqrt{\tau_1 - \tau_3} n\right)^{\frac{2}{3}}$$

$$\Rightarrow F_a = \frac{t_a}{\sqrt{\left(4\tau_2 \sqrt{\tau_1 - \tau_3} n\right)^{\frac{2}{3}} (\tau_1 - \tau_3)}}$$

$$\Rightarrow F_a \approx \tau_4 t_a n^{-\frac{1}{3}}$$

where tau_4 is a constant in terms of the other τs. This justifies our rule of thumb, with the constant τ_4 being the num parameter in the rule of thumb.

10.6 Extensions

10.6.1 Alternate Objective Functions

Our paper focuses on the scenario where precision and recall constraints are specified, while we try to minimize cost. An interesting variation of the problem is where a user has a fixed cost budget, and wishes to maximize the number correct tuples returned while maintaining some level of precision and not exceeding the budget.

Minor modifications of our techniques can be used to handle this variation. The cost now becomes one of the constraints, while recall (or precision) becomes the objective function to be maximized. We can tighten the constraints slightly to turn the resulting problem into a convex optimization problem, and solve it to maximize precision or recall while keeping cost within a given limit.

10.6.2 Multiple Predicates

Another variation of the problem is where the query might have multiple chained selects on a table (which is equivalent to a conjunction of multiple predicates). If any of these predicates is not an expensive UDF, then it makes sense to execute those predicate first. But after doing this, we may still have more than one UDF in the select condition, which creates an interesting variation of our problem. The precision and recall constraint are specified by the user for the final output, making it possible to trade-off accuracy in one predicate for higher accuracy in the other at the same cost. Also, the probability of a tuple being correct is now affected by the probabilities of it satisfying both the predicates, so if a tuple that is very unlikely to satisfy one predicate, then we may not bother to evaluate the second predicate on it even if it was likely to satisfy the second predicate (because it probably won't be in the final output anyway).

We can extend our method to solve this variation of the problem, by introducing decision variables representing combined decisions on the predicates (For example, there may be a decision variable RR_a for group a that is 1 if and only if we are assuming both predicates are true for the tuple). This gives us a problem where the number of variables is exponential in the number of predicates, but still linear in table size (which is likely to be much higher than number of predicates anyway), which we can solve with a variation of our techniques.

10.6.3 Single Predicate with Join

Another variation we can consider is where the table T being selected on is later going to be joined with another table T_2. Each tuple of the table T may match with a different number of tuples of T_2. Thus it may be worthwhile for us to evaluate a tuple with low correctness-probability that matches with a large number of tuples from T_2, over a tuple with higher correctness probability that joins with fewer tuples from T_2.

We can solve this variation by

- Creating a separate decision variable for each value in the join column. (Thus if $a \in A$ is a value in the correlated column, and j is a value in the join column, then we have decision variables $R_{a,j}$ and $E_{a,j}$).
- Multiplying the $R_{a,j}$ and $E_{a,j}$ terms in the precision and recall constraints by the number n_j of tuples from T_2 that have value j in the join column.