Efficient Top-K Count Queries over Imprecise Duplicates

Sunita Sarawagi
IIT Bombay
sunita@iitb.ac.in

Vinay S Deshpande
IIT Bombay
vinaysd@cse.iitb.ac.in

Sourabh Kasliwal
IIT Bombay
sourabh@cse.iitb.ac.in

ABSTRACT

We propose efficient techniques for processing various TopK count queries on data with noisy duplicates. Our method differs from existing work on duplicate elimination in two significant ways: First, we dedup on the fly only the part of the data needed for the answer — a requirement in massive and evolving sources where batch deduplication is expensive. The non-local nature of the problem of partitioning data into duplicate groups, makes it challenging to filter only those tuples forming the K largest groups. We propose a novel method of successively collapsing and pruning records which yield an order of magnitude reduction in running time compared to deduplicating the entire data first.

Second, we return multiple high scoring answers to handle situations where it is impossible to resolve if two records are indeed duplicates of each other. Since finding even the highest scoring deduplication is NP-hard, the existing approach is to deploy one of many variants of score-based clustering algorithms which do not easily generalize to finding multiple groupings. We model deduplication as a segmentation of a linear embedding of records and present a polynomial time algorithm for finding the R highest scoring answers. This method closely matches the accuracy of an exact exponential time algorithm on several datasets.

1. INTRODUCTION

The problem of eliminating duplicates in large record lists has been actively researched in the database community. Much work exists on designing efficient join algorithms for finding duplicate pairs [3, 16, 19, 33, 32, 8, 28, 26], formulating the criteria for identifying two records as duplicates [18, 31, 11, 9, 6], and clustering pairs to get duplicate groups [13, 4, 10, 7].

However, most existing work assume a batch mode of deduplication, typically performed at the time of data loading. This approach of deduplicating first and querying later is not practical over sources that are constantly evolving, or are otherwise too vast or open-ended to be amenable to offline deduplication. In such cases, it is necessary to perform on-the-fly deduplication of only the relevant data subset. Another important aspect of dealing with such data sources is that sometimes it is impossible to resolve if two records are indeed duplicates of each other. Instead of hiding this intrinsic ambiguity under arbitrary policies and a single hard grouping of records, we propose that queries expose such ambiguity by returning multiple plausible answers.

In this paper we concentrate on how to answer several variants of TopK count queries on such sources. The TopK query returns the set of K largest groups in the data where each group consists of *all* duplicate mentions of an entity. To capture the inherent uncertainty of resolving duplicates, we associate a score with each possible grouping, and return the R highest scoring answers. We present example scenarios where such queries arise:

- Finding prolific inventors on a topic from a patents database.
- Web query answering where the result of the query is expected to be a single entity where each entity's rank is derived from its frequency of occurrences [22].
- Tracking the most frequently mentioned organization in an online feed of news articles.
- Compiling the most cited authors in a citation database created through noisy extraction processes.

All these examples share the following characteristics: the final answer requires aggregation of counts over several noisy mentions of entity names, it is not meaningful or feasible to perform a full resolution in advance, and it is impossible to claim with certainty if two mentions refer to the same entity.

There are several challenges to solving the above problem efficiently. First, given the non-local nature of the problem of partitioning data into duplicate groups, there is no obvious extension of existing algorithms to retain data relevant only to the K largest groups. Two records can be called duplicates not just based on that pair but also on other related records in the data. This makes it difficult to predict for a record, how large a group it might eventually get merged with. Second, finding even the highest scoring answer is NP-hard for most reasonable scoring functions. Extending

EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia.

existing approximation methods to finding the R highest scoring answer is not obvious. Third, finding the score of a TopK answer is non-trivial because it requires summing over the score over exponential possible groupings.

1.1 Contributions

We propose a query processing framework on data with duplicates that extends the capabilities of existing systems in two significant ways: First, it requires deduping on the fly the part of the data actually needed for the answer. Second, it considers situations where it is impossible to fine tune an exact deduplication criteria that can precisely resolve when two records are the same. In this paper we illustrate how we address these two challenges for the TopK count query and other related queries.

We propose a novel method of pruning records guaranteed not to be in the TopK set by using a notion of sufficient predicates to collapse obvious duplicates and a notion of necessary predicates to estimating lower bound on the size of a group in the answer and pruning away irrelevant tuples. Our pruning algorithm is safeguarded against the two extremes of many mentions of the same entity and many groups with small number of mentions.

Experiments on three real-life datasets of up to a quarter million rows shows that for small values of K our algorithm reduces the data size to 1% of its original size. This leads to significant reduction in running time because most deduplication algorithms are in the worst case quadratic in the number of input records.

We propose a linear embedding of the records such that potential duplicates are clustered. Within this embedding, we provide an exact polynomial time algorithm for finding the R highest scoring TopK answers. We compared the quality of the answers we obtained under the embedding to exact answers on real-life datasets where it was possible to find the exact answer using existing algorithms. In all cases, the accuracy of the answer we obtain is 99% of the exact algorithm.

Outline. We review related work in Section 2. In Section 3 we formally define TopK queries on data with duplicate records and discuss existing deduplication algorithms. We then present our core algorithm for collapsing and pruning records in Section 4. In Section 5 we present our algorithm for finding the R highest scoring TopK answers from the pruned set of records. In Section 6 we present the results from our experimental evaluation. In Section 7 we present extensions of our pruning algorithm to other forms of aggregate count queries on data with duplicates.

2. RELATED WORK

We relate our work to existing work on duplicate elimination, TopK count queries, and queries over uncertain data integration steps.

Duplicate elimination. The literature on duplication elimination can be classified under the following categories: join algorithms for finding duplicate pairs [3, 16, 19, 33, 32, 1], index structures for approximate string matches [12, 37], formulating the criteria for identifying two records as duplicates [18, 31, 11, 9, 6], designing canopy predicates [8, 28, 26, 15], and clustering pairs to get duplicate groups [13, 4, 10, 7]. While this literature is relevant to solving the final deduplication problem on the pruned records, none of these have considered the problem of selectively deduping only part of the records relevant to a query answer.

Top-K aggregate queries. The TopK count problem, a special case of the TopK ranking problem, has been addressed in various settings. See [20] for a survey. The basic TopK count query is to return the K most frequently occurring attribute values in a set of records. Our problem differs in only one way from this — repeated mentions of values are not guaranteed to look the same, and it is expensive to establish if two values are indeed noisy variants of each other. This single difference, however, drastically changes the computational structure of the problem. For instance, all the optimizations proposed in [25] about scheduling the probing of attribute groups and member records, very crucially depend on being able to exactly and easily define the aggregation groups.

Top-K aggregate queries on uncertain data. The TopK aggregate query problem has recently been extended to uncertain data [35]. Their algorithm is designed to work with an arbitrary inference engine for computing uncertainty, so initially we attempted to find our R highest scoring answers by treating each potential group as an uncertain tuple in their framework. We soon realized that the nature of dependencies among the set of groups is too non-local for their algorithm to be practical. Also, they concentrate on computing general aggregate function after assuming that the set of groups and the bounds on the cardinality of each group is known. These assumptions do not hold in our case because there is no fixed set of groups. In fact, our primary problem is to compute the cardinality of such floating groups.

Uncertainty in data integration. There is increasing interest to model the outcome of data integration as a probabilistic database [17, 34, 2]. The focus in [17, 34] is on probabilistically representing the uncertainty in mapping the schema of two sources and computing the R most probable answers to queries. Our design point is similar: the scores that we assign to each possible grouping of data can be normalized to be probabilities and we are therefore returning the R most probable answers. However, the algorithmic challenges in uncertainty of schema mappings is drastically different from uncertainty of data grouping. Although [2] also deals with uncertainty in duplicate data, they assume that the groups produced by the deduplication algorithm is *certain* and the only imprecision is in whether a member exists at all. An exclusive OR model of assigning a probability to each member suffices for this setting. This is entirely in contrast to our setting — we are addressing the more difficult problem of uncertainty in group membership. The dependency structure of grouping data cannot be represented by tuple-level probability models and something more in-

volved, like our segmentation model, is needed to capture most of the interesting dependencies while retaining some computational tractability.

3. PROBLEM FORMULATION AND BACKGROUND

Given a dataset D comprising of d records $t_1, \ldots t_d$, the answer to a TopK query on D is a set of K groups $C_1 \ldots C_K$ that satisfy the following three condition:

1. Each group C_i is the maximal set of tuples such that all tuples in it are duplicates of each other. The maximality condition implies that there are no tuples outside of C_i that is a duplicate of any tuple in C_i.
2. The sizes of groups are non-increasing: size$(C_i) \geq$ size(C_j) for all $i > j$,
3. There exists no group in $D -$ TopK of size greater than size(C_K).

A trivial method of finding the top-K largest groups is to first find all duplicate groups and retain only the K-largest. We first review the existing algorithms for deduping records with the view of evaluating how they can be made more efficient for getting only the TopK largest groups. Almost all existing approaches follow these three steps:

Fast elimination of obvious non-duplicate pairs. First, a cheap canopy predicate is used to filter the set of tuple pairs that are likely to be duplicates. For example [26, 15] proposes to use TFIDF similarity on entity names to find likely duplicates. TFIDF similarity can be evaluated efficiently using an inverted index unlike measures like edit distance which might have been used for the final matching criteria. These are also called blocking predicates in [8, 28].

Join to get duplicate pairs. Next, a more expensive criteria P is applied on the filtered pairs. The criteria P is either user-defined or learned from examples using machine learning models [31, 11, 6]. It can output either a binary "duplicate" or "not-duplicate" decision or a real score where positive values indicate duplicates, negative values non-duplicates and the magnitude controls the degree of belief in the decision.

Cluster to get entity groups. Finally, since there is no guarantee that P satisfies triangle inequality, a clustering step is used to find the best grouping of records while minimizing the amount of violations of P. A well-known such criterion is correlation clustering which attempts to find that grouping for which the number of non-duplicate edge pairs within a group and duplicate edge pairs across groups is minimized [4]. This, and many other such criterion is NP-hard to optimize and approximate algorithm [10] and heuristics [14] are used to find the best grouping.

There is no obvious way of exploiting the fact that we are only interested in the K largest clusters to speed up any of the above three steps. The final groups are materialized only after the last clustering step whereas the most expensive part of a deduplication algorithm is the second join step. Since we are working with independent record pairs, it is not easy to avoid any part of the join. Likewise, the first step has no global information of the eventual group sizes to perform any pruning based on K. As we will see in the next section, a very different set of ideas have to be brought to bear to perform early pruning.

4. PRUNING AND COLLAPSING RECORDS

We propose to use a set of cheaper filters that can both prune irrelevant tuples not in the TopK set, and collapse duplicates as soon as possible. We define two kinds of binary predicates for this purpose:

1. **Necessary** predicates: these have to be true for any duplicate pair. Thus, a binary predicate $N(t_1, t_2)$ defined over tuple pair t_1, t_2 is a necessary predicate if

$$N(t_1, t_2) = false \implies \text{duplicate}(t_1, t_2) = false$$

2. **Sufficient** predicates: these have to be false for any non-duplicate pair. Thus, a binary predicate $S(t_1, t_2)$ defined over tuple pair t_1, t_2 is a sufficient predicate if

$$S(t_1, t_2) = true \implies \text{duplicate}(t_1, t_2) = true$$

Thus, for any duplicate pair it is necessary that $N(t_1, t_2)$ be true but $N(t_1, t_2)$ being true does not imply that t_1 is a duplicate of t_2. Conversely, if $S(t_1, t_2)$ is true then t_1 is a duplicate of t_2 but if $S(t_1, t_2) =$ false then it does not imply that t_1 is not a duplicate of t_2.

The necessary and sufficient predicates are assumed to be much cheaper to evaluate than the exact criteria P. Note that our necessary predicates correspond to Canopy predicates already in use in existing deduplication algorithms as discussed in Section 3. In this paper, we assume that such predicates are provided by the domain expert, along with the criteria P for pairwise deduplication. Since these are assumed to be weak bounds of the exact criteria, they are not too difficult to design. Consider one common class of predicates used in duplicate elimination that threshold the amount of overlap between the signature set of each record. For example, "Jaccard similarity of 3-grams $> T$" where T is a fraction between 0 and 1. A trivial sufficient predicate for such set-overlap predicates is to require the two sets to match exactly, or with a very high threshold. A trivial necessary predicate is that the sets need to have at least one word in common. Both these extremes are much easier to evaluate than the general case of an arbitrary threshold in-between. In the experiment section we provide more examples of such predicates.

In general there can be a series of necessary predicates $N_1 \ldots N_L$ and sufficient predicates $S_1, \ldots S_L$ of increasing cost. Let us first consider the case of using a single sufficient predicate S and necessary predicate N. There are three steps to reducing the size of the data using S and N.

Collapse using S:. Use the sufficient predicate S to group data D where each group is the transitive closure of tuple pairs that satisfy S. Collapse D by picking a representative from each of the groups. Let $c_1 \ldots c_n$ denote the groups generated in decreasing order of size. We elaborate on this step and prove why this preserves correctness of the rest of the steps in Section 4.1.

Estimate lower bound M on the size of a group in TopK:. Use the sizes of the groups $c_1, \ldots, c_n$ and the necessary predicate N to compute a lower bound M on the size of the smallest group in the answer. We show how to do this in Section 4.2.

Prune groups using M and N. Use N to obtain for each group c_i an upper bound u_i on its largest possible size. Prune away those groups c_i for which $u_i \leq M$. The output of this step is a set of groups $c_1 \ldots c_{n'}$ where we expect n' to be much smaller than n. We elaborate on this step in Section 4.3

4.1 Collapse tuples

Given a database consisting of d tuples $t_1, \ldots t_d$ we use the sufficient predicate S to create groups $c_1, c_2, \ldots c_n$ such that each group consists of the transitive closure of tuple pairs for which $S(t_i, t_j)$ is true. This can be computed efficiently because typically the sufficient predicate is based on stringent conditions, like exact match of names as illustrated in Section 6. Also, since we need a transitive closure any tuple pair that satisfies the predicate can be collapsed into a single record immediately. Subsequently, we replace all the tuples in a group by a single tuple as its representative. The correctness of the algorithm does not depend on which tuple is picked as a representative. In practice, one can apply criteria like centroidness to pick such a representative [36].

We prove that the above method of collapsing based on transitive closure and replacing each group by any arbitrary tuple as a representative is correct as far as further pruning is considered.

PROOF. First note that all tuples within a group are guaranteed to be duplicates of each other. That is, for all tuple pairs $t, t' \in c_i$, duplicate(t, t') is true. This holds because of the definition of sufficient predicates and the transitivity of the "duplicate-of" relation.

Next, we claim that for any choice of tuple pairs (t_1, t_2) where $t_1 \in c_1$ and $t_2 \in c_2$ as representatives for the clusters where

$$N(c_1, c_2) = N(t_1, t_2) \quad \text{and} \quad S(c_1, c_2) = S(t_1, t_2)$$

then, for all $t_1' \in c_1$ and $t_2' \in c_2$

$$N(t_1, t_2) = false \implies \text{duplicate}(t_1', t_2') = false.$$
$$S(t_1, t_2) = true \implies \text{duplicate}(t_1', t_2') = true$$

The proof easily follows from the fact that duplicate(t_1, t_1') and duplicate(t_2, t_2') are both true because of the first claim and the transitivity of the "duplicate-of" relation. □

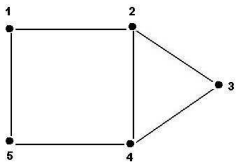

Figure 1: An example graph with optimal clique partition 2

This ensures that the predicates continue to be correctly applied in subsequent pruning and collapsing phases. Only in the final step when we need to find *all* duplicate pairs based on criteria P, the scores between tuples representing collapsed groups have to be calculated to reflect the aggregate score over the members on each side.

4.2 Estimate Lower Bound

We are given n collapsed groups $c_1, \ldots c_n$ in decreasing order of size and a necessary predicate N. We show how to estimate a lower bound M on the size of the smallest group in the TopK answer. In general, the lower bound will be smaller than the size of the Kth largest group c_K because any of the groups $c_1, \ldots c_K$ might be duplicates and collapsed in the final answer. Using N we estimate the smallest subscript m such that K distinct tuples are guaranteed to be found in $c_1, \ldots c_m$. Then $M = \text{size}(c_m)$ is a lower bound on the size of the K-th group in the TopK answer. We show how to estimate such an m.

A simple way is to check for each group c_i if it can potentially merge with any of the groups before it based on N. We apply this check for groups in decreasing size order $c_1, c_2 \ldots$ until we count k groups that cannot merge with any of the groups before it, or we reach the last group. However, this is a rather weak bound of m. Consider the example of five groups c_1, c_2, c_3, c_4, c_5 shown in Figure 1 where the edges connect group pairs for which N is true. Suppose $K = 2$. The above algorithm will return $m = 5$ since each of the five groups connect to a group before it in the ordering. But the optimal answer is $m = 3$ since $N(c_1, c_3)$ is false and so we are guaranteed to find $K = 2$ distinct groups in c_1, c_2, c_3.

We present next a provably better algorithm to estimate m. Consider a graph G where the vertices denote the n groups $c_1, c_2, \ldots c_n$ and edges connect vertex pair c_i, c_j when $N(c_i, c_j) = \text{true}$. Since N is a necessary condition for two tuples to be duplicates, any subset of groups that form a duplicate group in the answer has to be a clique in this graph. This implies that the minimum number of distinct groups represented by G is the minimum number of cliques required to cover all vertices of G. This is known as the clique partition number of a graph. More formally,

Given a graph $G(V, E)$, its Clique Partition Number (CPN) r is the smallest size of disjoint decompositions $V_1, \ldots V_r$ of V such that the vertices in each V_i form a clique. For the example graph in Figure 1, the CPN is 2 obtained by decomposing the vertex set into cliques (c_1, c_5) and (c_2, c_3, c_4). Thus, our required m is the smallest index $m \leq n$ such that $c_1, \ldots c_m$ has a CPN of K. We next we show how we

estimate the clique partition number CPN(G) of a graph.

4.2.1 Clique Partition Number of a graph

The estimation of the clique partition number of a graph is a well-known NP-hard problem. We obtain a lower bound using the following two observations.

- The CPN of a graph G' with a superset of the edges of G is a lower bound on the CPN of G. That is CPN($G' = (V, E')$) $\leq$ CPN($G = (V, E)$) if $E' \supset E$.
- The exact CPN of a triangulated graph can be found in polynomial time. A triangulated graph is one where all cycles of length greater than four have a short cut.

Our starting graph is not guaranteed to be triangulated but there are many well-known heuristics for adding edges to a graph to triangulate it. We use the well-known Min-fill algorithm [23]. Algorithm 1 shows how starting from a graph G we first triangulate G using the Min-fill algorithm and then estimate its clique partition number. The first loop obtains an ordering π of the vertices which for triangulated graphs has the property that all neighbors of the vertex π_i in the set $\pi_{i+1}, \ldots \pi_n$ are completely connected. For a non-triangulated graph, we have implicitly added extra edges to complete them, and also thereby triangulating the graph. In the second FOR loop, we go over the vertices in the new order π, and for each uncovered vertex v encountered we get a new vertex partition out of neighbors of v.

In the example graph in Figure 1, one likely ordering of vertices is c_3, c_1, c_2, c_4, c_5. In the second loop, this will first cover c_3 and its neighbors c_2, c_4, causing CPN to increase to 1. Next we cover the group c_1, c_5 causing the final CPN to be 2.

Algorithm 1 Find Clique Partition Number

```
Input: G = (V, E), V = c_1, ... c_n, E = {(c_i, c_j) : N(c_i, c_j) = true}
/* Min-fill algorithm for ordering vertices */
L = φ // set of marked vertices.
for ℓ = 1 to n do
    v = Vertex from V − L whose neighbors in V − L will require minimum number of extra edges to convert into a clique.
    Connect all neighbors Z of v to form a clique on Z ∪ {v}.
    π_ℓ = v
    Add v to L
end for
/* Get lower bound on the CPN of G */
Mark all vertices as uncovered.
CPN = 0
for ℓ = 1 to n do
    if π_ℓ is uncovered then
        Mark π_ℓ and all its neighbors as covered
        CPN = CPN + 1
    end if
end for
```

We have implemented an incremental version of the above algorithm so that with every addition of a new node to a graph, we can reuse work to decide if the CPN of the new graph has exceeded K. We skip the details here due to lack of space.

4.2.2 Proof of correctness

Let $V = c_1, \ldots c_n$ be a set of sufficient groups on a dataset D and $V' = c_1, \ldots c_m$ be any subset of V in decreasing order of size. We prove that if the TopK answer on D is $C_1, \ldots, C_K$ then size(C_K) $\geq M =$ size(c_m) if CPN(V', N) $\geq K$.

PROOF. We provide an outline of the proof based on three claims. A detailed proof of each claim is deferred to a full length version of the paper. First, we claim that the CPN of a graph induced by N on a set of groups $c_1, \ldots c_m$ is a lower bound on the number of distinct groups in it. Second, we claim that the CPN of a set of vertices V' in a graph $G = (V', E)$ cannot decrease with the addition of new vertices and corresponding edges. Third, if a subset of groups $c_1 \ldots c_m$ are guaranteed to form K groups based on N, then the size of each group in the final answer has to be at least size(c_m). □

Note, that the groups which finally comprise the TopK answer might not contain any of $c_1 \ldots, c_m$. The above proof has only shown that there exists one set of K distinct groups where each group is of size at least size(c_m).

4.3 Prune groups

Finally, we use the lower bound M to prune away all tuples that are guaranteed not to belong to any TopK group. For this, we use N to prune any group c_i for which an upper bound u_i on the largest possible group that it can belong to in the final answer is less than M. Clearly any group with size(c_i) $\geq M$ cannot be pruned. For the remaining, we get an upper bound by counting the sum of sizes of all groups c_j such that $N(c_i, c_j)$ is true. This upper bound can be progressively improved by recursively summing over the sum of sizes of all those groups c_j such that $N(c_i, c_j)$ is true and upper bound of c_j is greater than M. We implemented a two pass iterative version of this recursive definition.

4.4 Overall algorithm

Our overall method, which we call PrunedDedup, is summarized in Algorithm 2. In general, a set of L necessary and sufficient predicate pairs of increasing cost and increasing tightness is taken as input. For each pair (N_ℓ, S_ℓ), we apply the three steps of collapsing groups based on S_ℓ, estimating lower bound M using N_ℓ, and pruning groups based on N_ℓ to get a reduced set of groups. If we are left with K groups, we terminate since we found the exact answer. At the end we are left with a pruned dataset on which we apply the final pairwise function P and return the R highest scoring answers using techniques that we describe in the next section.

The main novelty of the PrunedDedup algorithm lies in the specific roles of the necessary and sufficient predicates to exploit small values of K for pruning. Sufficient predicates are key to ensuring that many obvious duplicates do not lead to the enumeration of quadratic number of pairs. Even for small K, the TopK groups might cover a large number of

Algorithm 2 PrunedDedup($D, K, S_1 \ldots S_L, N_1, \ldots, N_L, P, R$)

```
1: D_1 = D.
2: for ℓ = 1 to L do
3:    c_1, ... c_n=collapse(D_ℓ, S_ℓ)
4:    m, M=estimateLowerBound(c_1, ... c_n, N_ℓ, K)
5:    c_1, ... c_n'=prune(c_1, ... c_n, N_ℓ, M)
6:    D_ℓ+1 = c_1, ... c_n'.
7:    if (n' = K) return D_ℓ+1
8: end for
9: Apply criteria P on pairs in D_L+1 for which N_L is true.
10: Return R highest scoring TopK answers (Section 5.3)
```

tuples because real-life distributions are skewed. So, just pruning irrelevant tuples is not enough. The sufficient predicates make sure that large obvious groups get collapsed as soon as possible. Next, the collapsed groups lead to non-trivial estimates of the lower bound in conjunction with the necessary predicate. Neither the necessary nor the sufficient predicate on its own can provide such a lower bound. Without the sufficient predicate, size(c_m) would be 1 and without the necessary predicate we cannot guarantee which set of tuples will be in distinct groups. The algorithm avoids full enumeration of pairs based on the typically weak necessary predicates. The necessary predicates are used only in two modes: checking if the sizes of groups that a given group joins with is greater than M, and for a small subset of the data (typically of the order of K) enumerating the join edges when estimating the lower bound.

5. FINDING R HIGHEST SCORING ANSWERS

In this section we present how we handle the inherent imprecision of resolving if two records are the same. Instead of assuming that predicate P is capable of making precise boolean decisions, we use a scoring based framework where each possible grouping of the tuples in D is assigned a score where a higher score indicates a better grouping. These scores can be converted to probabilities through appropriate normalization, for example by constructing a Gibbs distribution from the scores. Since there is no single correct answer, we return a ranked list of the R highest scoring answers. We discuss scoring functions in Section 5.1 and algorithms for finding the ranked answer list in Sections 5.2 and 5.3.

5.1 Scoring functions

A number of different scoring methods have been proposed [27, 29, 4]. Most of these compose the score of a grouping in terms of pairwise scoring functions. We denote these as $P(t_1, t_2) \in \mathcal{R}$ where $P(t_1, t_2)$ denotes the score when t_1 and t_2 are duplicates and $-P(t_1, t_2)$ is the score when they are non-duplicates. Thus, when $P(t_1, t_2)$ is negative, the reward of calling the pair as non-duplicate is higher than calling it a duplicate. A score close to zero indicates uncertainty in resolving if the record pairs are duplicates or not. Such scores can be obtained through a number of means: hand tuned weighted combination of the similarity between the record pairs, log probability of the confidence value of a binary probabilistic classifier such as a logistic regression, scores from a binary SVM classifier, and so on.

The score Cscore($\mathcal{C}$) $\in \mathcal{R}$ of the partitioning $\mathcal{C} = c_1, \ldots c_n$ of D into duplicate groups can be composed out of the pairwise scores in a number of ways [4, 27]. A popular method called Correlation Clustering (CC) [4] defines the score of $\mathcal{C}$ as the sum of the scores of positive edges within a group minus the sum of the score of the negative edges across groups. That is,

$$\text{Cscore}(\mathcal{C}) = \sum_{i=1}^{n} \sum_{t \in c_i} \Big(\sum_{\substack{t' \in c_i \\ P(t,t')>0}} P(t,t') - \sum_{\substack{t' \notin c_i \\ P(t,t')<0}} P(t,t') \Big) \quad (1)$$

This function essentially measures the amount of agreement of the partition $\mathcal{C}$ with the pairwise scores. Finding an exact solution for this objective is NP-hard and many approximations have been proposed [4, 10, 14]. We discuss the solution proposed in [10] because it has the nice property that in many cases it is possible to detect if the algorithm did find the optimal answer.

LP-based grouping. In this approach, for each pair (i, j) of records we associate a binary decision variable x_{ij} that is 1 when the two records belong to the same group and 0 otherwise. In order for the pairwise decisions to correspond to a disjoint grouping of the data, they have to satisfy the consistency condition that for every triple of records i, j, k if any two of the pairs are 1, the third has to be a 1. Subject to this condition, we need to choose the best values of x_{ij}s so as to maximize the correlation clustering score. This can be formulated as a relaxed linear program as follows:

$$\max_{x_{ij}s} \textstyle\sum_{i,j:P(i,j)>0} P(i,j) x_{ij} - \sum_{i,j:P(i,j)<0} P(i,j)(1 - x_{ij})$$

such that

$$x_{ij} + x_{jk} - x_{ik} \leq 1 \qquad \forall i, j, k$$
$$0 \leq x_{ij} \leq 1 \qquad \forall i, j$$

When the above LP returns integral answers, the solution is guaranteed to be exact. [10] proposes a number of rounding schemes to obtain an approximate solution when the LP returns non-integral solutions.

The pairwise scoring function can be combined in other ways. For example, instead of summing over all positive pairs within a cluster, we take the score of the least positive pair. The exact choice of the scoring function is not the focus of this paper. We concentrate here on designing algorithms for finding the top-R highest scoring TopK groups for a user-provided R and scoring function. The only requirement of our algorithm is that the scoring function decompose over groups. That is, it should be possible to express the score of a partition as the sum of scores defined over individual groups where a group's score depends only on that group and the data, and not on how records outside it are grouped.

$$\text{Cscore}(\mathcal{C}) = \sum_{i=1}^{n} \text{Group\_Score}(c_i, D - c_i) \quad (2)$$

The CC scoring function in 1 above is decomposable based on the above definition.

The score of a TopK answer $(C_1 \ldots C_K)$ is the sum of the score of all groupings where $C_1 \ldots C_K$ are the K largest clusters. Computing the score of a candidate TopK answer is NP-hard because we need to sum over the score of the exponentially many possible grouping of the remaining tuples

such that no group is of size greater than size(C_K).

We need to somehow reduce the space of the set of partitions we consider both when estimating the score of a candidate answer and when searching for the R highest scoring answer. We propose initial arrangements of records that will allow us to efficiently aggregate over most high-scoring groupings of the data. We consider two alternatives for such arrangements for which it is possible to design polynomial time algorithms.

5.2 Hierarchical grouping

In this case we create a tree-like hierarchy of groups using any method from the vast literature on hierarchical clustering. Some example of such methods are: bottom-up agglomerative algorithms like single-link clustering and average-link clustering, top-down algorithms, and hybrids [21, 14] that first do a top-down division of points and then a bottom-up agglomeration.

Once we have all records arranged in a hierarchy, we can enumerate many possible disjoint groupings of the data by selecting different frontiers of the hierarchy. We designed a dynamic programming algorithm to find a ranked list of most likely groupings using leaf to root propagation algorithms. We do not present the algorithm because the method that we present next is for a more general initial arrangement that subsumes the case of hierarchies.

5.3 Linear ordering + segmentation

In this case we first create a linear arrangement of the records and define a grouping to be a segmentation of the linear ordering. Thus, given n starting records, we first reorder them as $\pi_1, \ldots \pi_n$. A grouping of the points is allowed to correspond to any valid segmentation of the πs where a group consists of a segment in the segmentation. The set of groupings that this method considers is strictly more than the set of groupings considered in the earlier method based on hierarchical clustering because we can always start from the linear ordering imposed by the hierarchy. Instead of restricting groups to be defined by frontiers of an initial hierarchy, we allow groupings to be arbitrary segmentation of the ordering.

We present algorithms for determining a good linear ordering in Section 5.3.1 and finally getting the R highest scoring TopK groups in Section 5.3.2.

5.3.1 Linear embedding

The goal in creating a linear embedding of the records is that similar records should be close together in the ordering so that good clusters are not missed by restricting to only clusters of contiguous records in the ordering. This goal qualitatively matches the goals of the well-known linear embedding problem defined as follows:

Let P_{ij} denote the similarity between points i and j. The linear embedding outputs a permutation $\pi_1, \ldots \pi_n$ of the n points so as to minimize $\sum_i \sum_j |\pi_i - \pi_j| P_{ij}$. This objective is minimized when pairs with large similarity values are placed close together.

Unfortunately this objective, like most objectives in clustering, is NP-hard. Many different approximation algorithms have been proposed including, a greedy algorithm, an iterative median finding algorithm, and a spectral method that arranges points based on the coordinates of the second Eigen vector of the similarity matrix [24]. We used the greedy method that incrementally chooses a next point to add at the ith position π_i so as to maximize distance weighted similarity with existing points defined as follows:

$$\pi_i = \text{argmax}_k \sum_{j=1}^{i-1} P(\pi_j, c_k)\alpha^{i-j-1} \quad (3)$$

where α is a fraction between 0 and 1 and serves to age the similarity values of positions far from i.

5.3.2 Finding R TopK segments

We show how to find the R highest scoring TopK answers by searching and aggregating over groupings corresponding to segmentations of the linear ordering.

First consider the case where $R = 1$. Let Ans1(k, i, ℓ) denote the highest scoring k largest groups obtained by the records between positions 1 and i where all but the top-k clusters are restricted to be of length no more than ℓ. We calculate Ans1(k, i, ℓ) recursively as follows:

$$\text{Ans1}(k,i,\ell) = \max\Big(\sum_{1\le j\le \ell} \text{Ans1}(k, i-j, \ell) + \text{S}(i-j+1, i), \max_{\ell < j \le i} \text{Ans1}(k-1, i-j, \ell) + \text{S}(i-j+1, i)\Big)$$

In the above S(i, j) is the Group_Score$(c, D - c)$ of creating a cluster c over the set of records between positions π_i and π_j, that is $c = \{\pi_k : i \le k \le j\}$.

The final answer is simply: $\max_\ell$ Ans1(K, n, ℓ) where n is the total number of records. Although the worst case complexity of this algorithm is $O(Kn^3)$, in practice it can be made faster by not considering any cluster including too many dissimilar points.

The above equation can be easily extended to maintain the top-R scoring answers instead of a single topmost scoring answer. Let AnsR denote the set of the R highest scoring answers. Define a new operator maxR that takes as input a set of values and returns the R highest. The AnsR set can now be calculated as:

$$\text{AnsR}(k,i,\ell) = \text{maxR}\Big(\sum_{1\le j\le \ell} \text{AnsR}(k, i-j, \ell) + \text{S}(i-j+1, i), \text{maxR}_{\ell < j \le i} \text{AnsR}(k-1, i-j, \ell) + \text{S}(i-j+1, i)\Big)$$

The inner maxR function takes as input $(i-\ell) * R$ candidate answers and returns R highest scoring of them. The outer maxR takes as input $2R$ candidate answers and returns R highest scoring of them. The final answer is maxR$_\ell\{\text{AnsR}(K, n, \ell)\}$.

6. EXPERIMENTS

In this section we evaluate the effectiveness of our data pruning and collapsing techniques for processing the TopK query

on large datasets, and the quality of our algorithm for finding the R highest scoring answers. We first establish the effectiveness of our pruning algorithm by showing in Section 6.2 the fraction of records pruned for different values of K. Next, we compare the running time of our algorithm to existing algorithm that do not perform such pruning in Section 6.3. Finally, in section 6.4 we evaluate our algorithm for getting the R highest scoring answers.

6.1 Datasets and predicates

Our experiments for evaluating the pruning strategies were performed on three large real-life datasets. For each dataset we manually selected necessary and sufficient predicates. We used hand-labeled dataset to validate that the chosen predicates indeed satisfy their respective conditions of being necessary and sufficient. Unlike the necessary and sufficient predicates, the final deduplication predicate is difficult to design manually. Therefore, we used a labeled dataset to train a classifier that takes as input a pair of records and outputs their signed score of being duplicates of each other [31].

6.1.1 The Citation dataset

We obtained a database of 150,000 citations from a crawl of the Citeseer database. The data has already been subjected to one level of deduplication within Citeseer and each citation has a count field indicating the number of citations that it summarizes. However, many duplicates still exist in the data. We segmented the raw citation records into five fields namely, author, title, year, page number and rest using a CRF-based information extraction system [30]. The query was to return the K most cited authors. We first collapsed obvious duplicate citations by merging all records with more than 90% common words. This yielded 81,000 citations out of a total of 150,000. Next, every author-citation pair is considered a record, giving rise to roughly 240,000 records with an average of 3 authors per citation.

We designed two levels of sufficient predicates S_1 and S_2 and necessary predicates N_1 and N_2.

Sufficient Predicate S_1 is true when author initials match and the minimum IDF over two author words is at least 13. Basically, this predicate requires that for two author fields to be matched their names needs to be sufficiently rare and their initials have to match exactly.

Sufficient Predicate S_2 is true when initials match exactly, there are at least three common co-author words, and the last names match.

Necessary Predicate N_1 is true when the common 3-Grams in the author field is more than 60% of the size of the smaller field.

Necessary Predicate N_2 is true when the common 3-Grams in the author field is more than 60% of the size of the smaller field and there is at least one common initial.

Similarity function for final predicate. We used a number of standard similarity functions like Jaccard and Overlap count on the name and co-authors fields with 3-grams and initials as signature. On the Author field we also used a JaroWinkler similarity — an efficient approximation of edit distance specifically tailored for names [9]. In addition, we created the following two custom similarity functions:

- A custom similarity function defined on Author field as follows. Similarity is 1 when full author names (i.e., names with no initials) match exactly. Otherwise, it is the maximum IDF weight of matching words scaled to take a maximum value of 1.
- A custom similarity function defined on co-author field as follows: The similarity is the same as the above custom author similarity function when it takes either of the two extremes of 0 or 1. Otherwise, it returns the percentage of matching co-author words.

6.1.2 The Students Dataset

This dataset is collected from an education service that conducts exams for primary school children. Each record is a student's exam paper consisting of fields like the student's name, birth date, class, school code, paper code, and score obtained. There are roughly 170,000 records. The query is to identify the TopK highest scoring students requiring aggregating scores obtained in various papers of each student. Here, disambiguation is needed because the student names and birth date fields as entered by primary school students often contain errors. A common mistake in name fields is missing space between different parts of the name. A common mistake in entering the birth date is filling in the current date instead of the birth date. Other fields like the school code and class code are believed to be correct. The scores were not available to us due to privacy concerns. We therefore synthetically assigned scores by first grouping students based on the sufficient predicates. Then, a Gaussian random number with mean 0 and variance 1 is used to assign a proficiency score to each group. The members of the group are then assigned marks based on the proficiency.

We designed two levels of sufficient predicates S_1 and S_2 and necessary predicates N_1 and N_2.

Sufficient Predicates S_1 is true when student name, class, school code, and birth date all match exactly.

Sufficient Predicates S_s is similar to S_1 except that instead of exact name match it requires at least 90% overlap in the 3-grams of the name field.

Necessary Predicates N_1 is true when there is at least one common initial in the name and the class and school code match.

Necessary Predicates N_2 is true when there are at least 50% common 3grams between the names field and the school and class code match exactly.

Similarity function for final predicate. For this dataset, we did not have sufficient labeled training data, we therefore perform only the pruning experiments and skip the final clustering step based on a learned scoring function.

6.1.3 The Address dataset

This dataset consists of names and addresses obtained from various utilities and government offices of the city of Pune in India. The data had three attributes: "name, Address and Pin" and a total of 250,000 records of total size 40 MB. This dataset is a union of addresses from multiple asset providers (such as vehicles, houses, etc) collected for purposes of detecting income tax evaders and as such contains many duplicates. Each address mention can be associated with a weight indicating roughly an assessment of the financial worth of the asset. Such scores were not available to us due to privacy concerns and we therefore assigned synthetic scores using a method similar to the school dataset. The TopK query is to find the addresses with the highest scores.

For this dataset we used a single level of necessary N_1 and sufficient predicate S_1 as described next.

Sufficient predicate S_1 is true when the initials of names match exactly, the fraction of common non-stop words in the name is greater than 0.7 and the fraction of matching non-stop words in the address is at least 0.6. Our stop words list consisted of a hand compiled list of words like "street", "house" commonly seen in addresses.

Necessary predicate N_1 is true when the number of common non-stop words in the concatenation of the name and address fields be at least 4.

Similarity function for final predicate. The similarity functions used for the final predicate were Jaccard on the Name and Address fields with 3-gram and initials as signature, JaroWinker on the Name field, fraction of common words in the Address field after removing stop words, match of the Pincode field, and the custom similarity function defined for Authors in Section 6.1.1.

6.2 Pruning performance

In this section we report the number of records pruned by our algorithm for increasing value of K. In Tables 2, 4, and 3 we report for each TopK query on each dataset the following statistics about algorithm PrunedDedup in Section 4.4.

1. n : The number of records remaining after collapsing as a percentage of the starting number of records.
2. m : The rank at which K distinct groups are guaranteed.
3. M : The minimum weight of group to which a record should belong in order not to get pruned.
4. n' : Number of records retained after pruning as a percentage of the starting number of records.

In each case we notice that the final number of groups after the collapsing and pruning steps is significantly smaller than the starting number of records. For the first dataset, we started with 0.24 million records but were able to reduce the size to less than 1% for $K = 1$. Even with K as large as 1000, the final number of records is just 25% of the starting number. Similar reductions are observed for the other two datasets. For the Student dataset the percent of data

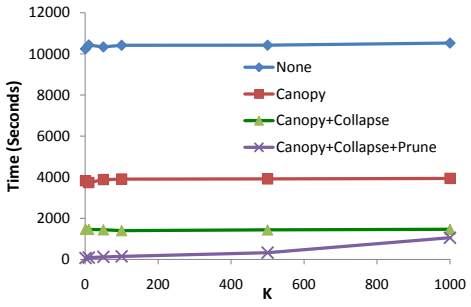

Figure 6: Timing comparison with different levels of optimizations.

remaining ranged from 1–4.7% of the starting number and for the address dataset the data was reduced to 0.55–4.05% of the starting size.

The reduction obtained by the first application of sufficient predicates is independent of K as expected. Even though the sufficient predicates do not reduce the number of records by large amounts, they are extremely useful in identifying a few large groups as evidenced by the huge skew in the values of M — the weight of group c_m obtained by collapsing using a sufficient predicate.

We also observe that our algorithm in Section 4.2 for estimating the value of m is tight in many cases. In particular for small values of K, the value of m at which K distinct groups are guaranteed is very close to K.

For the citations and students data where we applied two levels of necessary and sufficient predicates, we see a significant reduction of data sizes in the second stage over the results of the first stage. For the citation dataset for $K < 100$ we observe roughly 8–60 fold reduction in the first stage and a further 1.4–2 fold reduction in the second stage. For the students dataset, the second stage was lot more effective due to a tighter necessary predicate. Although, not shown in these tables, we observed that as per Section 4.3 recursively estimating the upper bounds via two iterations caused two-fold more pruning than using a single iteration. Going beyond two iterations caused very little improvement in the pruning performance.

6.3 Reduction in running time

In this section we compare the efficacy of our pruning and collapsing algorithm in reducing the overall running time. We compare our method against the following approaches:

- Canopy: This corresponds to applying the existing approaches of duplicate elimination on the entire data as listed in Section 3 where for the first canopy step we use the same necessary predicate as in our algorithm.
- Canopy+Collapse: This approach is the same as above but we also give the advantage of the sufficient predicates to the algorithm for collapsing sure duplicates before applying the canopy.

K	Iteration-1				Iteration-2			
	n	m	M	n'	n	m	M	n'
1	67.22	1	11970	1.70	1.56	1	11970	0.98
5	67.22	5	6896	5.55	4.60	5	6896	3.17
10	67.22	10	6101	6.49	5.33	10	6573	3.54
50	67.22	50	3396	13.67	10.77	51	3860	6.93
100	67.22	100	2674	17.59	13.80	101	2838	10.06
500	67.22	543	1166	31.34	24.69	547	1308	19.39
1000	67.22	1206	720	38.02	30.06	1166	802	25.50

Figure 2: Citation dataset: 240545 records.

K	Iteration-1				Iteration-2			
	n	m	M	n'	n	m	M	n'
1	39.00	1	2438	36.35	36.22	1	2438	0.99
5	39.00	5	2240	36.63	36.49	5	2240	1.23
10	39.00	10	2147	36.76	36.62	10	2165	1.38
50	39.00	58	1885	37.13	36.99	50	1906	2.25
100	39.00	115	1765	37.30	37.16	100	1795	2.75
500	39.00	760	1426	37.75	37.61	504	1498	4.10
1000	39.00	1628	1313	37.91	37.77	1011	1382	4.69

Figure 3: Student dataset: 169221 records

K	Iteration-1			
	n	m	M	n'
1	54.08	1	48213	0.56
5	54.08	5	32760	0.88
10	54.08	10	29958	0.94
50	54.08	52	19630	1.42
100	54.08	102	15690	1.84
500	54.08	508	9210	3.20
1000	54.08	1020	7418	4.05

Figure 4: Address dataset: 245260 records

Figure 5: Pruning performance for the three datasets. Values of n, M, n' are as a percentage of the total records.

In Figure 6 we show the running time for increasing values of K on a subset of 45,000 author citation records because the Canopy method took too long on the entire data. For reference, we also show numbers where we apply no optimizations at all (labeled "None" in Figure 6) — a straight Cartesian product of the records enumerates pairs on which we apply the final predicate to filter pairs which are then clustered. First, observe that necessary predicates when used as canopies achieve a significant reduction in running time compared to performing a full Cartesian product. Second, the sufficient predicates when used to collapse obvious duplicates reduce the running time by roughly a factor of two. Both of these methods cannot exploit any optimization specific to K. The biggest benefit is obtained by our method of estimating M and using that to prune records that will not participate in the final answer. For small values of K the running time is a factor of 20 smaller than the Canopy+Collapse method that uses the necessary and sufficient predicates in the obvious way. Note, neither of the competing approaches are capable of eliminating tuples altogether. The bounds on number of distinct clusters is key to the pruning of tuples. To the best of our knowledge, this has not been proposed before.

6.4 Finding the R highest scoring answers

It is difficult to comparatively evaluate our algorithm for finding the R highest scoring answers because finding even a single highest scoring grouping is NP-hard. Additionally, each TopK answer requires summing over a possibly exponential number of groupings that support the answer. We therefore restrict to small datasets where it is feasible to run the relaxed linear program (LP) of [10] discussed in Section 5. The LP can only provide a single highest scoring grouping, and that too only for cases when the LP variables take integral values. As a partial validation of the goodness of our algorithm we compare our highest scoring grouping with the exact solution returned by the LP.

Name	# Records	# Groups in LP
Author	1822	1466
Restaurant	860	734
Address	306	218
Getoor	1716	1172

Table 1: Datasets for comparing with exact algorithms

Our choice of dataset was therefore restricted to those where the LP-based algorithm terminated in reasonable time and returned integral solutions. For example, on the Cora benchmark [2], even though the LP completed within an hour, it returned non-integral solutions where we had no guarantees of optimality. We use the following four datasets for these experiments as summarized in Table 1: the Authors dataset consists of singleton list of author names collected from the Citation dataset, the Address dataset is a sample from the Address dataset described in Section 6.1.3, the Restaurant dataset is a popular deduplication benchmark [2] consisting of names and addresses of restaurants, the Getoor dataset is similar to the Citation dataset described in Section 6.1.1 and is collected by the authors of [5].

For each of the datasets, labeled duplicate groups were also available. We used 50% of the groups to train a binary logistic classifier using standard string similarity functions

[2]http://www.cs.utexas.edu/users/ml/riddle/data.html

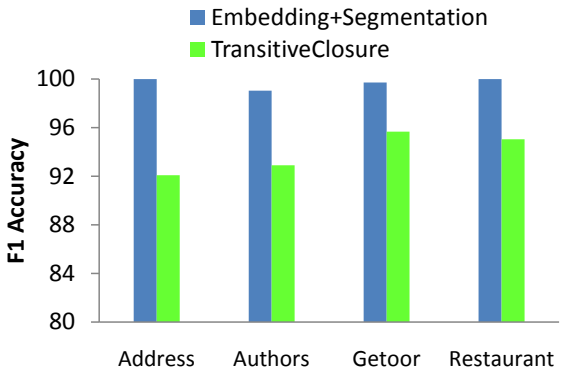

Figure 7: Comparing accuracy of highest scoring grouping with optimal

such as Jaccard and TF-IDF similarity at the level of words and N-grams. The classifier returns a real-valued score that is log of the probability that an input pair is a duplicate pair.

In Figure 7 we compare the accuracy of our algorithm with the groups identified by an exact LP-based approach. We measure accuracy as pairwise F1 value which treats as positive any pair of records that appears in the same cluster in the LP, and negative otherwise. For reference, we also compare the accuracy obtained by a baseline method that forms duplicate groups out of a transitive closure of all pairs with a positive score. We observe that our algorithm achieves a F1 accuracy of 100% on the Address and Restaurant dataset and above 99% on all four datasets. In contrast, the transitive closure algorithm achieves an agreement of only between 92–96% with the exact method. These results establishes that our representation of the partitioning space as a segmentation over a linear embedding preserves at least the highest scoring partition. We do not know of any other method of obtaining multiple groupings with which we could compare our R highest scoring answers.

7. EXTENSIONS TO OTHER QUERY TYPES

In this section we present two other query types that can exploit similar pruning and collapsing ideas as the TopK query.

7.1 TopK Rank Query

In the TopK rank query the user is only interested in getting a ranked order of the K largest group where a group can be identified by any canonical member in it. In order to answer this query, there is no need to find the exact size of each group in the TopK set. Instead, it suffices to find K pairs of the form $(c_1, u_1) \ldots (c_k, u_k)$ where

1. Each c_i is a set of duplicate records and u_i is an upper bound on the largest duplicate group containing c_i.
2. $\text{size}(c_i) \geq u_j$, for all groups after it in the ordering, that is $\forall i > j$.
3. There exists no group in $D-\text{TopK}$ of size greater than $\text{size}(c_k)$.

All the pruning and record collapsing ideas of the TopK count query apply to this query. In addition, we can perform more extensive pruning than the TopK count query where we needed to find the exact members of each group in the answer. The pruning step after removing all groups c_j where $u_j < M$ can also prune away additional groups defined as follows: First we define the notion of a *resolved* group. A group c_j is said to be resolved when the following condition is satisfied:

- $\forall g$ where $N(c_g, c_j) = false$, either $\text{size}(c_j) \geq u_g$ or $u_j \leq \text{size}(c_g)$
- $\forall g$ where $N(c_g, c_j) = true, u_g - \text{size}(c_j) < M$

The resolved group are those which have no ranking conflict with other groups and none of whose neighbors are capable of forming a group of size $\geq M$ on their own without the resolved group. The neighbors c_g of resolved groups can be pruned away provided they are not neighbors of other unresolved groups. In other words, we can prune away any group c_g which is disconnected from any unresolved group c_i whose $u_i \geq M$ even after all resolved groups are removed. This is because such groups have no role in resolving the rank of any group and by themselves they cannot form a new group because their size is $< M$.

7.2 Thresholded Rank query

For this query instead of controlling the number of groups via K, the user specifies a constant threshold T and the result is a ranked list of groups of size greater than T. This query can be processed using a simple modification of the TopK Rank query above. First, in Algorithm PrunedDedup instead of Step 4 where we estimated the lower bound, we set $M = T$. The pruning condition stays the same as for the above TopK rank query. The termination condition in Step 7 gets replaced with, there exists a k such that

- $\forall \;\; 1 \leq i < i' \leq k, \text{size}(c_i) \geq T, \text{size}(c_i) \geq u_{i'}$. This gives us a k above which all groups are guaranteed to form distinct groups of size $\geq T$ in the answer.
- $\forall j > k, j \leq n' \;\; \exists i \leq k$ such that $N(c_i, c_j) = true, u_j - \text{size}(c_i) \leq M$. This test ensures that all groups after k are redundant given the groups before k.

When this termination condition is satisfied we can return the groups $c_1, \ldots c_k$ as the answer, otherwise we continue with exact evaluation.

8. CONCLUSION AND FUTURE WORK

In this paper we presented efficient algorithms for computing the R highest scoring TopK groups on data where resolving duplicates is both expensive and uncertain. We proposed a novel method of exploiting necessary and sufficient predicates to exploit small values of K to significantly prune data records as evidenced by experiments on three real-life

datasets. Since finding both the score of a candidate answer and finding the highest scoring partitions is NP hard, we reduced the search space of plausible data partitions as segmentations of a linear embedding of records. In this space an exact polynomial time algorithm can be used to find the R highest scoring TopK groups. We showed that our pruning framework is applicable to other form of TopK count and ranking queries.

Future work includes methods for automatically choosing the necessary and sufficient predicates, designing a query optimization framework for selecting the best subset of predicates based on selectivity and running time, extending the ideas in this paper to more aggregation and ranking queries on data with noisy duplicates.

9. REFERENCES

[1] R. Ananthakrishna, S. chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, 2002.

[2] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, 2006.

[3] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, 2006.

[4] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. In *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*, page 238, Washington, DC, USA, 2002. IEEE Computer Society.

[5] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *TKDD*, 1(1), 2007.

[6] M. Bilenko. Learnable similarity functions and their applications to clustering and record linkage. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 981–982. AAAI Press / The MIT Press, 2004.

[7] M. Bilenko, S. Basu, and M. Sahami. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *ICDM*, 2005.

[8] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, 2006.

[9] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name-matching in information integration. *IEEE Intelligent Systems*, 2003.

[10] M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005.

[11] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik. Example-driven design of efficient record matching queries. In *VLDB*, pages 327–338, 2007.

[12] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, 2003.

[13] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *ICDE*, 2005.

[14] D. Cheng, R. Kannan, S. Vempala, and G. Wang. A divide-and-merge methodology for clustering. *ACM Trans. Database Syst.*, 31(4):1499–1525, 2006.

[15] W. Cohen and J. Richman. Learning to match and cluster entity names. In *ACM SIGIR' 01 Workshop on Mathematical/Formal Methods in Information Retrieval*, 2001.

[16] W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18(3):288–321, July 2000.

[17] X. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 687–698, 2007.

[18] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Society*, 64:1183–1210, 1969.

[19] L. Gravano, P. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *Proc. of the 27th Int'l Conference on Very Large Databases (VLDB)*, Rome,Italy, 2001.

[20] I. F. Ilyas, G. Beskales, and M. A. Soliman. Survey of top-k query processing techniques in relational database systems,. *To Appear in the ACM Computing Surveys, 2008*, 2008.

[21] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[22] J. Ko, T. Mitamura, and E. Nyberg. Language-independent probabilistic answer ranking for question answering. In *ACL*, 2007.

[23] D. Koller and N. Friedman. Structured probabilistic models. Under preparation, 2007.

[24] Y. Koren and D. Harel. A multi-scale algorithm for the linear arrangement problem. In *WG*, 2002.

[25] C. Li, K. C.-C. Chang, and I. F. Ilyas. Supporting ad-hoc ranking aggregates. In *SIGMOD Conference*, 2006.

[26] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Knowledge Discovery and Data Mining*, pages 169–178, 2000.

[27] A. McCallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web*, pages 79–86, Acapulco, Mexico, Aug. 2003.

[28] A. E. Monge and C. P. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996.

[29] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In *Advances in Neural Processing Systems 15*, Vancouver, British Columbia, 2002. MIT Press.

[30] S. Sarawagi. The crf project: a java implementation. `http://crf.sourceforge.net`, 2004.

[31] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD-2002)*, Edmonton, Canada, July 2002.

[32] S. Sarawagi and A. Kirpal. Scaling up the alias duplicate elimination system: A demostration. In *Proc. of the 19th IEEE Int'l Conference on Data Engineering (ICDE)*, Bangalore, March 2003.

[33] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2004.

[34] A. D. Sarma, X. Dong, and A. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008.

[35] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Probabilistic top- and ranking-aggregate queries. *ACM Trans. Database Syst.*, 33(3), 2008.

[36] M. L. Wick, K. Rohanimanesh, K. Schultz, and A. McCallum. A unified approach for schema matching, coreference and canonicalization. In *KDD*, 2008.

[37] X. Yang, B. Wang, and C. Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In *SIGMOD Conference*, pages 353–364, 2008.