Master's Thesis

K-ARQ:K-Anonymous Ranking Query

SukHyun An (안 석 현)

Division of Electrical and Computer Engineering

(Computer Science and Engineering)

Graduate School of the Pohang University of Science and Technology

2011

익명화된 순위화 질의

K-ARQ:K-Anonymous Ranking Query

K-ARQ:K-Anonymous Ranking Query

by
SukHyun An

Division of Electrical and Computer Engineering
(Computer Science and Engineering)
Pohang University of Science and Technology

A thesis submitted to the faculty of the Pohang University of Science and Technology in partial fulfillment of the requirements for the degree of Master of Science in the Division of Electrical and Computer Engineering (Computer Science and Engineering)

Pohang, Korea
12. 01. 2010
Approved by

Academic Advisor

K-ARQ:K-Anonymous Ranking Query

SukHyun An

The undersigned have examined this thesis and hereby certify that it is worthy of acceptance for a master's degree from POSTECH

12/01/2010

Committee	Chair	Seung-won Hwang	(Seal)
	Member	Hee-Kap Ahn	(Seal)
	Member	Hwanjo Yu	(Seal)

MECE 안 석 현 SukHyun An
20090688
K-ARQ:K-Anonymous Ranking Query, 익명화된 순위화 질의
Division of Electrical and Computer Engineering, 2011, 44P,
Advisor : Seung-won Hwang. Text in English.

ABSTRACT

Abstract

With the advent of an unprecedented magnitude of data, top-k queries have gained a lot of attention. However, existing work to date has focused on optimizing efficiency without looking closely at privacy preservation. In this paper, we study how existing approaches have failed to support a combination of accuracy and privacy requirements and we propose a new data publishing framework that supports both areas. We show that satisfying both requirements is an essential problem and propose two comprehensive algorithms. We also validated the correctness and efficiency of our approach using experiments.

Contents

List of Figures

List of Algorithms

List of Tables

	Quasi-identifier			Ranking attributes				
Tid	Age	Sex	Zipcode	Course1	Course2	Course3	Score	Rank
1	33	Female	53715	99	99	99	297	1
2	39	Male	53715	96	98	99	293	2
3	23	Female	53715	92	97	95	284	3
4	74	Male	53703	96	96	90	282	4
5	51	Male	53703	98	89	94	281	5
6	37	Female	53703	97	95	88	280	6
7	45	Female	53712	96	95	88	279	7
8	22	Male	53712	98	90	91	279	7
9	33	Female	53712	92	96	90	278	9
10	37	Female	53712	97	89	90	276	10

Figure 1: An Example Database

1 Introduction

With the advent of data on an unprecedented scale, there has been active research carried out on supporting ranking queries, to effectively narrow down results to a small desired number of matches according to a given ranking criteria [1, 2, 3, 4]. While existing work mostly focused on optimizing the efficiency of computing top-k ranked tuples, there has been less effort made on preserving privacy at the same time.

Privacy preservation research on databases has been also an active, study area including published work on k-anonymity [5], l-diversity [6], m-invariance [7], and t-closeness [8], but most of these approaches have treated every tuple more or less equally, without taking its ranking into account. As a result, applying these approaches directly on top-k ranked tuples would damage the result quality, by including many non top-k tuples in the results.

To illustrate this problem, consider an example database of job applicants as shown in Fig. 1, where the ranking function is the sum of all three test scores. A

recruitment company owns this database of job applicants and, a hiring manager of a company wants to see if this recruitment company has good candidates before paying a service fee. The recruitment company may reveal some information about the test scores of their top-k applicants to impress the hiring manager, without compromising the privacy of the applicants. A naive approach would be to publish the table as it is shown in Fig. 1 without any unique identifiers, such as names, SSNs, or passport numbers. However, as shown in [5], revealing quasi-identifiers may seem harmless but when combined with other seemingly harmless public data this may reveal the identities of the applicants. Another approach at the other end of the spectrum of privacy is to publish only the list of total scores, as in [9], but the returned data would only be of limited use for the hiring manager. Our goal is to assist the database owner to publish information on *ranking attributes*, the attributes that most affect tuples' ranking, without compromising too much on privacy.

To consider privacy, we adopt a widely-known privacy metric called k-anonymity. In a k-anonymous table, the tuples are indistinguishable from the other k-1 tuples. To analyze data quality, we employ a precision metric, *i.e.*, how many returned tuples are top-k ranked, out of the total number of returned tuples. Imagine a scenario where the recruitment company wants to publish the top-6 candidates with 3-anonymity.

Applying the top-k query algorithms to identify the top-6 ranked tuples and then aggregating their values results in Fig. 2(a). This approach includes all of the top-k tuples and, thus achieved *perfect recall*. Each ranking attribute is displayed as a range of values to provide 3-anonymity. Tuples 1, 2, and 3 are not individually identifiable, but are presented as a group of top-3 candidates. The tuple id's in

parentheses denote the tuples that are not in the top-6 but were included. Note that the hiring manager does not get to see the Tid column, so he or she can learn about the correlation between the individual course scores and the total score from the range but cannot know which exact tuples are included in the second range. This outcome displays a good result for the top-3 tuples, but the second top-3 tuples suffer only a 50% accuracy level, *i.e.*, only half of them are in the top-6.

Tid	Course1	Course2	Course3	Score	Precision
1,2,3	[92,99]	[97,99]	[95,99]	[284,297]	1
4,5,6,(7,8,10)	[96,98]	[89.96]	[88,94]	[226,282]	0.5

(a) Perfect Recall Example

Tid	Course1	Course2	Course3	Score	Precision
1,2,3	[92,99]	[97,99]	[95,99]	[284,297]	1
4,6,(7)	[96,97]	[95,96]	[88,90]	[279,282]	0.67

(b) k-ARQ Example

Figure 2: Results of three different algorithms applied on database in Fig. 1

We can also apply a 3-anonymity algorithm to the top-6 tuples. Unfortunately, achieving k-anonymity with a minimal change is NP-hard when $k > 3$ [10], so maximizing data quality while achieving k-anonymity on top-k tuples is not practical.

In this paper we propose the use of *k-ARQ*, a k-Anonymous Ranking Query approach, to ensure both privacy maintenance and precision when ranking attribute publication. We can show that calculating a set of tuples that optimize privacy and precision at the same time is an NP-Complete problem, and introduce two Greedy Approximation Algorithms. Fig. 2(b) shows the result from our Greedy Approximation Algorithms. *k-ARQ* achieves the same perfect precision as Fig. 2(a) for the

top-3 tuples, and achieves better precision for the next top-3. We summarize this comparison below in Table 1.

Table 1: The pros and cons of current state-of-arts

	Privacy	Precision	Ranking Attribute Publication
Privacy top-k [9]	✓	✓	
k-anonymity approximation [11]	✓		✓
k-ARQ	✓	✓	✓

Our key contributions to the research area are as follows:

- To the best of our knowledge, we are the first to study the problem of publishing ranking attributes, while supporting the dual requirements of privacy and accuracy.
- We formalize the problem, and show that it is an NP-complete problem.
- We develop two Greedy Approximation Algorithms, with optimization heuristics to enhance their efficiency.
- We evaluate our Greedy Algorithms in terms of appropriate use and efficiency, compared to existing approaches. The correctness is defined in this case as satisfying both the privacy maintenance and accuracy requirements, and the efficiency is measured in terms of elapsed time and precision.

The rest of the paper is organized as follows. Section 2 discusses the preliminaries and problem definition, and Section 3 shows that the problem is NP-Complete. Section 4 proposes our Greedy Algorithms. Section 5 improves greedy algorithm in terms of success ratio using local search algorithm. Section 6 presents our evaluation results. Section 7 surveys related work.

2 Preliminaries

As preliminaries, we first formally state our problem (Section 2.1) then discuss why existing solution (Section 2.2) has failed to address it.

2.1 Problem Definition

The input to our problem is a data table T = $\{t_1, \cdots, t_n\}$, which is a set of m-dimensional n tuples where each tuple t_i is represented by a set of values for m dimensions (attributes), $\{d_1, d_2, \cdots, d_m\}$. Each attribute in T is one of the following: *quasi-identifier*, *ranking attribute*, or *sensitive attribute*, which we will define formally below. Note that our definition is consistent with prior published literatures in this area, as we have indicated with citations.

Definition 1 (Quasi-identifier set QI [12])**.** *A quasi-identifer set QI is a set of attributes in table T that can be joined with external information to re-identify individual records, such as age, gender, and zip code in Fig. 1.*

Definition 2 (Ranking attribute set RA)**.** *A ranking attribute set RA is a maximal set of attributes that affects the overall ranking (i.e., parameters for $\mathcal{F}$ from the data table and its score), such as course test scores and the total score in Fig. 1. Typically ranking attributes have a total order, i.e.between any two values for a ranking attribute, it is defined as the one that is greater than or equal to the other.*

Definition 3 (Sensitive attribute SA)**.** *A sensitive attribute set SA is a set of attributes associated with the privacy of a user.*

We will now define our proposed problem: the output is a published table T_{pub}, which is derived from the input data table T and includes information on ranking at-

tributes, while satisfying privacy, ranking, and accuracy requirements. For privacy, we use the widely-used k-anonymity method, adopting its definition from [6]. The privacy requirement is specified by k_p for k_p-anonymity.

Definition 4 (K-Anonymity)**.** *A table T* $\{t_1, t_2, \cdots, t_n\}$ *is a set of tuples where each tuple has dimensions* d_j*. This table T satisfies k-anonymity if for every tuple* t_i *there exist* $k-1$ *other tuples* t_1, t_2, t_{k-1} *such that* $t_i[d_j] = t_1[d_j] = t_2[d_j] = \cdots = t_{k-1}[d_j]$ *for any dimension (attribute)* d_j*.*

The ranking requirement is specified by sending a query for the top k_q tuples based on the user-defined ranking function $\mathcal{F}$. For accuracy, we use the well-known metric of precision and, the ratio of the true positives in the overall results. In our case, this is the ratio of the number of top-k_q tuples in T_{pub} to the number of all tuples in T, and the requirement is specified by the lower bound of this ratio *prec*.

Definition 5 (k-ARQ problem)**.** *For the given data table* T *and the ranking function* $\mathcal{F}$ *we compute the score of each tuple and add this as another ranking attribute to* T*. For the given privacy, ranking, and accuracy requirements (*k_p*,* k_q*, prec), calculate a published output table* T_{pub} *that includes the ranking attribute set* RA*. (Note that* RA *includes the* $\mathcal{F}$ *score now.)* T_{pub} *is a successful result if and only if*

- k_p*-anonymity: for every tuple* t_i *in* T_{pub} *there exist* k_p *-1 other tuples* t_1, t_2, $\cdots$ t_{k_p-1} *such that* $t_i[d_j] = t_1[d_j] = t_2[d_j] = \cdots = t_{k_p-1}[d_j]$ *for any dimension (attribute)* d_j *in* RA*.*
- *Ranking and Accuracy* $|T_{pub} \cap K_q| \geq prec \times |T_{pub}|$*.*

where K_q *is a set of top-*k_q *tuples w.r.t* $\mathcal{F}$ *scores.*

Note that the traditional k-anonymity definition only includes QI. Our definition of k_p-anonymity is only defined over RA, but we can easily achieve k_p anonymity on QI as well. Based on the tuple groupings obtained for our problem, this can be done by simply aggregating the values of QI and SA of each group into ranges as shown in Fig. 3.

Tid	Age	Sex	Zipcode	Course1	Course2	Course3	Score	Precision
1,2,3	[23,39]	human	53715	[92,99]	[97,99]	[95,99]	[284,297]	1
4,6,(7)	[37,74]	human	[53703,53712]	[96,97]	[95,96]	[88,90]	[279,282]	0.67

Figure 3: Extension to Quasi-identifiers

However, this straightforward adoption is vulnerable to a *homogeneity attack* as discussed in [6], when all SA values in a group happen to be identical. For example, the first group in Fig. 3 shows that all of the top-3 candidates live in the 53715 zipcode area. If a zipcode were a sensitive attribute, then this provides no anonymity for the zipcode even though it is a group of 3 tuples. To avoid this vulnerability, we recommend the application of existing anonymization techniques such as those in [6, 7, 8] on QI and SA on the groupings obtained for our problem, instead of aggregating values into ranges as discussed above.

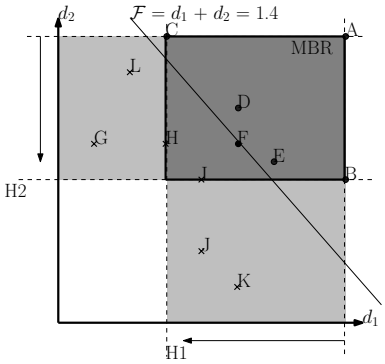
2.2 Baseline Approach I : Perfect Recall

We will now discuss a baseline approach of adopting existing solutions for our proposed problem in Section 2.1. We first discuss how to use top-k algorithms such as [1, 2, 3, 4] to distinguish the top-k tuples from the rest and how they, can be leveraged as a baseline solution to our problem.

Fig. 4 presents a geometric illustration of the top-k algorithms. Given that in-

d_1	d_2
B 1.0	C 1.0
A 1.0	A 1.0
E 0.8	L 0.9
K 0.7	D 0.8
F 0.7	G 0.7
D 0.7	H 0.7
J 0.6	F 0.7
I 0.6	E 0.65
H 0.5	I 0.6
C 0.5	B 0.6
L 0.4	J 0.4
G 0.3	K 0.3

(a) Ordered list of tuples



(b) 2-dimensional plot of tuples

Figure 4: Perfect Recall

dex structure ordering tuples in the order of attribute values d_1 and d_2 as shown in Fig. 4(a) illustrates that these algorithms access objects in a descending order of d_1 and d_2 values, which corresponds to a sweeping hyperplane orthogonal to the d_1 and d_2 axis respectively, downwards towards 0 in Fig. 4(b) as the arrows indicate. These algorithms terminate when the top-k results are guaranteed to exist among the tuples already accessed, *i.e.*, shaded regions in Fig. 4. In other words, when the upper bound score of all tuples in the unshaded region is no higher than k objects in the shaded region with the highest $\mathcal{F}$ scores, these algorithms can safely terminate.

We can use these algorithms to identify a "minimum-bounded rectangle" (which we denote as an MBR) where tightly bounding top k_q tuples are found, and marked as a box as shown in Fig. 4(b) above. When $k_p < k_q$ as in our example, publishing the range of the MBR satisfies both the privacy requirements and ensure all of the

actual top_{k_q} are included in T_{pub}, *i.e.*, achieves perfect recall.

While this baseline approach is guaranteed to publish all of the true positives, it does not have any control over how many false positives are included in the results. To illustrates this, when a ranking function is $\mathcal{F} = d_1 + d_2$, a hypersurface $d_1 + d_2 = 1.4$ represents a boundary distinguishing the true positives and negatives. This means that all the object values above this surface are true positives and the rest of object values in the dark-shaded triangular area are true negatives. Depending on data distributions and ranking functions, unbounded number object values may fall into the triangular area, which could lower their precision below the user-specified requirement $prec$.

2.3 Baseline Approach II: Perfect Precision

This section dicusses how to provide all top-k tuples while guaranteeing k-anonymity by dividing true positives and false positives in the MBR. As mentioned in Section 2.2, when a ranking function is $\mathcal{F} = d_1+d_2$, a hypersurface $d_1+d_2 = 1.4$ represents a boundary distinguishing the true positives and negatives for top-6 query in Fig. 4(b). This means that if we provide hypersurface $d_1 + d_2 = 1.4$ with the MBR of top-k tuples, the users can distinguish true positives and false positives in the MBR.

For example, in Fig. 4(b), assume that we provide a range [0.5, 1.0],[0.6, 1.0] and hypersurface $d_1 + d_2 = 1.4$. The tuples with the larger ranking score than 1.4 among tuples contained in the [0.5, 1.0],[0.6, 1.0] are the top-6 tuples. However, assume that a user requests a top-8 query. For k-anonymity, Perfect Precision provides a range [0.5, 1.0],[0.6, 1.0] with hypersurface $d_1 + d_2 = 1.2$ which is the ranking

score of the top-8 tuple H. As the top-6 query and top-8 query have the same range as results, understanding the difference between the results of the top-6 query and top-8 query is difficult. For this reason, the user cannot understand the quality of the range, such as precision.

2.4 Baseline Approach III: Mondrian

An alternative approach is to apply existing anonymization algorithms, that have been typically applied to anonymize both QI and RA. Specifically, we adopted the *mondrian* [11] method, which uses one of the state-of-the-art k-anonymity approximation algorithms, as a baseline approach.

mondrian achieves the user-specified privacy requirement k_p-anonymity by iteratively dividing a data space into equal-sized partitions such that each partition includes at least k_p objects. Mondrian divides the true positive set into half, initially, with respect to d_1 into partitions $P1$ and $P2$, as shown by the vertical division line in Fig. 5.

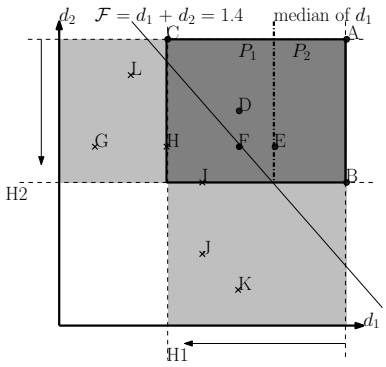


Figure 5: Mondrian

Fig. 5 illustrates such iterations, based on the same example used in Section 2.2,

where $k_q = 6$ and $k_p = 3$, More specifically, we divide the true positive set into half, initially with respect to d_1, as shown by a vertical division line in Fig. 5. This line divides the result set into partitions $P1$ and $P2$ as marked in the figure.

Tid	Course1	Course2	Course3	Score
1	[97,99]	99	99	297
5	[97,99]	89	94	281
6	[97,99]	95	88	280
2	[92,96]	98	99	293
3	[92,96]	97	95	284
4	[92,96]	96	90	282

(a) Step1 - choosing dimension

Tid	Course1	Course2	Course3	Score
1	[97,99]	[89,99]	[88,99]	297
5	[97,99]	[89,99]	[88,99]	281
6	[97,99]	[89,99]	[88,99]	280
2	[92,96]	[96,98]	[90,99]	293
3	[92,96]	[96,98]	[90,99]	284
4	[92,96]	[96,98]	[90,99]	282

(b) Step2 - recoding other dimension

Figure 6: Step-by-Step Mondrian execution

Fig. 6 illustrates each iteration of Mondrian as applied to our current example of $k_p = 3$ and $k_q = 3$ for the database in shown Fig. 1. The *Mondrian* approach first divides the values with respect to course 1, by picking the median course 1 score of six true positives, which is 97. Based on this value, we split course 1 into $[97, 99]$ and $[92, 96]$ segments, based on which tuples are divided into the two partitions of tuples 1,5 and 6 and 2,3, and 4, as Fig 6(a) shows. Mondrian then aggregate the values of the course 2 and course 3 scores in each partition into privacy ranges as Fig. 6(b) shows.

Note that this generalization (aggregation) of course 2 and course 3 values results in many false positives, such as for tuple 8,9 and 10 (shown in parentheses to mark false positives) in Fig. 6. Depending on the data distributions and the ranking function, an unbounded number of such false positives can be included in the published table, which makes it difficult to satisfy the user-specified accuracy requirement $prec$. This is due to the nature of the Mondrian method and other

k-anonymity approximation algorithms that treat all tuples equally without considering their ranks.

3 Hardness analysis

To show that *k-ARQ* is an NP-Complete problem, first we need to show that the solution to this problem is verifiable in polynomial time. After the verification proof, we will show that a well-known NP-Complete problem, *Subset Sum* problem can be reduced to apply to this problem in polynomial time by showing the polynomial-time reduction and how the solution to one problem is also a solution to the other. For the given parameters (k_p, k_q, *prec*), we can convert these parameters into (k, p) for a simpler proof. Given the privacy requirement k_p and the ranking requirement k_q, k = $\max(k_p, k_q)$. If $k_p < k_q$, then we can repeat this algorithm multiple times until we get at least k_q tuples. Given k, we can then identify the appropriate p value for the definition below.

Definition 6 (k-ARQ problem)**.** *A table T* $\{t_1, t_2, \cdots, t_n\}$ *is a set of* m*-dimensional tuples, and each tuple* t_i *is labeled as tp if* t_i *is one of the top* k *tuples, and as fp otherwise. Each tuple* t_i *is also associated with count* c_i*, which indicates how many duplicates of* t_i *is in* T*. k-ARQ*(T, k, p) *returns a split value vector* $SV = \langle sv_1, sv_1, \cdots, sv_d \rangle$*, which defines a set* S *as* $S = \{t_i : t_i[d_j] \leq sv_j\}$*for all* $j, 1 \leq j \leq m$*, that satisfies the following:*

1. $\sum c_i$ *for all* $t_i \in S$=k *and*
2. p =*(the num of* tp *tuples in* S *- the num of* fp *tuples in* S*)*

if such SV *exists. If not, k-ARQ*(T, k, p) *returns a pre-assigned value.*

From this definition, the precision of S is $\frac{p+k}{2k}$, thus p=$\frac{prec}{2k} - k$.

Definition 7 (Subset sum problem)**.** *A is a set of integers $\{a_1, a_2, \cdots, a_{pn}\}$ and sum is also an integer. Subsetsum(A, sum) returns A_s = $\{a_{p1}, a_{p2}, \cdots, a_{pt}\}$, where $\sum_{j=1}^{t} a_{pj} = sum$, if such A_s exists. If not, Subsetsum(A,sum) returns a pre-assigned value.*

Polynomial time verification proof: Given a split value vector $SV = \langle sv_1, sv_2, \cdots, sv_m \rangle$ for k-ARQ(T, k, p), where T contains n m-dimensional tuples, we can construct S in $O(mn)$ for integer comparisons. By verifying the first condition, $|S| = k$ can be checked in $O(n)$, the worst case being $S = T$, and for the same worst case $O(m)$ for the second condition. Thus, the verification of any solution to k-ARQ(T, k, p) is done in $O(nm)$, polynomial time to its input size.

Polynomial time reduction from a Subset sum problem[1]**:** Given a subset sum problem *Subsetsum(A, sum)* with an integer set $\{a_1, a_2, \cdots, a_{pn}\}$, for each a_i in A, a pair (t_i, c_i) is created where tuple t_i, $t_i[d_j] = 1$ if $i = j$ and $t_i[d_j] = 0$ otherwise, and the count c_i of t_i is set to be a_i. A pn-dimensional table T is constructed with all t_i. All tuples are labelled in T as *tp*. Transforming *Subsetsum(A,sum)* into k-ARQ(T, sum, sum) takes polynomial time.

$Subsetsum \Rightarrow k{-}ARQ$**:** Imagine that there is a solution A_s for $Subsetsum(A, sum)$, A_s = $\{a_{p1}, a_{p2}, \cdots, a_{pt}\}$. By the definition of the subset sum problem, $\sum_{j=1}^{t} a_{pj}$=sum. Define SV as $sv_i = 1$ if $a_i \in A_s$, 0 otherwise. By definition of S, tuple t_i generated from a_i will be in S. This S satisfies the first condition because $\sum c_i$ for all $t_i \in S$ = $\sum a_j$ for all a_j in A_s = sum. This S also satisfies the second condition because the number of tp tuples in S is equal to $\sum_{j=1}^{t} a_{pj}$ = sum.

$k - ARQ \Rightarrow Subsetsum$**:** Given a solution SV for *k-ARQ*, then compute the

[1]This reduction technique was inspired by the NP-Completeness proof in [11]

set S and construct a subset of A, A_s, such that a_i is in A_s if $sv_i = 1$. $\sum_{a_i \in A_s} a_i$ = $\sum c_i$ for all $t_i \in S$ = sum. This satisfies the *Subsetsum(A, sum)* as a solution.

The proof above shows that the *k-ARQ* problem is an NP-Complete problem. If the desired results is to solve an optimized version of *k-ARQ*,optimization is possible using p, the precision parameter, or k, the accuracy parameter. Given the same p, as k approaches k_q in the original top-k_q ranking query, the result become more accurate relative to the original result of the top-k_q ranking query. There are only k_q number of candidates for p values, as $1 \leq p \leq k_q$ naturally. Thus, the optimization problem over p is also NP-Complete. Similarly, there are only a limited number of candidates for k value, as $k_p \leq |S| \leq k_p/prec$. As a result,the optimization problem of *k-ARQ* over k and p is also an NP-Complete Problem.

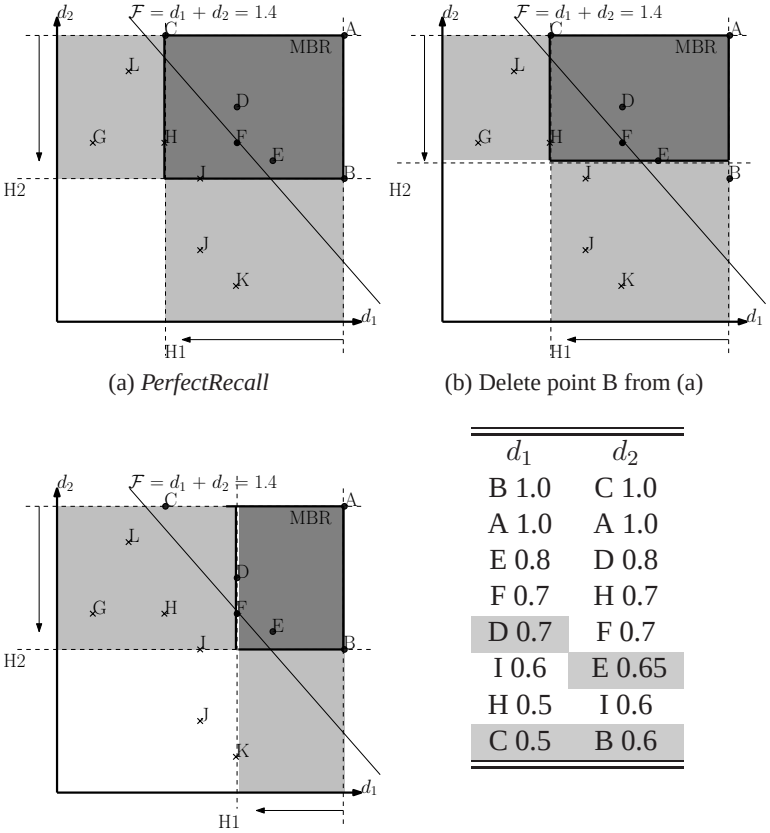
4 Proposed Solution

This section proposes algorithms to address our proposed problem in Section 2.1. As an exact solution has proven to be NP-complete, we propose the use of two Greedy Approximation Algorithms in Sections 4.1 and 4.2.

4.1 *Greedy Deletion* algorithm

The *Greedy Deletion* algorithm first initializes the result set as the MBR of true positives (*i.e.*, the output of *PerfectRecall*) and Greedy improves its precision toward the optimal value by reducing the boundary on one dimension at a time, until any more deletions would result in a violation of the privacy requirement k_p or reduce the precision. Intuitively, a desirable reduction would eliminate as many false positives as possible while eliminating as few true positives as possible. Our *Greedy Deletion* algorithm chooses a true positive tuple that has the largest $\Delta_{fp} - \Delta_{tp}$ when removed from the result set, where Δ_{tp} is the number of eliminated true positives and Δ_{fp} is the number of eliminated false positives. To assist in the calculation of $\Delta_{fp} - \Delta_{tp}$, we utilize an ordered list (index) returned by the top-k algorithms.

For example, Fig. 7(a) shows the initial set of results which corresponds to an MBR for true positives. At the first iteration, the *Deletion* algorithm may choose B with the lowest d_2 value or C with the lowest d_1 value. The outcome of each deletion is shown in Fig. 7(b) and (c), respectively. $\Delta_{fp} - \Delta_{tp}$ is calculated by using the sorted lists shown in Fig. 7(d). B is at the bottom of the sorted list for d_2 and the *Deletion* mechanism traverses the list up until the next true positive E. There is one false positive I between B and E, so $\Delta_{fp} - \Delta_{tp}$= 1 − 1 = 0. Similarly, there is H and I between C and D, so $\Delta_{fp} - \Delta_{tp}$ for C is 2 − 1 = 1. The *Deletion*



(a) *PerfectRecall*

(b) Delete point B from (a)

(c) Delete point C from (a)

d_1	d_2
B 1.0	C 1.0
A 1.0	A 1.0
E 0.8	D 0.8
F 0.7	H 0.7
D 0.7	F 0.7
I 0.6	E 0.65
H 0.5	I 0.6
C 0.5	B 0.6

(d) Ordered list for fast $\Delta_{fp} - \Delta_{tp}$ calculation

Figure 7: *Greedy Deletion* algorithm from *PerfectRecall*

mechanism chooses C to remove it from the result set and the sorted list is then updated accordingly. The pseudo-code of the *Deletion* algorithm is shown below.

4.2 *Greedy Insertion* algorithm

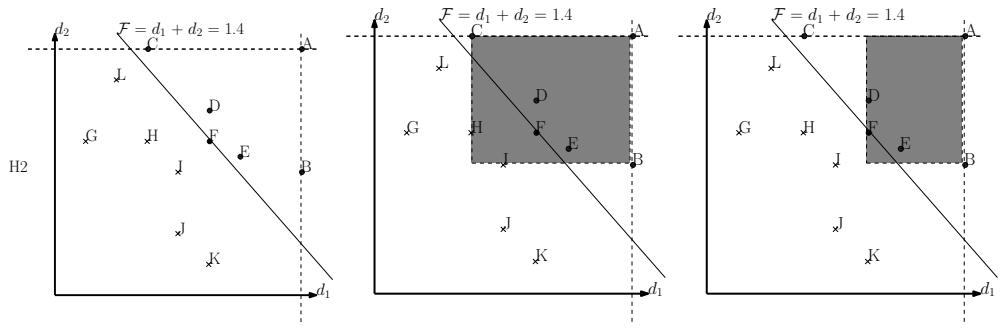
An alternative Greedy algorithm, called the *Insertion algorithm*, starts with an empty set and inserts the true positive tuple with the largest $\Delta_{tp} - \Delta_{fp}$, breaking ties with ranking, until any more addition would render the precision below the

Algorithm 1 *Greedy Deletion* algorithm

Input: A dataset D divided into tp and fp with m ranking attributes, and user-specified requirements k and p as specified in 6.

Output: The result set RS, MBR of RS

1: Obtain a sorted list for each dimension in descending order from top-k ranking algorithm
2: $RS_t \Leftarrow$ all top-k tuples and false positive tuples in MBR of top-k tuples
3: **while** $|RS_t| \leq k$ and precision of $RS_t \geq p$ **do**
4: traverse sorted list for each dimension
5: $candidate \Leftarrow tp$ with the largest Δfp - Δtp that is not in RS_t
6: $RS_t \Leftarrow RS_t \setminus \{candidate\}$
7: $RS \Leftarrow RS_t$
8: update MBR based on true positive tuples in RS and update RS
9: **end while**
10: **if** MBR of RS satisfy p and k **then**
11: return RS, MBR
12: **end if**



(a) After 2 iterations (b) Insert point C (c) Insert point D

Figure 8: *Greedy Insertion* algorithm

basic requirement.

In our example in Fig. 8(a), after the first iteration, MBR contains only A = $(1.0, 1.0)$. At the second iteration, as all true positives have the same $\Delta_{tp} - \Delta_{fp}$, we pick B=$(1.0, 0.6)$ with the highest ranking. Now the MBR extends to $[1.0, 1.0], [0.6, 1.0]$ and the precision is still 1. At the third iteration, among true positives C,D,E and F, we pick D with the largest $\Delta_{tp} - \Delta_{fp}$. To illustrate, adding C to the result set will lead to the in MBR shown in Fig. 8(b) with $\Delta_{tp} - \Delta_{fp}$=

$|\{C, D, E, F\}|$ - $|\{H, J\}|$ = 2 , while adding D will result in the MBR shown in Fig. 8(c) with $\Delta_{tp} - \Delta_{fp}$= $|\{D, E, F\}|$ - 0 = 3.

5 Extension of Greedy algorithms

This section proposes local search algorithms to return the results with the higher success ratio than *Greedy* algorithms by applying local search algorithm. The Section 5.1 introduces the *Grasp* algorithm proposed in [18]. Section 5.2 and 5.3 improve the *Greedy* algorithms using the concepts of the *Grasp* algorithm.

5.1 *Grasp* Algorithm

Grasp was proposed to solve hard optimization problem that involves a large number of alternatives in [18].

Definition 8 (Optimization Problem)**.** *The problem of finding the best solution from all feasible solutions, given constraints that define which of the solutions are feasible, and a goal function defining which of the feasible solutions is the best one.*

To provide top-k tuples, for this problem, we provide an MBR satisfying ranking requirement k_p. However, to guarantee the quality of the solution, we defined the precision in Section 2.1. Our goal is to find an MBR satisfying the *prec* of an MBR including tuples more than k_p. Namely, an MBR satisfying k_p is a feasible solution and the *prec* is the goal function. As our problem is also the optimization problem, the *Grasp* is viable for our problem.

Basically, *Grasp* consists of two iterative phases, a construction phase and a local search phase. The *Construction phase* generates an initial feasible solution S by choosing one random true positive at a time. Starting from the generated solution, the *Local search phase* explores neighbors to find more successful solution until finding local-maxima. As the *Construction phase* does not guarantee that it will

return local-maxima, applying the *Local search phase* to find a better solution is beneficial. Next, we customize *Grasp* for our problem.

1)*Construction phase*: For this problem, at each construction iteration, we determine the next element to be added by ordering all elements in a candidate list with respect to $\Delta_{tp} - \Delta_{fp}$. Note that in this problem, a element in a candidate list means a true positive, and a candidate is the set of true positives. One of the best candidates is randomly chosen as a next element, but not necessarily the top candidate. The list of best candidates is called the *restricted candidate list* (RCL). Parameter α sets the number of candidates in RCL. When α is 3, the RCL has 3 candidates with the largest $\Delta_{tp} - \Delta_{fp}$. Using RCL allows us to obtain different solutions at each *Grasp* without compromising the power of the greedy.

The choice is iterated until a feasible solution is detected. The *prec* is used to control the quality of the results. In contrast, the k_p is necessary requirement for privacy. We thus regard a solution as feasible only if it satisfies k_p. The pseudo-code of *Construction phase* is shown below.

Algorithm 2 Construction phase

Input: A dataset D, privacy requirements k_p, greedy function f which is $\Delta_{tp} - \Delta_{fp}$
Output: Solution set S

```
1: S ⇐ { }
2: for |S| < k_p do
3:     MakeRCL(RCL,α, f);
4:     Choose a element {e_i} from RCL at random
5:     S ⇐ S ∪ {e_i}
6: end for
7: return S
```

2)*Local search phase*: The *Local search phase* iteratively replaces the current solution with a better solution in the neighborhood of the current solution. It termi-

nates when there is no better solution in the neighborhood of the current solution. For this problem, we use m-n exchange as neighborhood structure N, similar to [18]. An m-n exchange is defined as follows : for all m-tuples in an S, the m tuple is exchanged with an n tuple not in S. For this problem, at each iteration, we explore neighbors by applying an exhaustive search method to find m tuples and n tuples, which make the most precision of solution, until the precision of the solution declines. The pseudo-code of *Local search phase* is shown below.

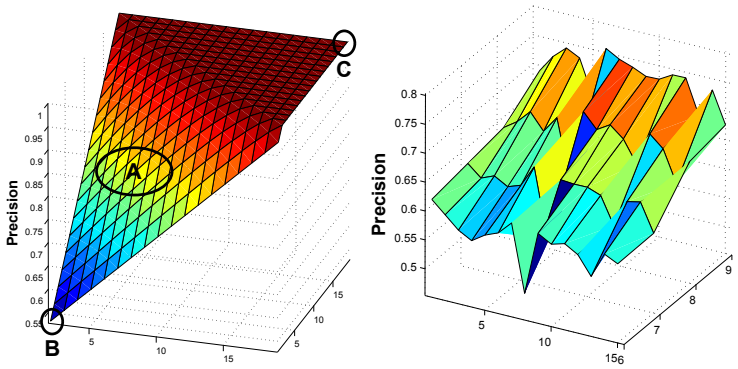
Algorithm 3 Local search phase

Input: A dataset D, a start solution C which is the results of *Construction phase*
Output: Solution set S

```
1: S ⇐ C
2: Max is the solution whose the precision is 0
3: for each m true positive {v_1...v_m} in S do
4:     for each n true positive {w_1...w_n} not in S do
5:         C ⇐ S \ {v_1...v_m}
6:         C ⇐ C ∪ {w_1...w_n}
7:         Max ⇐ Maximum(prec(C),prec(Max));
8:     end for
9: end for
10: if Max satisfies k_p and prec then
11:     return Max
12: else if prec(S) < prec(Max) then
13:     LocalSearch(D, Max);
14: else
15:     return S
16: end if
```

5.2 θ Algorithm

This section proposes the θ algorithm to improve the success ratio when the *Greedy* algorithms fail, using the concepts of *Grasp*. For this problem, starting from the solution S_g of the *Greedy* algorithms in the *Construction phase*, the *Local search phase* of the θ algorithm explores the neighbors, similarly to the *Grasp*.



(a) Whole space (b) Magnification of the region A

Figure 9: *The precision of MBRs*

First, we analyze the search space to provide a hint for the θ algorithm. As the successful solution includes the top-1 tuple in most cases, Fig. 9(a) illustrates the precision of all MBRs including a top-1 tuple on 2-dimensional space, as the search space. For example, a point C shows the precision of the MBR including only the top-1 tuple. Even though the precision of the point C is 1, it does not staisfy k_p. In contrast, the point B indicates the precision of the *PerfectRecall* which is too low.

The *Greedy* algorithms explore the neighbor toward Region A and stop in Region A. The θ algorithm explores neighbors from Region A. Fig. 9(b) is the magnification of Region A, and it provides the precision of the neighbors which mean candidates in the θ algorithm. Even though choosing only the best candidate is useful for searching local-maxima, we does not assure that the local-maxima is a successful solution. In addition, as there are a lot of peaks, which are local-maxima, shown as Fig. 9(b), it is difficult to find the successful solution by choosing only best candidate. Note that the *Grasp* explores the neighborhoods exhaustively to find the local-maxima in the *Local search phase*. In contrast, for our problem, the local-maxima does not guarantee a successful solution. We take bad moves to

avoid the local-maxima by randomly choosing one neighborhood. As mentioned in Section 5.1, as the RCL takes bad moves without compromising the power of the greedy, we apply the RCL in the θ algorithm.

However, as the α controls only the size of RCL, it is used to add the elements with low precision to the RCL. Assume that at iteration i, there are the 5 candidates A, B, C, D and E in the RCL. The precision of each candidate is 0.7, 0.6, 0.5, 0.4 and 0.3 respectively. When α is 3, we choose one of A, B, C randomly. In contrast, at iteration j, there are the 4 candidates F, G, H and I whose precision are 0.7, 0.69, 0.68 and 0.67 respectively. When α is also 3 at iteration j, even though I has higher precision than the C, the RCL does not select I as a candidate.

To solve this problem, we propose new parameter θ to choose candidates with higher precision. For this problem, bounded precision $prec_b$ bounds the minimum precision of the candidates in the RCL. In the above example, if the bounded precision is 0.67, the RCL contains the candidates with precision that is higher or equal to 0.67, such as the I at iteration j. To consider the best candidates in the RCL for each iteration, we define the bounded precision as the difference between the highest precision and θ, and the bounded precision denotes the $prec_b$ = the highest precision - θ.

Finally, we change the terminal condition by applying allowable time t. Basically, the *Local search phase* of the *Grasp* stops when the precision declines. Assume that the $prec_b$ is lower than the precision of the solution at the previous iteration. Even though the RCL has the candidates with higher precision than those at the previous iteration, if a candidate with precision higher than the $prec_b$ and lower than the previous candidate is chosen, the precision of S declines and the

algorithm is stopped. In addition, as mentioned in Fig. 9(b), as there are a lot of peaks, we cannot guarantee that the precision will increase again after it declines. To address this problem, we define the allowable time t as parameter. The allowable time t is a given time for the *Local search phase* of the θ algorithm.We consider that the θ algorithm fails when it does not find a successful solution in allowable time t.

Algorithm 4 θ algorithm

Input: A dataset D, a start solution C which is the solution of the *Greedy* algorithm
Output: Solution set S

```
S ⇐ C
for each m true positive {v_1...v_m} in S do
    for each n true positive {w_1...w_n} not in S do
        C ⇐ S \ {v_1...v_m}
        C ⇐ C ∪ {w_1...w_n}
        InsertToRCL(RCL,θ, f, C)
    end for
end for
Choose C from RCL at random
if satisfy k_p and prec then
    return C
else if allowable time t then
    LocalSearch(D, C);
else
    return C
end if
```

5.3 θ - rapid Algorithm

This section proposes the θ - rapid algorithm by applying a new neighborhood structure to the θ algorithm. The m-n exchange of the θ algorithm has some disadvantages. The large m and n increase the computational time because of the large number of exchanges as candidates. Let S_m be the number of elements in the S. Similarly, let S_n be the number of elements not in the S. The number of candidates at each iteration is $\binom{S_m}{m}$ * $\binom{S_n}{n}$ when using the m-n exchange. Calculating the

precision of $\binom{S_m}{m}$ * $\binom{S_n}{n}$ candidates at each iteration requires considerable computational time. In addition, tuning the m and n is difficult work. In the rest of the paper, we study how to address this problem.

Note that the m-n exchange is a combination of deleting the m tuples and inserting the n tuples. For example, assume that we have a 6 elements such as a, b, c, d, e and f. When S has only a, b and c at i iteration, and both the m and n are 1, there are 9 candidates such as $\{a, b, d\}$, $\{a, b, e\}$, $\{a, b, f\}$, $\{a, c, d\}$ and so on. The $\{a, b, d\}$ is the result of deleting the c in the S and inserting the d to the S.

To reduce the number of candidates, we first define insertion and deletion neighborhoods to reduce the number of candidates. The insertion neighborhoods are the results adding a tuple to the S. In contrast, the deletion neighborhoods are the results deleting a tuple in the S. Both neighborhoods denote N_r for neighborhood structure. As the N_r helps to find the solution of the m-n exchange greedily without considering all combinations of insertion and deletion neighborhoods, the *Local search phase* using N_r chooses best candidates while reducing the number of candidates to S_m + S_n. In the above example, the candidates $\{a, b, c, d\}$, $\{a, b, c, e\}$ and $\{a, b, c, f\}$ are insertion neighborhoods. The candidates $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$ are the deletion neighborhoods. In the θ - rapid algorithm, the *Local search phase* chooses one candidate of the insertion and deletion neighborhoods using the RCL for the allowable time t.

6 Experimental Results

To validate the effectiveness and efficiency of our algorithms, we generated 100 synthetic datasets, each of which consisted of 10000 tuples with 5 ranking attributes. The ranking function $\mathcal{F}$ is the sum of all of the ranking attribute values. Each dataset was tested 100 times. Our algorithms were implemented in C++ using an ODBC connection. All of the performance measurements were performed on a 3.3GHz Intel Dual Core Processor with 2GB RAM running Windows OS. The experiment parameters are summarized below:

Parameter	Result size (k)	precision (*prec*)	dimension	cardinality	θ
Default value	20	0.7	5	10000	0.04

Note that to explore the neighborhoods, the *Grasp* and the θ algorithm used the 1-1 exchange as neighborhoods structure. As the number of the iterations influences the performance of the algorithms, the θ algorithm and θ-rapid algorithm change the terminal condition into the number of iterations, instead of allowable time t. For a fair comparison, the number of iterations in the θ algorithm is 3 which is the average of the number of iterations in the *Grasp*. In contrast, as the θ - rapid algorithm needs (m+n) times iterations than the θ algorithm to find the solution of the m-n exchange greedily, we set 6 as the terminal condition of the θ - rapid algorithm. Finally, the θ algorithm and θ-rapid algorithm use the *Greedy Deletion* algorithm as the *Construction phase*.

To validate the effectiveness of the *Greedy* algorithms, Fig. 10 compares the *Greedy* algorithms with baseline algorithms in terms of the success ratio. Similarly,

Fig 11 displays the computational time of the *Greedy* algorithms and baseline algorithms. In addition, we study how the θ algorithm and θ-rapid algorithm improve the success ratio of the *Greedy* algorithms as displayed in Fig. 12. Finally, Fig 13 shows computational time in the *Local search phase* of each algorithm to show the efficiency.

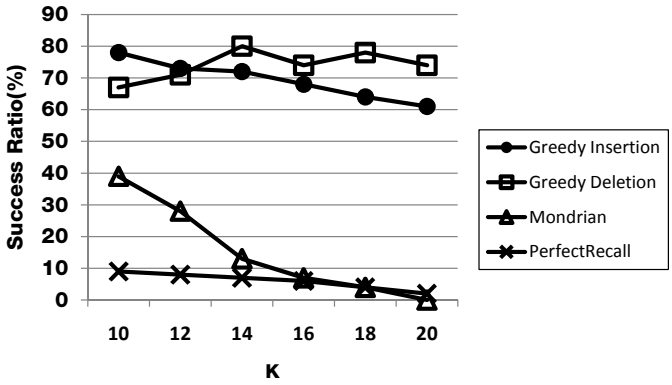


Figure 10: The success ratio of the greedy algorithms

Fig. 10 compares our *Greedy* algorithms with *Perfect Recall* and *Mondrian* in terms of the success ratio as the ranking requirement k increases. When a user requests a top-k query, each algorithm may not be able to satisfy both k and p. For each k, the result is an average from 100 executions. Our *Greedy* algorithms succeed far more times than *Perfect Recall* and *Mondrian*. This is expected in the case of *Mondrian* as it does not consider the ranking of tuples in partitioning and suffers from the lowest success rate. *PerfectRecall* performs better when k is small, but still only succeeds in about half the number of times when compared to our *Greedy* algorithms. When k becomes as large as 20, *PerfectRecall* always fails to satisfy p.

Fig. 11 compares the efficiency time of our algorithms to others by measuring the execution time as k increases. Since *PerfectRecall* gets its MBR from the top-k ranking algorithm, it would not incur any extra cost. For this reason, Fig. 11 does

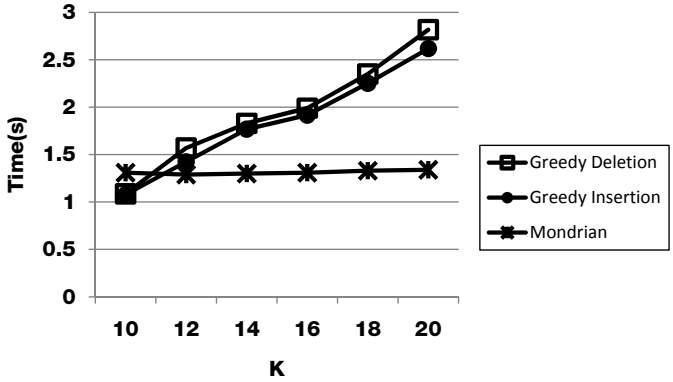


Figure 11: The computational time of the greedy algorithms

not depict the computational time of *PerfectRecall* but one can imagine this being 0. Each data point is from an average of 100 executions. Since $Mondrian$ divides data into equal-sized datasets, the computational time is not affected by k. On the other hand, our *Greedy* algorithms need more iterations as k increases and therefore, our execution time increases fast. However, note that this k is limited by the privacy requirement k_p, which usually is much smaller than k_q. For example, if a user requests the top-100 queries with 10-anonymity, our execution time is dominated by 10 and, not by 100. To be precise, it takes less than 1.5 seconds to compute a result set of this type. As future work, we plan to experiment with more heuristics and realistic k_p and k_q values.

Fig. 12 shows the success ratio of our algorithms, θ algorithm and θ-rapid algorithm, to compare with the *Grasp* and *Greedy* algorithm. To find the best success ratio, we tuned the parameters of each algorithm. As the *Grasp* executes the *Local search phase* using the m-n exchange, it has a higher success ratio than the *Greedy Deletion* algorithm. The θ algorithm and θ-rapid algorithm succeed more times than *Grasp*, and they improve the *Greedy Deletion* algorithm in terms of the success ratio. In particular, the gap between the θ algorithm and the *Greedy Deletion* in terms

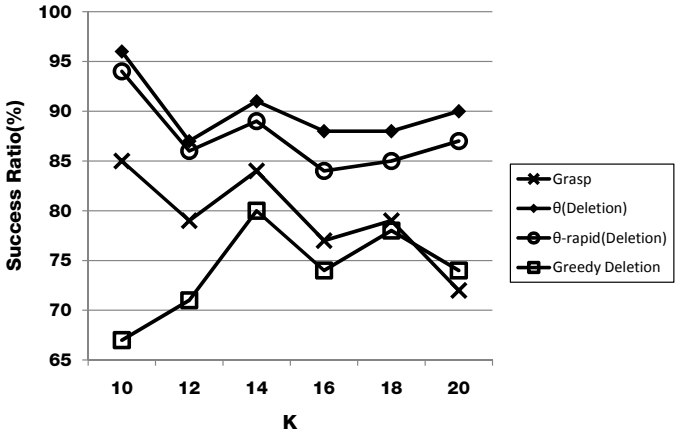


Figure 12: The success ratio of the local search algorithms

of the success ratio is more than 10%. In addition, the success ratio of the θ algorithm is never lower than 87 times out of 100 executions. Finally, as the θ - rapid algorithm finds the solution using the m-n exchange greedily, the gap between the success ratio of the θ algorithm and the θ - rapid algorithm is never lower than 4%.

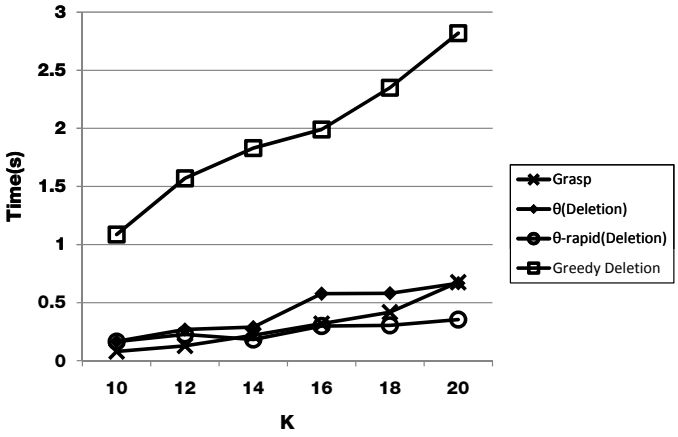


Figure 13: The computational time of the local search phase

Fig. 13 shows the time in the *Local search phase* of each algorithm, *Grasp*, the θ algorithm and the θ - rapid algorithm, as k increases. As the θ algorithm needs more cost than *Grasp* because of the RCL, it takes longer than *Grasp* in the *Local search*

phase. In contrast, as the θ - rapid algorithm reduces the number of candidates, even though it uses the RCL similarly to the θ algorithm, the computational time is shorter than *Grasp*. In addition, as the computational time is almost not effected by k, the gap between the θ - rapid algorithm and *Grasp* is larger, as k increases. In particular, when k is 20, the θ - rapid algorithm is executed with the half the computational time of *Grasp*.

7 Related Work

Ranking queries have been actively studied, as an effective means to narrow down to a relevant data subset in large-scale repositories [1, 2, 3, 4]. Most existing algorithms focus on optimizing the computation of such results and reveals all the information for the top-k results, *i.e.*, identifiers, attribute values, and rank ordering, which may include sensitive data on each individual.

To protect privacy, the notion of k-anonymity [5] has been introduced, to publish an "anonymized" table with the assurance that no individual can be uniquely distinguished from $k-1$ other tuples. In order to move toward this direction of publishing anonymized tables, the problem is proved to be NP-hard problem for $k > 3$ in [10], followed by efficient approximation partitioning algorithms algorithms [11, 12] and clustering algorithms [14, 15]. The limitations of k-anonymity were discussed and mitigated in l-diversity [6], m-invariance [7], t-closeness [8], and also in variations of k-anonymity such as (α, k)-anonymity [16] and (p+, α)-anonymity [17].

A straightforward solution to our proposed problem with k_p and k_q requirements is to apply these anonymization algorithms over top-k_q tuples returned by top-k query algorithms such as TA-family algorithms [1, 2, 3, 4]. We use one of the algorithms, Mondrian [11], as part of our performance evaluation as a reference point. However, there are limitations in this basic application. Most approximation algorithms assume that there are many more tuples than k_p and use some form of partitioning to ensure k_p-anonymity, so when the input size k_q is already close to k_p, they either cannot partition the input and fail to achieve k_p-anonymity, or have to include too many tuples that are not in the top-k_q results and sacrifice data quality as discussed in Section 2.

To pursue the dual requirements of the privacy and ranking accuracy, the privacy-preserving top-k query algorithm [9] returns the exact top-k results but publishes no other attribute values. For example, if a user wants to find top-100 most popular search keywords is, then the algorithm returns the 100 search keywords but no other information such as how many searches were done for each keyword or in, which region each keyword. Since this algorithm does not publish any attributes, it is very difficult to infer any meaningful statistics from the results, *e.g.*, attributes' sensitivity to rankings or attribute correlations.

The *Grasp* algorithm has solved the hard optimization problem that involves a large number of alternatives in [18]. *Grasp* consists of two iterative phases, a construction phase and a local search phase. The *Construction phase* generates an initial feasible solution S by choosing one random true positive at a time. Starting from the generated solution, the *Local search phase* explores neighbors to find a successful solution by replacing the current solution until finding local-maxima.

References

[1] Fagin, R., Lote, A., Naor, M.: Optimal aggregation algorithms for middleware. In: PODS. (2001)

[2] Guentzer, U., Balke, W., Kiessling, W.: Optimizing multi-feature queries in image databases. In: VLDB. (2000)

[3] Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. In: ICDE. (2002)

[4] Hwang, S., Chang, K.C.C.: Optimizing access cost for top-k queries over web sources: A unified cost-based approach. In: ICDE. (2005)

[5] Sweeney, L.: K-anonymity:a model for protecting privacy. In: International Journal of Uncertainty Fuzziness and Knowledge-based Systems. (2002)

[6] Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: Privacy beyond k-anonymity. ACM TKDD **1**(1) (2007) 3

[7] Xiao, X., Tao, Y.: M-invariance: towards privacy preserving re-publication of dynamic data sets. In: the 2007 ACM SIGMOD. (2007)

[8] Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: ICDE. (2007)

[9] Vaidya, J., Clifton, C.: Privacy-preserving top-k queries. In: ACM SIGMOD. (2005)

[10] Muthukrishnan, S., Poosala, V., Suel, T.: On rectangular partitionings in two dimensions. In: ICDT. (1999)

[11] LeFevre, K., DeWitt, D.J., Ramakrishnan, R.: Mondrian multidimensional k-anonymity. In: ICDE. (2006)

[12] Samarati, P., Sweeney, L.: Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. In: IEEE S&P. (1998)

[13] Ahn, S., Hwang, S.W., Jung, E.: k-anonymous ranking queries. Technical Report TR09-04, Dept. of Computer Science, The University of Iowa, September 2009

[14] Aggarwal, G., Feder, T., Kenthapadi, K., Khuller, S., Panigrahy, R., Thomas, D., Zhu, A.: Achieving anonymity via clustering. In: PODS (2006)

[15] Miller, J., Campan, A., Truta, T.M.: Constrained k-anonymity:privacy with generalization boundaries. In: P3DM. (2008)

[16] Wong, R.C.W., Li, J., Fu, A.W.C., Wang, K.: (alpha, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing. In: the 12th ACM SIGKDD. (2006)

[17] Sun, X., Wang, H., Li, J., Truta, T.M., Li, P.: (p+, α)-sensitive k-anonymity: a new enhanced privacy protection model. In: the 8th IEEE ICCIT. (2008)c

[18] Mauricio G. C. Resende., CELSO C. RIBEIRO.: Greedy Randomized Adaptive Search Procedures. In: Encyclopedia of Optimization(2009)

Curriculum Vitae

Name : SukHyun An

Education

2005.3 - 2009.2 : Computer Science, Soongsil University (B.S.)

2009.3 - 2011.2 : Computer Science, Pohang University of Science and Technology (M.S.)