UTILITY-PRESERVING k-ANONYMITY

RHONDA CHAYTOR

Utility-Preserving k-Anonymity

by

© Rhonda Chaytor

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

October 2006

St. John's Newfoundland

Abstract

As technology advances and more and more person-specific data like health information becomes publicly available, much attention is being given to confidentiality and privacy protection. On one hand, increased availability of information can lead to advantageous knowledge discovery; on the other hand, this information belongs to individuals and their identities must not be disclosed without consent. A recently proposed concept called k-Anonymity addresses this conflict between doctor-patient confidentiality and society-beneficial research.

Several k-Anonymity-based problems have been proposed in the literature; however, these problems do not adequately address preserving utility for the researcher and their algorithms are not computationally efficient. This thesis highlights these inadequacies through a comprehensive overview of previous research, where it is shown that previous solutions lack sufficient ability to meet specific researcher needs. To this end, new utility-preserving problems are proposed and their computational complexities are analyzed. Many results for k-Anonymity-based problems are systematically derived through this analysis, including two of particular interest: (1) the first known polynomial-time solvable k-Anonymity-based problem and (2) the first known algorithm-independent polynomial-time approximation intractability results for k-Anonymity-based problems.

Acknowledgments

First and foremost I would like to thank Drs. Todd Wareham, Ph.D., and Gerard Farrell, M.D., for taking me on as a graduate student and sharing invaluable computational complexity and medical knowledge. The time and effort spent helping me with my program, course work, proposals, applications, presentations, etc., is beyond what is expected.

I also appreciate the suggestions and further insight I have received from my thesis reviewers, Drs. Cao An Wang and Staal Vinterbo, and from members of MUN Medical Informatics Interest Group (MIG) and Computational Privacy Interest Group (CPIG). Besides computational and medical aspects of information privacy, I have also had the opportunity to discuss the legal aspects of my problem with one member of both interest groups, Dr. Edward Brown, Ph.D., LL.B.

I would also like to thank NSERC for scholarship funding and faculty members of the Computer Science department for writing reference letters for various applications: Drs. Banzhaf, Vidyasankar, and Zuberek. I need to thank the administrative and technical staff who have always been there to answer my questions or fix something that was broken - Elaine, Sharon, Darlene, Regina, Dwayne, Paul, Mike, Nolan, and Marian. These people genuinely care about the students in this department.

I would like to thank my family and friends for understanding my decision to pursue a graduate degree and why visits home are shorter and less frequent.

Finally, I would like to thank Mark, Gretta, and Gabe, for their constant love, support, and patience during the writeup of this thesis.

Contents

List of Tables

List of Figures

List of Notations

Set Theory

$\in$	is an element of	11
$\notin$	is not an element of	41
$\mathbb{R}$	the set of real numbers	11
$\mathbb{Q}$	the set of rational numbers	17
$\mathbb{N}$	the set of natural numbers	23
$\lvert A\rvert$	the cardinality or size of a set A	13
$\subseteq$	is a subset of	34
$\subset$	is a proper subset of	20
$\cup$	union	37
$\cap$	intersection	121
$\times$	Cartesian product	11
$\setminus$	set difference	35

Graph Theory

$G = (V, E)$	G is a graph with vertex set V and edge set E	11
$e = \{u, v\}$	e is an edge that joins the two vertices u and v	83

$w(e)$	weight of edge e	118
K_n	complete graph on n vertices	9

Complexity Theory

A, B, C	computational problems	11
e	encoding scheme	11
n	instance length	11
Σ	alphabet	11
Σ^*	set of all finite strings of symbols from alphabet Σ	11
DTM	deterministic Turing machine	14
NDTM	nondeterministic Turing machine	18
$O(g(n))$	asymptotic upper bound (*Big Oh*)	15
$\log n$	logarithm to the base 2	20
PTAS	polynomial-time approximation scheme	17
FPTAS	fully polynomial-time approximation scheme	17
OPT_A	cost of any optimal solution for a given instance of optimization problem A	15
c_A	cost of an arbitrary feasible solution for a given instance of optimization problem A	15
P	class of decision problems that are solvable in deterministic polynomial time	18
NP	class of decision problems that are solvable in nondeterministic polynomial time	18

EXPTIME	class of decision problems that are solvable in exponential time . . 19
OptP	class of optimal-cost evaluation problems that are solvable in nondeterministic polynomial time . . . 19
OptP$[O(\log n)]$	subclass of OptP problems that have solutions which are polynomially-bounded by their instance lengths . . . 19
PO	class of optimization problems that are solvable in deterministic polynomial time . . . 19
NPO	class of optimization problems that are solvable in nondeterministic polynomial time . . . 19
APX	class of NPO problems that are approximable within a constant factor . . . 19
log-APX	class of NPO problems that are approximable within a logarithmic factor . . . 20
poly-APX	class of NPO problems that are approximable within a polynomial factor . . . 20
exp-APX	class of NPO problems that are approximable within an exponential factor . . . 20
$\propto$	reduces to . . . 20
$\leq_m$	many-one reduces to . . . 22
$\leq_L$	L-reduces to . . . 23
RHS	right-hand side . . . 31
LHS	left-hand side . . . 31

k-Anonymity

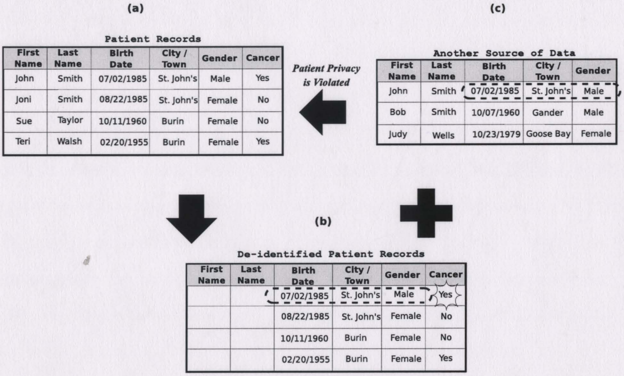
k	number of identical people required to satisfy k-Anonymity	35
T	$n \times m$ database table	34
n	number of rows in database table T	34
m	number of columns in database table T	34
e_{ij}	the j^{th} attribute of person i in database table T	34
Σ	alphabet, i.e., all possible values for entry e_{ij}	34
Σ^m	all possible values for database table T	34
$*$	suppressed entry	41
$\top$	indiscernible entry	45
e	number of entries of T which may be suppressed	41
c	number of columns of T which may be modified	42
r	number of rows of T which may be modified	59
c'	number of entries per column of T which may be suppressed	59
r'	number of entries per row of T which may be suppressed	59
ε	the empty string	63
P	partition of database table T either into rows or columns	55
$\preceq_{c^\star}$	total order on column partition P	55
G	k-partition	54
g_i	k-group from k-partition G	54
s_{ij}	the j^{th} k-segment of k-group g_i	54
X	set of rows (people) in database table T	34
x_i	member from set of people X	34
A	set of columns (attributes) in database table T	34

Chapter 1

Introduction

One of the most important promises a physician makes to a patient is that of confidentiality. Therefore, whether on paper or recorded electronically, patients expect their personal health information remain confidential. At the same time, there is a conflicting need to release this information for health research and surveillance. However, today in our growing digital society, guaranteeing patient privacy while providing researchers with worthwhile data has become increasingly difficult.

Figure 1.1(a) is a simplified example of a set of digitized patient records for a particular region. Suppose it is desirable to make a public release of these records for clinical, epidemiological, and health administration research purposes. The original data set cannot be publicly released without consent, since in doing so, patient privacy is compromised. In the past, it was believed that removing obvious identifiers like social security number and name would be sufficient in the protection of patient privacy. Figure 1.1(b) illustrates this *de-identification* process. Recent studies, however, indicate that it is possible to *re-identify* individuals, even if the data set is



Patient Records

First Name	Last Name	Birth Date	City / Town	Gender	Cancer
John	Smith	07/02/1985	St. John's	Male	Yes
Joni	Smith	08/22/1985	St. John's	Female	No
Sue	Taylor	10/11/1960	Burin	Female	No
Teri	Walsh	02/20/1955	Burin	Female	Yes

Another Source of Data

First Name	Last Name	Birth Date	City / Town	Gender
John	Smith	07/02/1985	St. John's	Male
Bob	Smith	10/07/1960	Gander	Male
Judy	Wells	10/23/1979	Goose Bay	Female

De-identified Patient Records

First Name	Last Name	Birth Date	City / Town	Gender	Cancer
		07/02/1985	St. John's	Male	Yes
		08/22/1985	St. John's	Female	No
		10/11/1960	Burin	Female	No
		02/20/1955	Burin	Female	Yes

Figure 1.1: Example of Data Linkage After De-identification. (a) A simplified example of a set of digitized patient records for a particular region. (b) The resulting set of records after removing obvious identifiers (de-identification). (c) Another information source (*e.g.*, voter registries, insurance policies, and telemarketing profiles).

de-identified. Sweeney's research, for example, shows the ease with which a medical record can be linked to an individual using little more then a zip code, date of birth, and gender [40]. Sweeney was able to pinpoint the governor of Massachusetts' medical record using publicly available medical records and a voter registry (six people in the state share his birth date, only three of them are male, and he is the only one in his 5-digit ZIP code). Figure 1.1(b)-(c) shows how re-identification is accomplished by linking records containing unique data values to another information source (*e.g.*, voter registries, insurance policies, and telemarketing profiles). Because John Smith is the only male from St. John's born on July 2nd, 1985, his patient record can be re-identified and his private medical information consequently violated.

To understand the serious consequences of personal information collection and linkage, consider a recent shocking story told by UK journalist Steve Boggan [9]. He picked an airline boarding-pass stub out of a garbage bin near Heathrow and was able to commit a serious breach of privacy. From that small piece of paper, Boggan knew the passenger's name, flight plan, boarding date and time, flight number, seat number, and his gold member frequent-flyer number. The frequent-flyer number alone provided access to all the man's personal details, including his passport number, the date it expired, his nationality, and his date of birth. Using the frequent-flyer information, within 15 minutes of looking at publicly available databases, Boggan found out where the man lived, who lived there with him, where he worked, which universities he had attended and even how much his house was worth when he bought it two years ago.

As more and more reports are relayed regarding identity theft and the misuse of personal information, such as the one above, people are becoming increasingly paranoid and angry. They demand a solution. One such solution might be to determine which identifiers are possible linking agents and remove those identifiers in the de-identification process as well. However, this solution is not viable; there would be no data left for research, which was the reason for releasing the data in the first place. Researchers agree that some other privacy protection technique is required. There is active research in privacy-preserving data mining, where data values are perturbed by methods like adding noise and swapping values, while ensuring the statistical properties of the data remain intact (*e.g.*, [4]). Other data mining researchers attack the problem from a different angle; they privately compute the sought-after statistical measure without releasing the data at all (*e.g.*, [10]). Solutions such as these work very well in certain contexts, but fail in others. For example, researchers may want to perform correlational data analysis to explore relationships between patient age, gender, and disease. In this case, the validity of the resulting data values is mandatory and it may not be known in advance which statistics are required.

When a release of data, not a single aggregate value, is required, the approach adopted is to make an individual patient *anonymous* with respect to the entire release, thereby ensuring that no information is distinctive to a particular individual. A technique has recently been proposed called k-Anonymity [39], in which released data is made less specific, yet remains truthful (unlike the data releases from privacy-preserving data mining). k-Anonymity is not a perfect concept, making it ideal to study. In particular, current k-Anonymity-based problems are NP-hard [3, 31] and they lack sufficient ability to meet specific researcher needs (see Section 3.2).

The following is a summary of the contributions of this thesis:

- Section 2.2 summarizes previous and related k-Anonymity work. It can be difficult in this work to distinguish between the original proposed problem and easier-to-solve versions. This thesis uses Sweeney's [39] problem definitions for k-Anonymity as benchmarks and gives a unified presentation of all versions of k-Anonymity research. In this way, the summaries of previous and related work found herein use a universal set of notation and definitions, so that they may be used as concrete references for future work.

- Chapter 3 proposes new utility-preserving k-Anonymity-based problems. Following existing complexity-theoretic analyses (*e.g.*, [31, 3]), we define our problems using suppression only, rather than suppression and generalization (see Section 2.2.2 for details). Moreover, our conception of utility differs from all previous conceptions in that we believe researchers should be allowed to specify what portions of the data have the greatest utility in their research and hence cannot be altered during the anonymization process. For example, gender might be crucial to a researcher's study, and any release of data that deletes gender values in order to satisfy k-anonymity would not be useful to this researcher. Several approaches to preserving utility under k-anonymity have been proposed [39, 23, 42, 35, 20]; however, all of these proposals are inadequate, because they either cannot tailor output to the full range of researcher needs (*e.g.*, [39]) or are too complicated for practical application (*e.g.*, [42]). To overcome inadequacies in previous work, the proposed utility-preserving k-Anonymity-based problems better capture the trade off between ensuring patient privacy and providing researchers with worthwhile data. These problems form a related set of problems called a *family* (see Section 2.1.4.2 for details).

- Chapters 4 and 5 summarize computational complexity results derived for the utility-preserving k-Anonymity family of problems mentioned above. Two results of particular interest are the first known polynomial-time solvable k-Anonymity-based problem (Section 4.3) and the first known algorithm-independent polynomial-time approximation intractability results for k-Anonymity-based problems (Section 5.2).

These results are all derived within a new family-oriented systematic analysis framework. Our analysis involves identifying basic problems, creating template reductions, and forming a web of reductions. This web is structured to permit multiple types of complexity results for selected problems to propagate through the family, allowing new results to be obtained with minimal additional effort.

This thesis is organized as follows. Chapter 2, **Background**, consists of two sections that provide overviews of the main themes of this thesis: Complexity Theory and k-Anonymity. This chapter also proposes a framework for analyzing a family of problems and reviews useful mathematical definitions for sets and graphs. Chapter 3, **Utility-Preserving k-Anonymity**, marks the beginning of new material. First, the motivation for studying utility-preservation is given and previous work on utility is summarized and critiqued. The last section of this chapter then defines a new family of utility-preserving k-Anonymity problems (individual problem definitions are given in Appendix A). Chapter 4, **Optimal Solutions**, uses material from Chapters 2 and 3 to explore the computational complexity of finding optimal solutions for these newly defined problems. The content in Chapter 4 is organized in the following sequence: problems, reductions, classes, tractability/intractability. Notice how this differs from traditional procedure for analyzing complexity, *i.e.*, problems, tractability, classes, reductions, and intractability; the apparent backwards nature of Chapter 4 is due the proposed framework for analyzing problem families, mentioned above. Chapter 5, **Approximate Solutions**, contains results for the computational complexity of approximate solutions. The reusability of the previously mentioned analysis framework becomes apparent in this chapter, as the majority of results are by-products of those derived in Chapter 4. Chapters 4 and 5 both conclude with brief discussions of the implications of the acquired results and directions for future research. Finally, Chapter 6, **Conclusions**, summarizes the major contributions of this work and suggests several additional directions for future research.

Chapter 2

Background

This chapter reviews the two major themes of this thesis: Computational Complexity and k-Anonymity. Section 2.1 gives a side-by-side summary of concepts from traditional complexity theory and approximation complexity theory, and provides a new framework for analyzing the computational complexity of a family of related problems. The fundamentals of k-Anonymity, as well as previous and related work on the topic, are discussed in Section 2.2.

Standard mathematical notations used in this thesis are included in the List of Notations on pages x to xiv. Basic definitions pertaining to sets and graphs (see [18]) are reviewed below.

- **Sets:** Set $A = \{a_1, a_2, \ldots, a_n\}$ is a collection of n objects called **elements** or **members**. If sets A and B have no common elements, then A and B are **disjoint**. A **partition** of a set A is a collection of disjoint nonempty subsets of A whose union is A. A **partial order** on a set A is a reflexive, antisymmetric, transitive relation on A (see [18, Section 2.5] for a more detailed definition). A

partially ordered set is a pair $(A, \preceq)$, where $\preceq$ is a partial order on a set A. The symbol $\preceq$ is similar to $\leq$, and as such, the expression "a_i is less than or equal to a_j" is used whenever $a_i \preceq a_j$. If $(A, \preceq)$ is a partially ordered set, elements a_i and a_j of A are said to be **comparable** if and only if either $a_i \preceq a_j$ or $a_j \preceq a_i$. If $\preceq$ is a partial order on a set A and every two elements of A are comparable, then $\preceq$ is called a **total order** and the pair $(A, \preceq)$ is called a **totally ordered set**.

- **Graphs:** A graph G_1 is a **subgraph** of another graph G if and only if the vertex and edge sets of G_1 are, respectively, subsets of the vertex and edge sets of G. If there is an edge between every pair of vertices in G_1, then G_1 is a **clique**, *i.e.*, a **complete graph** on n vertices K_n.

  A **directed graph** is essentially a graph in which each edge has a direction assigned to it, *i.e.*, $\{u, v\}$ is an edge which is directed away from vertex u and towards vertex v. These directed edges are called **arcs**. The number of arcs directed towards a vertex is called the **indegree** and the number of arcs directed away from a vertex is called the **outdegree**.

  Vertices u and v are said to be **incident** to the edge/arc $\{u, v\}$. Two edges are incident if they share a common vertex. Two arcs are incident if they share a common vertex and orientation is respected, *i.e.*, one arc is directed towards the common vertex and the other arc is directed away from the common vertex.

  A **walk** is an alternating sequence of vertices and edges/arcs, beginning and ending with a vertex, in which each vertex (except the last) is incident with the edge/arc which follows and the last edge/arc is incident with the edge/arc which precedes it. A **path** is a walk in which all vertices are distinct. A **cycle** is a walk in which all edges/arcs are distinct, the first vertex appears exactly twice (at the beginning and the end), and no other vertex appears more than once. A graph is **connected** if and only if there exists a walk between any two vertices. A **component** is a maximal connected subgraph of a graph, *i.e.*, a connected subgraph which is properly contained in no other connected subgraph that has more vertices or edges/arcs. A connected graph which contains no cycles is called a **tree** and the disjoint union of a set of trees is called a **forest**.

2.1 Complexity Theory

The majority of complexity theory found herein is from [16, 34], while specific material on approximation theory comes from [7]. The content is organized in the usual sequence: problems, algorithms, tractability, classes, reductions, and intractability. The curious reader should observe that this sequence is not adhered to in Chapters 4 and 5. The apparent backwards nature of those chapters is justified in Section 2.1.4.2.

2.1.1 Computational Problems

Computational problems [16, Section 1.2] are essentially questions possessing several **aspects**, whose values are left unspecified. Depending on how the question is asked, there are different types of answers, *e.g.*, logical, numeric, mathematical structure, and hence different types of problems (see below). A **problem definition** includes a general description of all the problem's aspects (the **input**) and a statement of what the **solution** is required to satisfy (the **output**). A particular problem **instance** is obtained by specifying values for the problem's aspects.

Every instance of a problem A can be formally mapped into a string x describing that instance using an **encoding scheme** e over some fixed **alphabet** Σ. This encoding scheme should be *reasonable*, *i.e.*, it should be compact and easily decodable [16, page 21]. For example, a graph $G = (V, E)$ can be represented by a structured string (x, y), where x is a structured string representing the set V and y is a structured string representing the set E. The total number n of symbols in the string representation of x is called the **instance length** of x.

As mentioned briefly above, different types of problems exist, depending on the nature of their required solutions. To define the four types of related problems in this thesis, let S be a relation $S : \Sigma^* \times \Sigma^*$ on pairs of objects that underlies the related problems, *e.g.*, *graphs* $\times$ *cliques*, let P be the stem of the name of problem A, let b be a valuation function $b : S \rightarrow \mathbb{R}$, and let the symbol $\bowtie$ be an operator from the set $\{=, \leq, \geq\}$. Computational problems can be viewed as functions defined on the projection of S onto a given element x. Four constraint-type problems are formally described here using Wagner and Wechsung's framework [43, pages 100–101]:

Definition 2.1.1 (Decision Problem) *The solution to* **decision problem** *k-P is defined as the boolean result of evaluating the formula* $\exists\, y\ \{\ (x, y) \in S \wedge b(x, y) \bowtie k\}$.

Definition 2.1.2 (Solution Problem) *The solution set to* **solution problem** SOL-*k-P is defined as* $\{\ y \mid (x, y) \in S \wedge b(x, y) \bowtie k\}$.

Definition 2.1.3 (Optimal-Cost Evaluation Problem) *An* **optimal-cost evaluation problem** *can either be* **min-cost evaluation problem** MIN-COST-*k-P,* *whose solution is defined as*

$$\{\ b(x,y) \mid (x,y) \in S \wedge \forall\ (p,q) \in S\ \{b(x,y) = \mathit{min}\ b(p,q)\}\}$$

or **max-cost evaluation problem** MAX-COST-k-P*, whose solution is defined as*

$$\{\ b(x,y) \mid (x,y) \in S \wedge \forall\ (p,q) \in S\ \{b(x,y) = \mathit{max}\ b(p,q)\}\}.$$

Definition 2.1.4 (Optimization Problem) *An* **optimization problem** *can either be* **minimization problem** MIN-k-P *whose solution set is defined as*

$$\{\ y \mid (x,y) \in S \wedge \forall\ (p,q) \in S\ \{b(x,y) = \mathit{min}\ b(p,q)\}\}$$

or **maximization problem** MAX-k-P *whose solution set is defined as*

$$\{\ y \mid (x,y) \in S \wedge \forall\ (p,q) \in S\ \{b(x,y) = \mathit{max}\ b(p,q)\}\}.$$

Simply stated, a decision problem asks a question that requires either a "yes" or "no" answer based on some constraint, a solution problem asks for solutions that satisfy a constraint, an optimal-cost evaluation problem asks for the best cost for solutions that satisfy a constraint (*i.e.*, the highest or lowest cost), and an optimization problem asks for the best solutions that satisfy a constraint (*i.e.*, those solutions that have the lowest or highest cost).

For example, consider the related problems generated from S : *graphs* $\times$ *cliques*. Let aspect k be the size of a clique, problem name P be CLIQUE, instance x be a graph, output y be a clique, valuation function b count the number of vertices in a clique, and comparison operator $\bowtie$ be greater than or equal to ($\geq$). This generates the following well-known decision problem from graph theory:

k-CLIQUE ([16, GT19])

Instance: An undirected graph $G = (V, E)$ and positive integer $k \leq |V|$.

Question: Is there a clique in G, which has size $\geq k$?

The associated solution (SOL), maximal-cost evaluation(MAX-COST), and maximization (MAX) problems are as follows:

SOL-k-CLIQUE

Instance: An undirected graph $G = (V, E)$ and positive integer $k \leq |V|$.

Solution: Any clique in G, which has size $\geq k$.

MAX-COST-k-CLIQUE

Instance: An undirected graph $G = (V, E)$.

Solution: The size of the largest clique in G.

MAX-k-CLIQUE

Instance: An undirected graph $G = (V, E)$.

Solution: The largest sized clique in G.

The naming conventions introduced here will be used throughout this thesis. It is not a mistake that k is a part of the name of the optimal-cost and optimization problems above; in this thesis, the variable which appears directly after MAX-COST, MIN-COST, MAX, or MIN in the associated problem name implicitly specifies the cost being optimized, since it is the value that valuation function b is being compared against. This is useful notation for Chapter 5, where optimal-cost evaluation and optimization problems can have have more than one related solution problem.

2.1.2 Algorithms

An **algorithm** is a finite sequence of instructions that computes the appropriate solution for any instance of a problem. The computational model of an algorithm adopted in this thesis is the **deterministic Turing machine** (DTM) (see [16, Section 2.2] for an explanation of the operation of a DTM). Essentially, given an instance x of a problem A, the DTM's tape prior to computation corresponds to the string encoding for x. If A is a decision problem and the DTM halts, then the solution for x is "yes"; otherwise the solution is "no". If, on the other hand, A is a solution or optimization problem, then the solution for x is the tape contents after the DTM halts.

As there are many possible algorithms for a particular problem, some method of comparison is required to choose the "best" algorithm for a particular problem. One commonly-used comparison measure is the amount of time it takes the algorithm to solve the problem.

Given a problem A, an instance x of A that has a particular length n, and an algorithm γ for A, the simplest way to measure the running time $T(x)$ is to count the number of instructions γ executes to solve A. Note that this is an abstract view of the exact running time, since in practice not all instructions take the same amount of time to execute. It is possible that some algorithm δ may run faster than γ for x, yet it may run slower for some other instance y, even if $|x| = |y|$. Therefore, the comparison of running times has to be general enough to cover all possible problem instances of a particular length n. To accomplish this, one can choose to compare algorithms based on their fastest, average, or slowest (*i.e.*, **worst**) running time $T(n)$ over all instances of length n, which further diminishes run time exactness. Furthermore, the time complexities of algorithms are compared in this thesis using **asymptotic worst-case time complexity** $O(g(n))$, which is an upper bound on the worst-case running time for any instance length n to within a constant factor as n goes to infinity. For example, given an algorithm with running time $T(n) = 2n^2 + 3n + 1$, as the multiplicative constants and lower order terms are dominated by the highest-order term for large instance lengths, the algorithm's asymptotic worst-case time complexity is $O(n^2)$.

Given an asymptotic bounding function g on the running time $T(n)$ for some instance length n, an **efficient** algorithm is one whose asymptotic worst-case time complexity is $O(g(n))$, such that $g(n)$ is a polynomial (*i.e.*, an algorithm that runs in polynomial time). A decision problem is said to be **tractable** if it has an efficient algorithm; otherwise it is **intractable**. If an efficient algorithm that returns an **optimal solution** for an optimization problem does not exist, it may be acceptable to settle for an efficient algorithm that returns an **approximate solution** instead. To express the quality of such a solution for comparison purposes, there are several *distance-from-optimality* measures (see [7, Section 3.1]). In this thesis, we focus on the following two more commonly-used measure:

Definition 2.1.5 (Absolute Error) *[7, Adapted from Definition 3.2] Given an optimization problem A, any instance x of A, and any feasible solution y for x, let OPT_A be the cost of any optimal solution for x and let c_A be the cost of y. The* **absolute error** *of y with respect to x is defined as:*

$$D(x, y) \quad = \quad |OPT_A - c_A|$$

Definition 2.1.6 (Performance Ratio) *[7, Adapted from Definition 3.6] Given an optimization problem A, any instance x of A, and any feasible solution y for x, let OPT_A be the cost of any optimal solution for x and let c_A be the cost of y. The* **performance ratio** *of y with respect to x is defined as:*

$$R(x,y) = max\left(\frac{c_A}{OPT_A}, \frac{OPT_A}{c_A}\right)$$

Approximation algorithms that are of particular interest are those which have these quality measures bounded by constants for all input instances.

Definition 2.1.7 (absolute approximation algorithm) *[7, Definition 3.3] Given an optimization problem A and an approximation algorithm γ for A, γ is an* **absolute approximation algorithm** *for A if, given any instance x of A, the absolute error of the approximate solution y of x is bounded by a constant k, that is:*

$$D(x,y) \leq k$$

Definition 2.1.8 (r-approximate algorithm) *[7, Definition 3.7] Given an optimization problem A and an approximation algorithm γ for A, γ is an* **r-approximate algorithm** *for A if, given any instance x of A, the performance ratio of the approximate solution y of x is bounded by r, that is:*

$$R(x,y) \leq r$$

For example, a 2-approximate algorithm for a maximization (minimization) problem will always provide a solution whose cost is at least one half of (double) the optimal cost.

When analyzing computational complexity, it is convenient to have some notion of intractability. Recall that an intractable decision problem is simply one that does not have an efficient algorithm. Given that quality also plays a role in approximation complexity analysis, this single criterion is no longer seems adequate. To further complicate matters, there are varying degrees of quality (*e.g.*, absolute error, performance ratio). In this thesis, we consider a range of tractable problems, from those with absolute approximation algorithms to those which have one of the two approximation schemes defined below. If an optimization problem is not tractable, then it is considered to be intractable. The scheme defined below encompasses r-approximate algorithms that have the ability to sacrifice computation time for better performance guarantees. Note that problems which allow this type of approximation are extremely useful in practice.

Definition 2.1.9 (Polynomial-Time Approximation Scheme) *[7, Definition 3.10] Given an optimization problem A, whose associated decision problem is intractable, an algorithm γ is said to have a* **polynomial-time approximation scheme (PTAS)** *for A if for any instance x of A and any $r \in \mathbb{Q}$, $r > 1$, γ with input (x, r) returns an r-approximate solution of x in time polynomial in the instance length $|x|$.*

The next scheme is even better as its running time is not only polynomial in the instance size, but also in the inverse of the performance ratio.

Definition 2.1.10 (Fully Polynomial-Time Approximation Scheme) *[7, Definition 3.12] Given an optimization problem A, whose associated decision problem is intractable, an algorithm γ is said to have a* **fully polynomial-time approximation scheme (FPTAS)** *for A if for any instance x of A and any rational value*

$r > 1$, γ with input (x, r) returns an r-approximate solution of x in time polynomial both in the instance length $|x|$ and in $1/(r-1)$.

2.1.3 Classes and Reductions

In the last section it was implicitly stated that the existence of an algorithm with certain properties proves that a particular problem is tractable. A proof of intractability, on the other hand, is more involved and first requires a discussion of complexity classes and reductions.

2.1.3.1 Classes

Computational complexity theory involves classifying problems in terms of their algorithms. Complexity classes of interest in this thesis are sets of problems having a particular time complexity, such as the following three standard complexity classes for decision problems:

- **P**: All decision problems solvable in polynomial time by an algorithm running on a DTM.
- **NP**: All decision problems **verifiable** in polynomial time, *i.e.*, for a given instance x and candidate solution for x, it is possible to check if this candidate solution is in fact a solution for x in polynomial time by an algorithm running on a DTM. The verification *and* generation of all candidate solutions can be implemented using a **nondeterministic Turing machine** (NDTM) (see [16, Section 2.3] for an explanation of the operation of a NDTM).

- **EXPTIME**: All decision problems solvable in $O(2^{p(n)})$ time, for some polynomial p of instance length n, by an algorithm running on a DTM.

The complexity classes of interest for optimal-cost evaluation problems (due to Krentel [24]) are:

- **OptP**: All optimal-cost evaluation problems whose associated decision problems are in **NP**.

- **OptP**[$O(\log n)$]: All problems in OptP that have solutions which are polynomially-bounded by their instance lengths.

Also of interest are the following standard complexity classes for optimization problems:

- **PO**: All optimization problems whose associated decision problems are in **P**.

- **NPO**: All optimization problems whose associated decision problems are in **NP**.

- **FPTAS**: All **NPO** optimization problems that admit fully polynomial-time approximation schemes.

- **PTAS**: All **NPO** optimization problems that admit polynomial-time approximation schemes.

- **APX**: All **NPO** optimization problems solvable in polynomial time by an r-approximate algorithm, where $r \in \mathbb{Q}$, $r > 1$.

- F-**APX**: All **NPO** optimization problems solvable in polynomial time by an r-approximate algorithm, where for some polynomial p of instance length n,

F-APX is either **log-APX** ($r = O(\log n)$), **poly-APX** ($r = O(p(n))$), or **exp-APX** ($r = O(2^{p(n)})$).

The following are known inclusion relationships between the classes above: (1) **P** $\subseteq$ **NP** $\subseteq$ **EXPTIME** [16, page 32], (2) **OptP**$[O(\log n)]$ $\subseteq$ **OptP** [24, page 490], and (3) **PO** $\subseteq$ **FPTAS** $\subseteq$ **PTAS** $\subseteq$ **APX** $\subseteq$ **log-APX** $\subseteq$ **poly-APX** $\subseteq$ **exp-APX** $\subseteq$ **NPO** [7, page 112 (**FPTAS** $\subseteq$ **PTAS**) and Equation 8.1(remainder)]. The only proper inclusion above is **P** $\subset$ **EXPTIME**; however, it is strongly conjectured that **P**$\neq$**NP**, *i.e.*, **P** $\subset$ **NP**. This widely accepted conjecture can be used to show that all the inclusions from (2) and (3) above are proper as well (see [17, page 488] for (2) and [7, Exercise 8.1] for (3)).

In Section 2.1.4.1, as part of analyzing the complexity of a particular problem, it is explained how to isolate the hardest problems in a class. The above class inclusions and the notion of a reductions, discussed next, are crucial in that process.

2.1.3.2 Reductions

This section is divided into three main parts: (1) informal introduction to reducibility, (2) formal definitions of two particular types of reducibility, and (3) several specific properties exploited in later chapters.

A **reduction** from a problem A to a problem B, denoted as $A \propto B$, is an algorithm that can use an algorithm for B to solve A. A reduction can be viewed as an algorithm for A calling an algorithm for B as a subroutine; A **reduces** to B if for any given instance x of A, solving x only requires solving one or more *constructed* instances of B. For example, consider again the CLIQUE problems from Section 2.1.1. Reductions are used to describe several of the relationships between these problems in Figure 2.1.

```
k-CLIQUE(<G,k>)
[V,E] = SOL-k-CLIQUE(<G,k>)
If |V| >= k Then
    return(True)
Else
    return(False)
End If
```

(a)

```
SOL-k-CLIQUE(<G,k>)
[V,E] = MAX-k-CLIQUE(<G>)
If |V| >= k Then
    return([V,E])
Else
    return(False)
End If
```

(b)

Figure 2.1: Reductions Between Different Problem Types. These reductions are essentially the following subroutine calls: (a) decision problem k-CLIQUE calling solution problem SOL-k-CLIQUE and (b) SOL-k-CLIQUE calling optimization problem MAX-k-CLIQUE. Refer to Section 2.1.1 for problem descriptions.

Reductions can be classified into sets based on their interrelational properties. A **reducibility** is a set of reductions that preserve the following two properties:

1. **Transitivity**: For all problems A, B, and C, if $A \propto B$ and $B \propto C$, then $A \propto C$.
2. **Tractability Preservation**: For all problems A, B, if $A \propto B$ and B is tractable, then A is tractable.

Another property, **Intractability Preservation**, is implicit from (2) above: For all problems A, B, if $A \propto B$ and A is intractable, then so is B. This **flow** of tractability and intractability along a reduction is shown in Figure 2.2.

This collection of properties is extremely useful for complexity analysis, especially within the framework proposed in Section 2.1.4.2, where reductions play a central role. As a preview, suppose the reductions in Figure 2.1, namely k-CLIQUE $\propto$ SOL-k-CLIQUE and SOL-k-CLIQUE $\propto$ MAX-k-CLIQUE, are members of a polynomial-time reducibility. From Property (1), k-CLIQUE $\propto$ MAX-k-CLIQUE, which means that if a polynomial-time algorithm can be designed for MAX-k-CLIQUE, then one also must exist for

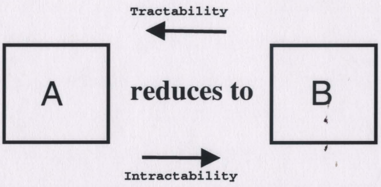


Figure 2.2: Flow of (In)Tractability Along a Reduction. Labeled boxes represent problems A and B.

k-CLIQUE (Property (2)). Moreover, if proof exists that k-CLIQUE is intractable, then time need not be wasted on trying to design a polynomial-time algorithm for MAX-k-CLIQUE, since it is implicit from Property (2) that one cannot exist.

In Chapter 4, when decision problems are analyzed, **polynomial time many-one reducibility** is used; however, in Chapter 5 when optimization problems are analyzed, **metric reducibility** and **L-reducibility** are used instead. These reducibilities are defined as follows:

Definition 2.1.11 (many-one reduction) *[7, Definition 1.10] A decision problem A is said to be many-one reducible to a decision problem B if there exists a polynomial-time algorithm R which given any instance x of A transforms it into an instance $R(x)$ of B in such a way that the solution to $A =$ "yes" if and only if the solution to $B =$ "yes". In such a case, R is said to be a* **many-one reduction** *from A to B, denoted $A \leq_m B$.*

Definition 2.1.12 (metric reduction) *[24, page 493] Let $f, g : \Sigma^* \to \mathbb{N}$ (i.e., two optimal-cost evaluation problems A and B, respectively). A function f is said to be metric reducible to a function g if two polynomial-time functions $T_1 : \Sigma^* \to \Sigma^*$ (i.e., algorithm R from Definition 2.1.11) and $T_2 : \Sigma^* \times \mathbb{N} \to \Sigma^*$ (i.e., transformation of one optimal-cost evaluation solution into another optimal-cost evaluation solution) exist, such that $f(x) = T_2(x, g(T_1(x)))$ for all $x \in \Sigma^*$ (i.e., for all instances of A).*

Definition 2.1.13 (L-reduction) *[7, Adapted from Definition 8.4] Let A and B be two optimization problems in the class NPO. A is L-reducible to B ($A \leq_L B$) if two functions f and g and two positive constants α and β exist such that:*

1. *For any instance x of A, $f(x)$ is computable in polynomial time.*
2. *For any instance x of A, if a feasible solution exists for x then a feasible solution exists for instance $f(x)$ of B.*
3. *For any instance x of A and any feasible solution y for instance $f(x)$ of B, $g(x, y)$ is a feasible solution for instance x of A and is computable in polynomial time.*
4. *For any instance x of A, any feasible solution y for instance $f(x)$ of B, and any feasible solution $g(x, y)$ for x, if OPT_A and OPT_B are the costs of any optimal solution for x and $f(x)$, respectively, and c_A and c_B are the costs of solutions $g(x, y)$ and y, respectively, then:*

$$OPT_B \leq \alpha(OPT_A) \tag{2.1}$$

$$|OPT_A - c_A| \leq \beta(|OPT_B - c_B|) \tag{2.2}$$

Lemma 2.1.14 *Polynomial-time many-one reductions satisfy transitivity and tractability preservation [16, Lemmas 2.1 and 2.2].*

Lemma 2.1.15 *Metric reductions satisfy transitivity preservation [24, page 493].*

Lemma 2.1.16 *L-reductions satisfy transitivity and tractability preservation [33, Propositions 1 and 2].*

There are many reduction techniques (see [16, Section 3.2]), but one stands out in this thesis due to the its potential for reusability. A *reduction by restriction*, or just **restriction reduction**, R from a decision problem A to a decision problem B transforms an instance x of A into instance $R(x)$ of B using a simple restriction so that x and $R(x)$ are identical. This is possible when B contains A as a special case. A **quasi-restriction reduction** transforms x into $R(x)$ so that x and $R(x)$ are almost identical, except $R(x)$ has additional structures that have to be artificially constructed in such a way that *forces* A to be a special case of B. The next lemma states how applying these reduction techniques to polynomial-time many-one reductions allows easy derivation of corresponding metric reductions and L-reductions.

Lemma 2.1.17 *Any restriction or quasi-restriction polynomial-time many-one reduction is also a metric reduction between their associated optimal-cost evaluation problems and a L-reduction between their associated optimization problems.*

Proof: Consider two optimal-cost evaluation problems A and B and suppose R is either a restriction or a quasi-restriction many-one reduction between their associated decision problems A' and B'. From the definitions of restriction and quasi-restriction reductions above, the given instance x of problem A' and the constructed instance $R(x)$ of problem B' are identical. Therefore, their costs (and hence the costs of their optimal solutions) must also be identical. Observe that $OPT_{A'} = OPT_{B'}$ is by definition a metric reduction such that T_1 is the reduction R above and T_2 is the trivial function which maps an optimal-cost evaluation problem to itself. Following from this same logic, in the L-reduction definition (Definition 2.1.13), OPT_A can be substituted into Equation 2.1 for OPT_B, making $\alpha = 1$. Similarly, OPT_A can be substituted for OPT_B and c_A for c_B in Equation 2.2, making $\beta = 1$. ∎

As the majority of polynomial-time many-one reductions in Chapter 4 are either restriction or quasi-restriction reductions, this lemma will prove to be quite useful in Chapter 5.

2.1.4 Analyzing Problem Complexity

In this section, first the traditional sequence of tasks is described for analyzing a single computational problem (Section 2.1.4.1). This works well for *one* problem; however, it is shown how a reordering of these tasks turns out to be ideal when analyzing a *family* of related problems (Section 2.1.4.2).

2.1.4.1 Analyzing a Single Problem

Given a single problem A, the first task involves designing an efficient algorithm for A, *e.g.*, one that has an asymptotic worst-case time complexity that is polynomial in the instance length. Failing that, so as to not waste any more time, the next tasks to prove that no efficient algorithm for A exists. To establish intractability in general, one must define the following, *e.g.*, [46, page 14]:

1. A universe U of problems.

2. A class $T \subset U$ of tractable problems.

3. A reducibility $\propto$ between pairs of problems in U.

4. One or more classes of problems $C \subset U$ such that $T \subset C$.

Given a class of problems $K \subset U$, a problem B is **K-hard** if for all problems $A \in K$, $A \propto B$. If B is also in K, then B is **K-complete**. K-complete problems are the most computationally difficult problems in K and K-hard problems are at least as difficult at the most difficult problems in K. Hence, given that a problem B is C-hard relative to $\propto$ for any class C, B cannot be in T and therefore does not have an efficient algorithm. These concepts are illustrated in Figure 2.3, where a reducibility from a problem A to a problem B is depicted as an arrow from A to B.

For example, if U is the universe of decision problems, then T is the class **P** of tractable decision problems, the reducibility is $\leq_m$, and C is the class **NP** of problems, which includes the class **P**. Note that we only know **P** $\subseteq$ **NP** and **P** $\subset$ **NP** is only true assuming that **P** $\neq$ **NP**, which has never been proven (but is nonetheless widely accepted) [16, page 33]. Given a problem B, the usual process then, is attempt to prove tractability by designing an efficient algorithm and failing that, prove intractability by finding a C-complete problem A (the appendices of [7, 16, 17] are good places to start), and proving $A \propto B$. As shown next, this order may not be ideal when studying the complexity of a family of related problems.

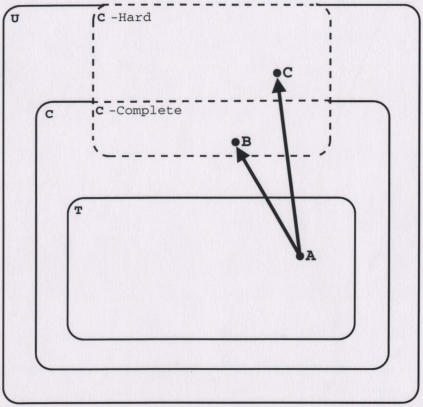


Figure 2.3: The Relationship Between Hardness and Completeness Results. See main text for explanation of symbols (Adapted from [46, Figure 2.2]).

2.1.4.2 Analyzing a Family of Problems

This section proposes a framework for systematically analyzing the complexity of a related set of problems called a **family**. Every family has a core (usually intractable) problem of interest. Constraints which exist in practice are added to this core problem to create new problems which may or may not be tractable. These constraints may be simple restrictions on the core problem's allowable instances, or brand new features. For example, instances of a problem involving a database table could be restricted to those which have a limited number of columns, or they could have an additional feature like a partition of the columns based on some criteria. Ultimately, the aim is to find versions of a core problem of interest which are polynomial-time solvable and thus useful in practice.

Studying a set of related problems is not a new concept. Garey and Johnson [16, Section 4.1] point out that analyzing the complexity of a problem should not end with a proof of NP-completeness; subproblems of an NP-complete problem, which appear as special cases in practice, may be solvable in polynomial time. To take advantage of this phenomenon, they propose placing restrictions on the allowable instances of a known NP-complete problem until the *frontier of tractability* is located. Although locating tractable subproblems is worthwhile, Garey and Johnson's framework does not have a precise method for generating these subproblems. Wareham [46, Section 2.1.3], on the other hand, generates all related problems systematically. His *systematic parameterized complexity analysis* framework is more desirable than the one proposed by Garey and Johnson because, in examining *all possible* restrictions of a core intractable problem relative to a specified set of aspects, it can provide a partial characterization of the set of non-polynomial time algorithms for that problem; Garey and Johnson generate individual problems haphazardly and cannot point to which aspect-subsets make a problem (in)tractable. Our new family-oriented systematic analysis framework is inspired by Wareham's analysis framework; however, it is superior for the following reasons:

- Our framework, by explicitly building all inter-family reductions first, generates *and analyzes* problems systematically, permitting complexity results for selected problems to propagate through the family so that new results may be obtained with minimal additional effort; Wareham's analysis investigates all generated problems individually.

- Our framework is applicable to *multiple types of complexity* analysis and results may be transfered between types; Wareham's analysis is only applicable within the realm of *parameterized* complexity [12].

Our framework for analyzing the complexity of a family of problems is made up of the following set of tasks:

1. Characterize reduction types.

2. Prove the correctness of the template reductions.

3. Systematically acquire all reductions.

4. Create a web of selected reductions.

5. Find (in)tractability results, focusing efforts on roots of (in)tractability.

The first task characterizes reduction types, exploiting the fact that reductions in a problem family may be similar. Recognizing and formally stating the differences and commonalities between reductions will drastically cut down on the effort of acquiring all reductions. For example, suppose two problems A and B would be identical if aspect r of B was set to zero. Then the reduction $A \propto B$ and its associated proof of correctness (discussed below), could serve as a template reduction for any other pair of problems that shares this characteristic.

The second task in analyzing a family of problems is proving the correctness for each basic type of reduction so that in the third task, a complete summary of reductions can be obtained systematically to cover all possibilities. This involves stating how the proofs in the second task may be modified to become proofs of correctness for reductions between other pairs of problems.

In the fourth task, selected reductions from the third task are chosen to build a **reduction web**. In general, a reduction web looks like Figure 2.4. The circles represent all members of a family of problems and an arrow from problem A to problem B in the web signifies reduction $A \propto B$. In Figure 2.4, certain problems are singled out: the problems on the right-hand side (RHS) are called **tractability roots** because if tractable algorithms are found there, given the flow of tractability, other tractability results will propagate back through the reduction web; similarly, problems on the left-hand side (LHS) are named **intractability roots**. Note that these roots only locate *ideal* places in the web, *i.e.*, places from which results could propagate the furthest (other problems may be the actual roots of (in)tractability (see below)). Furthermore, it is not necessary to include all reductions from the third task in the reduction web; in order to complete this fourth task, (a) all problems

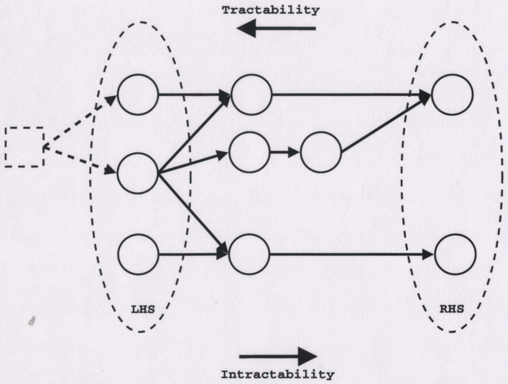


Figure 2.4: A Reduction Web. A circle represents one problem from a family of problems and an arrow represents the reduction from one problem to another (the bigger arrows show the flow of (in)tractability). The problems circled on the right-hand side (RHS) and left-hand side (LHS) are potential tractability and intractability roots, respectively. The dotted arrows are supplementary reductions.

must appear in the web and (b) enough reductions must appear in the web so that the numbers of (in)tractability roots are minimized.

The fifth task involves deriving results for these (in)tractability roots. For a tractability root, the idea is to design an efficient algorithm for it; however, if this appears to be too difficult, then one should iteratively move one reduction to the left in the web until an efficient algorithm is found. For an intractability root, the idea is to reduce a known C-complete problem to it. This reduction from a C-complete problem outside the reduction web is called a **supplementary reduction** (dotted arrow in Figure 2.4). Note that both of these result-finding processes may overlap. Therefore, depending on results obtained, the web may have to be rebuilt so that the number of (in)tractability roots is kept to a minimum. This is another reason why systematically acquiring all reductions in the third task is important.

This new framework may seem backwards when compared to the traditional way of analyzing the complexity of a problem B (Section 2.1.4.1), *i.e.*, designing an algorithm for B, searching for a C-complete problem A, and then proving $A \propto B$; however, there are benefits to systematically acquiring all reductions between problems in a family first, namely:

- for a particular complexity theory, (in)tractability results for selected problems will propagate through the reduction web so that new results may be obtained with minimal additional effort (*e.g.*, Lemmas 2.1.14 and 2.1.16) and
- reduction proof mechanisms from one complexity theory can sometimes be reused in another theory (*e.g.*, Lemma 2.1.17).

For example, Chapter 4 creates a many-one reduction web and uses it to derive results for decision problems in the family described in the next chapter. Although the complexity analysis changes from traditional in Chapter 4 to approximation in Chapter 5, given Lemma 2.1.17, this same many-one reduction web is reused to lay the foundations for a metric reduction web and an L-reduction web. These webs are then used to derive the first known approximation intractability results for the optimization problems in the family. Hence, if it is designed carefully, a reduction web is analogous to a road map (see Figure 2.5); goods (complexity results) can be transported from place to place (problem to problem) along the same roads (reductions) using a various modes of transportation (multiple complexity theories).

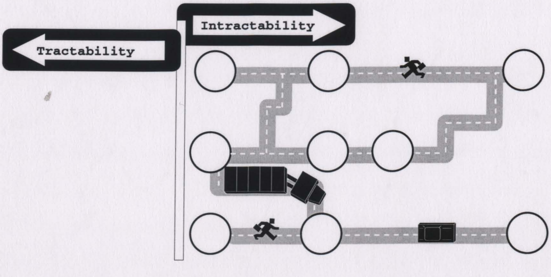


Figure 2.5: A Reduction Web Viewed as a Road Map. This is an analogue of Figure 2.4; (in)tractability results may transported along the reduction road system from problem to problem. Several modes of transportation symbolize different complexity theories sharing one reduction web.

The basis for the utility-preserving k-Anonymity family of problems analyzed in this thesis are two related benchmark problems from the literature. The next section describes these problems in detail, and summarizes work on these and related problems to date.

2.2 k-Anonymity

This section serves two purposes: (1) it describes two basic benchmark problems, from which the rest of the problems in this thesis are derived (Section 2.2.1), and (2) it organizes the previous (Section 2.2.2) and related (Section 2.2.3) work to facilitate future work.

2.2.1 Fundamentals

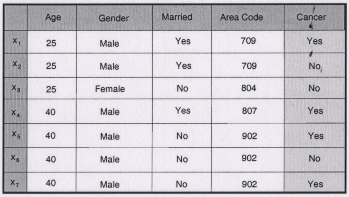
General k-anonymity concepts in this section are adapted[1] from Sweeney [39], using newly defined terminology, as well as existing notation from Meyerson and Williams [31].

Let a **person-specific table** be an $n \times m$ database table T. The n rows represent a set $X = \{x_1, x_2, \ldots, x_n\}$ of people and the m columns correspond to a set $A = \{a_1, a_2, \ldots, a_m\}$ of human attributes[2]. Let entry e_{ij} be the the j^{th} attribute of person x_i in T. Each entry comes from a finite alphabet of all possible attribute values Σ and since a row x_i is made up of m values from Σ, T can formally be defined as a subset $T \subseteq \Sigma^m$ (in general, Σ could be infinite, *e.g.*, real numbers, and could differ for each attribute). In Figure 2.6, T is a 7×5 table with $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$, $A =$

[1]Notation had to be adapted and simplified from [39] because the original notation is unclear and potentially erroneous.

[2]The terms *attribute* and *column* are used interchangeably throughout this thesis.

{Age, Gender, Area Code, Cancer}, and to exemplify what an entry is, e_{34} =804.



	Age	Gender	Married	Area Code	Cancer
x_1	25	Male	Yes	709	Yes
x_2	25	Male	Yes	709	No
x_3	25	Female	No	804	No
x_4	40	Male	Yes	807	Yes
x_5	40	Male	No	902	Yes
x_6	40	Male	No	902	No
x_7	40	Male	No	902	Yes

Figure 2.6: Example of a Person-Specific Database Table. Row x_i is an individual's health record and each column is a specific attribute. The unshaded attributes are considered *personal information*, while the shaded attribute is *private information.*

A user-defined **quasi-identifier**, $Q = \{q_1, q_2, \ldots, q_h\} \subseteq A$, $h \leq m$ specifies the **personal information**. To achieve k-anonymity, either the entries or attributes of the quasi-identifier are **generalized** (*i.e.*, made less specific) so that groups of at least k people look identical in terms of their personal information. The other $A \backslash Q$ attributes (*i.e.*, **private information** like *Cancer* from Figure 2.6) remain unmodified.

Sweeney [39] explains how this generalization can be accomplished using quasi-identifier **domain generalization hierarchies** (DGH's). In essence, a DGH is a user-defined structure which imposes a total order on the set of possible generalizations for domain values of a particular attribute. The generalized values in this structure are partitioned into levels such that the bottom level consists of domain values and each consecutive level contains increasingly more general values. To illustrate how they are used, consider the Area Code DGH from Figure 2.7. The domain for this attribute is $\{204, 250, 306, 403, 418, 506, 705, 709, 780, 804, 807, 819, 867, 902\}$. At the bottom of the Area Code DGH no generalization is applied; however, as we traverse up the hierarchy levels, we notice that the area code becomes increasingly more general by using the symbol $*$ to replace numbers. At the top of the DGH, we see total suppression. For example, the Area Code 804 can be generalized to $80*$, or can be further generalized to $8**$, or even further to $***$. The generalization at the top of the DGH, where no information is revealed, is called **suppression**.

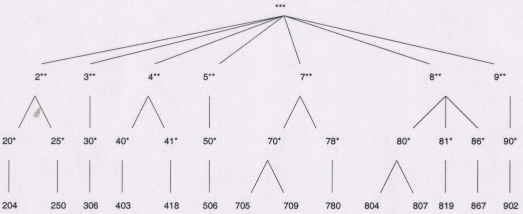


Figure 2.7: Domain Generalization Hierarchy for Area Code. Usage explained in text.

Observe that T can also be represented by a matrix (as illustrated in Figure 2.8); it is sometimes convenient to refer to T as a set of n m-dimensional vectors $\{x_1, x_2, \ldots, x_n\}$, where entry e_{ij} is the j^{th} entry in vector x_i (*i.e.*, $x_i[j]$).

$$\begin{array}{c|cccc|} & a_1 & a_2 & \dots & a_m \\ x_1 & e_{11} & e_{12} & \dots & e_{1m} \\ x_2 & e_{21} & e_{22} & \dots & e_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n & e_{n1} & e_{n2} & \dots & e_{nm} \end{array}$$

Figure 2.8: A Person-Specific Dataset Represented by $n \times m$ Matrix.

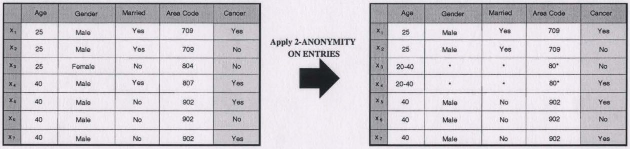
Two formulations of k-Anonymity have been considered in the literature, which only differ in DGH usage. Two more definitions are necessary before those problems are stated formally. For the first definition, let Σ_{DGH_i} denote the alphabet of **permissible generalizations for an attribute** q_i according to the DGH for q_i and for the second definition, let $\Sigma_{DGH_i}(y)$ denote the alphabet of **permissible generalizations for a particular value** y from the domain of attribute q_i. To clarify, refer back to Figure 2.7; if Area Code is the fourth attribute in quasi-identifier Q, then Σ_{DGH_4} is the set of all values pictured in Figure 2.7, *i.e.*, $\{204, 250, \dots, 20*, 25*, \dots, 2**, 3**, \dots, 9**, ***\}$ and $\Sigma_{DGH_4}(804) = \{804, 80*, 8**, ***\}$. Next define g to be the function $g : T \longrightarrow (\Sigma_{DGH_1} \cup \Sigma_{DGH_2} \cup \dots \cup \Sigma_{DGH_h})^h$, such that every quasi-identifier entry e_{ij} in $g(T)$ is $\in \Sigma_{DGH_j}(e_{ij})$.

The first problem definition can now be stated as (adapted from Sweeney [39, Definition 6: k-MINIMAL DISTORTION]):

SOL-k-ANONYMITY ON ENTRIES

Instance: An $n \times m$ table T over an alphabet Σ, a quasi-identifier Q, DGH's for each attribute in Q, and a positive integer k.

Solution: Any k-anonymous table $g(T)$.


Apply 2-ANONYMITY ON ENTRIES

	Age	Gender	Married	Area Code	Cancer
x_1	25	Male	Yes	709	Yes
x_2	25	Male	Yes	709	No
x_3	25	Female	No	804	No
x_4	40	Male	Yes	807	Yes
x_5	40	Male	No	902	Yes
x_6	40	Male	No	902	No
x_7	40	Male	No	902	Yes

	Age	Gender	Married	Area Code	Cancer
x_1	25	Male	Yes	709	Yes
x_2	25	Male	Yes	709	No
x_3	20-40	*	*	80*	No
x_4	20-40	*	*	80*	Yes
x_5	40	Male	No	902	Yes
x_6	40	Male	No	902	No
x_7	40	Male	No	902	Yes

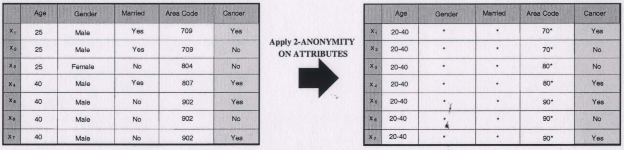
Figure 2.9: Solving SOL-k-ANONYMITY ON ENTRIES.

The transformation of a person-specific table into a 2-anonymous table satisfying SOL-k-ANONYMITY ON ENTRIES is depicted in Figure 2.9. Before stating the second (subtly different) problem, define $\bar{g}$ to be another function that is the same as g, with the added stipulation that every $e_{ij} \in q_j$ must be generalized up to the same level in DGH_j. The solution version of the problem (adapted from Sweeney [39, Definition 4: k-MINIMAL GENERALIZATION]) is as follows:

SOL-k-ANONYMITY ON ATTRIBUTES

Instance: An $n \times m$ table T over an alphabet Σ, a quasi-identifier Q, DGH's for each attribute in Q, and a positive integer k..

Solution: Any k-anonymous table $\bar{g}(T)$.



	Age	Gender	Married	Area Code	Cancer
x_1	25	Male	Yes	709	Yes
x_2	25	Male	Yes	709	No
x_3	25	Female	No	804	No
x_4	40	Male	Yes	807	Yes
x_5	40	Male	No	902	Yes
x_6	40	Male	No	902	No
x_7	40	Male	No	902	Yes

Apply 2-ANONYMITY ON ATTRIBUTES

	Age	Gender	Married	Area Code	Cancer
x_1	20-40	*	*	70*	Yes
x_2	20-40	*	*	70*	No
x_3	20-40	*	*	80*	No
x_4	20-40	*	*	80*	Yes
x_5	20-40	*	*	90*	Yes
x_6	20-40	*	*	90*	No
x_7	20-40	*	*	90*	Yes

Figure 2.10: Solving SOL-k-ANONYMITY ON ATTRIBUTES.

Figure 2.10 shows how a person-specific table is transformed into a 2-anonymous table satisfying SOL-k-ANONYMITY ON ATTRIBUTES.

The historical evolution of these problems seems to have arisen from Sweeney's need to formalize the problem solved by the heuristic algorithm [37] she developed in 1997. A year later Samarati and Sweeney gave the first definition of k-Anonymity [36]. While Samarati is still working in the area of data security, she has only published one k-Anonymity paper [35] since working with Sweeney; however, Sweeney (considered to be the founder of k-Anonymity) wrote her Ph.D. thesis [38] on this topic and has contributed several papers (*e.g.*, [40, 39]). This work and other k-Anonymity research is discussed next.

2.2.2 Previous Work

As the research in this area is relatively new and there has been a recent explosion of new contributions, it can be difficult to distinguish between the original problems stated by Sweeney [39] and recently proposed *easier-to-solve* versions. This section gives a unified presentation of all versions of k-Anonymity research, using a common set of notations and definitions, so that it may be used as concrete reference for future work. Note that a clear distinction has been made between *previous* and *related* work.

The only optimal solutions in the literature for SOL-k-ANONYMITY ON ENTRIES are Sweeney's Preferred Minimal Generalization (MinGen) [39] and k-Similar [38] algorithms, while the only optimal solution proposed for SOL-k-ANONYMITY ON ATTRIBUTES is Samarati's k-Minimal Generalization algorithm [35]. None of these algorithms are efficient, since in the worst case, they require checking all possible generalization schemes of an $n \times m$ database table T. Suppose h is the maximum number of levels in any of the m DGH's and all DGH's have this number of levels. As each of the m entries in a row can be one of h values in a generalization scheme for SOL-k-ANONYMITY ON ENTRIES, and there are n rows in a table, each of which can adopt a different scheme, there are $O(\prod_{i=1}^{n} h^m) = O(h^{nm})$ possible schemes to check. Since in SOL-k-ANONYMITY ON ATTRIBUTES, the generalization of any value in a column predetermines the generalization level for all other values in that column, there are $O(h^m)$ possible generalization schemes to check. Notice for both problems, if DGH's were constrained so that at each level (except the last), at least two values generalize to the same value in the next level, then h is bounded by Σ.

The computational complexity has been examined for simplified versions of SOL-k-ANONYMITY ON ENTRIES and SOL-k-ANONYMITY ON ATTRIBUTES by Meyerson and Williams [31] and SOL-k-ANONYMITY ON ENTRIES by G. Aggarwal *et al.* [3]. These simplifications are as follows:

- The quasi-identifier concept is dropped.

- Each domain generalization hierarchy has only two levels.

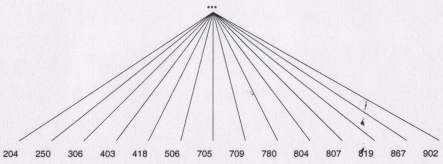


Figure 2.11: Suppression Domain Generalization Hierarchy for Area Code.

Consider the Area Code DGH in Figure 2.11; it is similar to the Area Code DGH from Figure 2.7, except there is only two levels (no generalization and suppression). The majority of the remaining content of this thesis will only discuss this special case of generalization, since any generalization problems will be at least as hard to solve as their purely suppression counterparts.

A **suppressed entry** in T is represented by the symbol $*$, where $* \notin \Sigma$. Define function f to be a **suppression function** on T, $f : T \to (\Sigma \cup \{*\})^m$,such that $\forall(x_i \in T)\forall(j \in \{1, 2, \ldots, m\})\{f(x_i[j]) \in \{x_i[j], *\}\}$. Furthermore, $f(T) = \{f(x_1), f(x_2), \ldots, f(x_n)\}$ is a **k-anonymous table** if $\forall(x_i \in T)\{\exists(i_1, i_2, \ldots, i_{k-1} \in \{1, 2, \ldots, n\}) \mid f(x_{i_1}) = f(x_{i_2}) = \ldots = f(x_{i_{k-1}}) = f(x_i)\}$. Informally, a k-anonymous table is the same as the original table, except some entries are suppressed so that each row becomes identical to at least $k-1$ other rows. It is important to note that f does not permute, add, or delete any of the rows or columns of T. Also, the number of suppressed entries in any k-anonymous table cannot exceed $n \times m$.

The suppression versions of SOL-k-ANONYMITY ON ENTRIES and SOL-k-ANONYMITY ON ATTRIBUTES are as follows (see Figures 2.12 and 2.13, respectively):

SOL-e-SUPPRESSION (SOL-e-SUP)

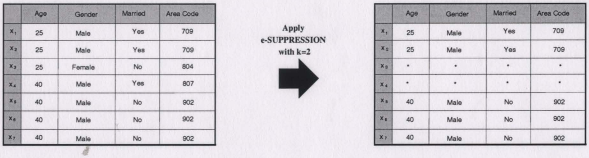
Instance: An $n \times m$ table T over an alphabet Σ and positive integers e and k.

Solution: Any k-anonymous table $f(T)$ that has at most e suppressed entries.

SOL-c-DELETION (SOL-c-DEL)

Instance: An $n \times m$ table T over an alphabet Σ and positives integers c and k.

Solution: Any k-anonymous table $f(T)$ that has at most c deleted columns.



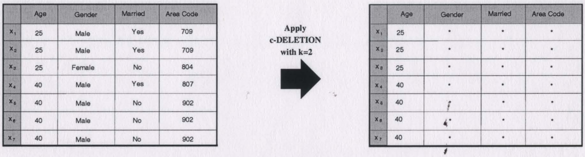
	Age	Gender	Married	Area Code
x_1	25	Male	Yes	709
x_2	25	Male	Yes	709
x_3	25	Female	No	804
x_4	40	Male	Yes	807
x_5	40	Male	No	902
x_6	40	Male	No	902
x_7	40	Male	No	902

Apply
e-SUPPRESSION
with k=2

	Age	Gender	Married	Area Code
x_1	25	Male	Yes	709
x_2	25	Male	Yes	709
x_3	*	*	*	*
x_4	*	*	*	*
x_5	40	Male	No	902
x_6	40	Male	No	902
x_7	40	Male	No	902

Figure 2.12: Solving SOL-e-SUPPRESSION. $e \geq 8$.

Using these simplified versions, Meyerson and Williams [31] prove that both e-SUP ($k \geq 3$ and $|\Sigma| \geq n$) and c-DEL ($k \geq 3$ and $|\Sigma| \geq 2$) are NP-hard by reductions from EXACT COVER BY 3-SETS [16, SP2], which they call 3-DIMENSIONAL PERFECT MATCHING. Given that polynomial-time algorithms thus seem unlikely, they give a polynomial-time $O(m \log k)$-approximate algorithm for MIN-e-SUP. Improving upon these results, G. Aggarwal *et al.* [3] prove that e-SUP ($k \geq 3$ and $|\Sigma| \geq 3$) is NP-hard



	Age	Gender	Married	Area Code
x_1	25	Male	Yes	709
x_2	25	Male	Yes	709
x_3	25	Female	No	804
x_4	40	Male	Yes	807
x_5	40	Male	No	902
x_6	40	Male	No	902
x_7	40	Male	No	902

	Age	Gender	Married	Area Code
x_1	25	•	•	•
x_2	25	•	•	•
x_3	25	•	•	•
x_4	40	•	•	•
x_5	40	•	•	•
x_6	40	•	•	•
x_7	40	•	•	•

Figure 2.13: Solving SOL-c-DELETION. $c \geq 3$.

by a reduction from EDGE PARTITION INTO TRIANGLES [21, Problem 1] and give a polynomial-time $O(k)$-approximate algorithm [3] for MIN-e-SUP. They also prove that using the graph representation required by this $O(k)$-approximate algorithm, one cannot achieve a better than $O(k)$-approximation factor. This proof involves giving two instances of MIN-e-SUP that have identical graphs and have costs which differ by a factor of $O(k)$.

The only other research directly related to the original formulations are the greedy heuristics Datafly [38] and GreedyRelease [5], which offer no performance guarantees (not even that these algorithms run in polynomial time). All other research on k-Anonymity is considered to be *related work* in the context of this thesis and is discussed separately in the next section.

2.2.3 Related Work

Perhaps due to the inherent difficulty of the problem as it was first phrased, researchers are attacking k-Anonymity from many angles. This research either (1) uses different cost metrics to exploit techniques from Data Mining like Classification, Clustering, and Association Rule Mining (for a good introductions see [19]), or (2) uses a different suppression function. This section describes these two groups of research, and gives brief summaries of work related to possible inference attacks on k-Anonymity and how algorithms for k-Anonymity are being used in other contexts.

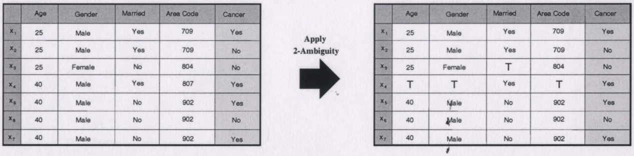
The original problem has been rephrased several times already to take advantage of known algorithms for decision trees, which are used for classification. Decision trees [19, Figure 7.4] segment the original dataset so that records within each segmentation are similar to each other based on an attribute selection measure called *Information Gain* [19, Equation 7.4]. Iyengar [20] incorporates this measure into k-ANONYMITY ON ATTRIBUTES, automatically produces the generalization hierarchies, and assumes a total order on attributes and domains. His solution is a genetic algorithm and he gives an experimental analysis only. Since Iyengar published his paper in 2002, others have adopted his framework [8, 15, 44, 47] – Bayardo and Agrawal [8] use a set representation and implement a systematic search strategy using set enumeration; Winkler [47] proposes another genetic algorithm using simulated annealing; Wang *et al.* offers a greedy heuristic bottom-up approach [44]; and Fung *et al.* [15] gives a greedy heuristic top-down approach that improves upon the bottom-up approach, which only handled categorical attributes.

Other researchers have noticed similarities between k-Anonymity and clustering. Clustering is the process of grouping data into clusters in such a way that objects within a cluster are very similar to one another and are not similar to objects in other clusters (*e.g.*, [19, Figure 8.2]). The similarity is calculated based on a distance metric (*e.g.*, [19, Equation 8.18]). G. Aggarwal *et al.* [2] give a constant-factor approximation algorithm for their clustering version of k-Anonymity.

Other data mining concepts like On-Line Analytic Processing (OLAP) have been used in recent research. Lefevre *et al.* [25] use OLAP operations in the multidimensional data model like *rollup* (see [19, Figure 2.10]). They also incorporate ideas like the *Apriori property* from another data mining technique called Association Rule Mining [19, Chapter 6]. Like Iyengar, LeFevre *et al.* modify k-ANONYMITY ON ATTRIBUTES, provide an algorithm, and perform an experimental analysis. Most recently, LeFevre *et al.* [26] are working on what they call OPTIMAL STRICT k-ANONYMOUS MULTIDIMENSIONAL PARTITIONING; they prove it is NP-hard, provide a greedy $O(n \log n)$-approximate algorithm, as well as experimental results.

Unlike the other research in this section so far, Vinterbo's [41] work is not related to data mining. He defines a slightly different suppression function and calls the problem k-AMBIGUITY, which he proves is NP-hard. Recall that in all other work on k-anonymity the suppressed entry symbol $*$ acts as an extra unknown value and a particular entry is the same as another entry if and only if both entries have the same value (including $*$). Vinterbo's suppressed entry symbol, $\top$, is *indiscernible* from all possible values. That means a particular entry is the same as another entry if both entries have the same value (including $\top$) *or* one of the entries is $\top$ (see Figure 2.14).

Another stream of related work investigates potential inference attacks on k-Anonymity. Even in the beginning, Sweeney [40, Section 4] recognized at least three of these attacks, namely an unsorted matching attack, a complementary release attack, and a temporal attack. New research by Machanavajjhala *et al.* [27] point out other possible inference attacks and proposes a new privacy model, l-diversity, which avoids such an attack by ensuring all the individuals in a group do not have the same *private information*; however, Xiao and Tau [48] demonstrate why l-diversity still


Apply
2-Ambiguity

	Age	Gender	Married	Area Code	Cancer
x_1	25	Male	Yes	709	Yes
x_2	25	Male	Yes	709	No
x_3	25	Female	No	804	No
x_4	40	Male	Yes	807	Yes
x_5	40	Male	No	902	Yes
x_6	40	Male	No	902	No
x_7	40	Male	No	902	Yes

	Age	Gender	Married	Area Code	Cancer
x_1	25	Male	Yes	709	Yes
x_2	25	Male	Yes	709	No
x_3	25	Female	T	804	No
x_4	T	T	Yes	T	Yes
x_5	40	Male	No	902	Yes
x_6	40	Male	No	902	No
x_7	40	Male	No	902	Yes

Figure 2.14: Solving SOL-k-AMBIGUITY.

does not work and introduces a new concept to fix it. Related to this same sort of inference attack, C. Aggarwal [1] shows why k-Anonymity does not work on tables which have a high number of attributes.

Some researchers are not attempting to alter k-Anonymity, rather they are using k-Anonymity in other fields of research. For example, Yao *et al.* [49] implement a procedure that checks for k-anonymity violation when there are multiple views of the information. The authors prove that when functional dependencies are considered, their problem is NP-hard, but is checkable in polynomial-time if functional dependencies are ignored. Zhong *et al.* [50] study k-anonymity in a distributed setting and Atzori *et al.* [6] use k-Anonymity to block inference opportunities in association rule mining. Most recently, Malin [28] shows that when an individual?s personal information is spread over multiple locations, it leaves an *identity trail.* Given this trail, he offers a method for limiting the risk of re-identification using k-Anonymity.

Finally, another vein of related work is that of utility-preservation. Given that this is a major theme of this thesis, previous work on utility-preservation is scrutinized in the next chapter.

Chapter 3

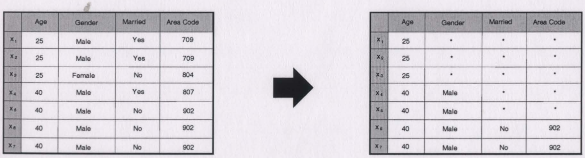
Utility-Preserving k-Anonymity

Several k-Anonymity-based problems have been proposed in the literature; however, these problems do not adequately address preserving utility for the researcher and their algorithms are not computationally efficient. This chapter highlights these inadequacies through a comprehensive overview of previous research, where it is shown that previous solutions lack sufficient ability to meet specific researcher needs. To this end, new utility-preserving problems are proposed.

3.1 Motivation

Consider again the 2-anonymous data set satisfying SOL-e-SUPPRESSION in Figure 2.12. Two entire rows are deleted, leaving researchers with data that could be biased, as the entire population is not included. Furthermore, if a particular attribute is crucial for the researcher's study, such as `Age`, using SOL-e-SUPPRESSION would not allow a cause-and-effect analysis to be 100% accurate. Even more inadequate than SOL-e-SUPPRESSION is the solution to SOL-c-DELETION given in Figure 2.13, where all data except the `Age` attribute is suppressed.

Notice that in both of these problems, the focus is on protecting privacy; researchers do not have the ability to specify what portions of the data set are most crucial to their research, and hence should not be modified during the anonymization process. In these two example solutions, entire rows of information or nearly all information is deleted, resulting in a data set that may be biased and may not even be relevant to the researcher's study. If k-anonymity problems could be reformulated such that the privacy of each individual is still protected *and* researchers receive the information they actually need, then the obtained solutions would truly be optimal. For example, if the researcher was mainly interested in `Age`, but also wanted to receive as much information on `Gender` as possible, a utility-preserving solution such as the one in Figure 3.1 might be preferable.



	Age	Gender	Married	Area Code
x_1	25	Male	Yes	709
x_2	25	Male	Yes	709
x_3	25	Female	No	804
x_4	40	Male	Yes	807
x_5	40	Male	No	902
x_6	40	Male	No	902
x_7	40	Male	No	902

	Age	Gender	Married	Area Code
x_1	25	•	•	•
x_2	25	•	•	•
x_3	25	•	•	•
x_4	40	Male	•	•
x_5	40	Male	•	•
x_6	40	Male	No	902
x_7	40	Male	No	902

Figure 3.1: Solving a Utility-Preserving k-Anonymity Problem. The data set shown on the right is a possible solution to a particular utility-preserving k-anonymity problem called SOL-c'-PARTITION, where a researcher specifies which columns are important as input and individual entries are deleted accordingly. In this case, `Age` is the most important, followed by `Gender`, `Married`, and `Area Code`, in that sequence. At most c' (in this case $c' \geq 5$) individual entries are deleted in the least important column (`Area Code`) and each other column in the importance sequence can have no more deleted entries than the one after it.

These examples illustrate that counting the total number of entries suppressed [3, 31] or columns deleted [31] to achieve k-Anonymity are substandard formulations of the problem; if the problem was formulated in such a way that the privacy of each individual is protected *and* researchers receive the information they need, then the problem's solution would truly be optimal.

3.2 Previous Work: Summary and Critique

The first researcher to attempt utility-preservation within the k-anonymity framework was Sweeney [39, 38]. Sweeney acknowledges in [38, Section 5.4] that the most useful solution is application specific and relies on user-specific preferences. On the surface, it appears as though her work benefits the end-user, but they do not; when she refers to *user input*, she means the *data holder*'s input (not the researcher's). Sweeney actually discourages utility for researchers. For example, to use her Datafly II System [38, Section 6.1], the data holder assigns a number from 0 to 1 to each quasi-identifier attribute based the likelihood that the researcher will link that attribute to outside data (0 means not likely and 1 means highly probable). The algorithm never asks for the researcher's input. Even if Sweeny's weighted scheme accepted input from the researcher, the idea of requiring the assignment of numbers from 0 to 1 is unreasonable. Typically human preference can be partitioned into two groups: *important* and *not important*, or a fixed range of no more than maybe five choices, like on a survey questionnaire.

More recently, Kifer and Gehrke [23] acknowledge that utility has not been well studied for k-Anonymity (*e.g.*, [39]) or l-diversity (*e.g.*, [27]). In addition to the anonymized data set, they propose to release frequency information. This extra information is in the form of a *marginal*, such as those pictured in Figure 3.2 (c)–(f). Obviously, all four marginals in Figure 3.2 cannot be released, since the unique combinations of data values in the marginals may be linked together to reconstruct the original database table. For example, as patient x_4 in Figure 3.2 (a) is the only unmarried forty year old male in area code 807, that information is generalized in Figure 3.2 (b); however, given that his data values have *Count* = 1 in marginals Figure 3.2 (d)-(f), there can be no confidentiality guarantees for his private medical condition. Kifer and Gehrke's framework only releases a selection of marginals whose combination does not lead to this sort of inference attack. This framework decides which marginals to release based on an entropy measure, rather than based on marginals a researcher may actually need. Therefore, while this approach has its merits, from a utility-preserving point of view, this framework is not much better than Sweeney's [39].

Other researchers (*e.g.*, Samarati [35] and Iyengar [20]), on the other hand, discuss utility in a context that could accommodate specific research needs and propose new cost functions that do more than just count the number of suppressed entries or attributes deleted. These cost functions may prefer solutions which contain the greatest number of distinct tuples, those that suppress the least number of rows [35], or those which are used for specific purposes like the data mining technique of classification [20]. Although exploring new cost functions is a step in the right direction, in order to eliminate domain-specific solutions, a wider variety of these functions have to be incorporated into a more general model of k-Anonymity.

In contrast to the work described previously, Vinterbo [42] has arguably taken

(a)

	Age	Gender	Married	Area Code	Cancer
x_1	25	Male	Yes	709	Yes
x_2	25	Male	Yes	709	No
x_3	25	Female	No	804	No
x_4	40	Male	Yes	807	Yes
x_5	40	Male	No	902	Yes
x_6	40	Male	No	902	No
x_7	40	Male	No	902	Yes

(b)

	Age	Gender	Married	Area Code	Cancer
x_1	25	Male	Yes	709	Yes
x_2	25	Male	Yes	709	No
x_3	20-40	*	*	80*	No
x_4	20-40	*	*	80*	Yes
x_5	40	Male	No	902	Yes
x_6	40	Male	No	902	No
x_7	40	Male	No	902	Yes

(c)

Age	Count
25	3
40	4

(d)

Age	Area Code	Count
25	709	2
	804	1
40	807	1
	902	3

(e)

Age	Married	Count
25	Yes	2
	No	1
40	Yes	1
	No	3

(f)

Cancer	Area Code	Count
Yes	709	1
	807	1
	902	2
No	709	1
	804	1
	902	1

Figure 3.2: Injecting Utility into k-Anonymity Using Marginals. This figure depicts (a) A digitized set of records, (b) a k-anonymized version of (a) after applying e-Sup with $k = 2$, (c) the `Age` marginal, (d) the `Age / Area Code` marginal, (d) the `Age / Married` marginal, and (f) the `Gender / Area Code` marginal.

utility to the other extreme, making it too general. Recall from Section 2.2.3 that Vinterbo uses a slightly different suppression function than other researchers called k-Ambiguity. In general, given the domains for each column, he organizes all possible rows that could possibly occur in the table into a lattice. For example, Figure 3.3 illustrates the 3-tier lattice generated from a table with only two columns A and B, where the domain of $A = \{0, 1\}$ and the domain of $B = \{a, b\}$. He then uses the tiers of the lattice to compute information loss and utility; the further up the lattice, the more information loss, so the less utility. To incorporate the concept of providing researchers with pertinent data, Vinterbo defines a utility-encoding scheme that is applied to this lattice. In this overly general scheme, it is not obvious how one goes about encoding specific research needs and it is even less obvious how these encodings are superimposed on the lattice. To date, Vinterbo's work has only been cited (but not pursued) by one other k-Anonymity researcher; there is one sentence in Malin's Ph.D. thesis referring to the NP-hardness of k-Ambiguity [28, page 88]. Given that Vinterbo's mathematically rigorous formalization of k-Ambiguity may be difficult to follow, this lack of interest in his work is perhaps not surprising.

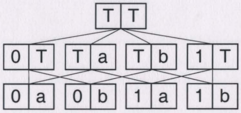


Figure 3.3: A Lattice of Possible Rows for k-Ambiguity. A 3-tier lattice is generated from a table with only two columns A and B, where the domain of $A = \{0, 1\}$ and the domain of $B = \{a, b\}$.

To summarize, a major problem with almost all current implementations of k-anonymity is that they do not allow researchers to specify what portions of the data have the greatest utility in their research and hence cannot be altered during the anonymization process. For example, gender might be crucial to a researcher's study, and any release of data that deletes or generalizes gender values in order to satisfy k-anonymity would not be useful to this researcher. Several approaches to preserving utility under k-anonymity have been proposed [39, 23, 42, 35, 20]; however, all of these proposals are inadequate, because they

1. cannot tailor output to the full range of researcher needs (*e.g.*, [39]), or

2. are too complicated for practical application (*e.g.*, [42]).

These two deficiencies are addressed in the new model of k-Anonymity proposed in the next section.

3.3 A New Family of Utility-Preserving k-Anonymous Problems

Although current definitions for k-Anonymity are inadequate, the underlying idea of guaranteeing anonymity by making groups of patients indistinguishable is still effective. While one ambition of this thesis is finding computationally feasible solutions, first and foremost, k-Anonymity must be made useful for researchers. The new family of Utility-Preserving k-Anonymous Problems presented next is superior to previous utility work because these problem formulations:

1. tailor output to suit specific research needs (by specifying both where and the manner in which table-entry suppression can occur during the anonymization process), and

2. are defined in a way that is clear, concise, and easy to understand.

This is done by building **suppression-location** and **suppression-quantity** variables into their definitions as constraints. Furthermore, each definition also has a **problem-type** variable built in; a problem may be a **deletion**, **suppression**, or **partition** problem. Of particular interest here is the partition problem-type, as it provides a realistic way of allowing researchers to specify the relative importance of particular types of data in the table (and hence the degrees to which they must be preserved during anonymization).

Before the new family of utility-preserving problems can be described in Section 3.3.2, several definitions are required. Section 3.3.1 provides definitions and lemmas which are essential for the remainder of this thesis.

3.3.1 Preliminaries

3.3.1.1 Definitions: Groups and Partitions

The definitions in this section are useful when describing precise regions of a k-anonymous table, such as groups of identical rows and deleted rows or columns.

Definition 3.3.1 (k-partition) *Given an $n \times m$ k-anonymous table $f(T)$, $G = \{g_1, g_2, \ldots, g_l\}$ is a **k-partition** of $f(T) \Leftrightarrow G$ is a partition of the rows of $f(T)$ and $\forall(g \in G)\{k \leq |g| \leq 2k-1 \land \forall(f(x_i), f(x_j) \in g)\{f(x_i) = f(x_j)\}\}$.*

Definition 3.3.2 (k-group) *[31, page 224] Given a k-partition $G = \{g_1, g_2, \ldots, g_l\}$ of an $n \times m$ k-anonymous table $f(T)$, g_i is the i^{th} **k-group** of G.*

Definition 3.3.3 (k-segment) *Given a k-partition $G = \{g_1, g_2, \ldots, g_l\}$ of an $n \times m$ table $f(T)$, let **k-segment** s_{ij} be a $|g_i|$-dimensional column vector, with entries corresponding to the j^{th} entry of each row in k-group g_i.*

Note that all entries in s_{ij} are the same and for convenience, s_{ij} is sometimes referred to as a single value.

Definition 3.3.4 (deleted column) *Given an $n \times m$ table $T = \{x_1, x_2, \ldots, x_n\}$ with attributes $A = \{a_1, a_2, \ldots, a_m\}$, a_j is a **deleted column** if $\forall(x_i \in T)\{x_i[j] = *\}$.*

Definition 3.3.5 (deleted row) *Given an $n \times m$ table $T = \{x_1, x_2, \ldots, x_n\}$, x_i is a* **deleted row** *if $\forall (j \in \{1, 2, \ldots, m\})\{x_i[j] = *\}$.*

3.3.1.2 Definitions: Importance Preservation

The definitions in this section are useful when describing a partition of the database table based on importance. All definitions in this section are only phrased in terms of columns; however, similar definitions exist for their row counterparts.

Definition 3.3.6 (column partition) *Given an $n \times m$ table T and a unique set of column identifiers $S = \{1, 2, \ldots, m\}$ corresponding to the columns in T, $P = \{p_1, p_2, \ldots, p_l\}$, $l \geq 2$, is a* **column partition** *of $T \Leftrightarrow P$ is a partition of S.*

Definition 3.3.7 ($\preceq_{c^*}$) *Given a column partition $P = \{p_1, p_2, \ldots, p_l\}$ of table T, $\preceq_{c^*}$ is a total order on P, where $p_i \preceq_{c^*} p_j$ means that each column of T identified in p_i has no more suppressed entries than any column of T identified in p_j.*

Definition 3.3.8 (column utility-partition) *Given a k-anonymous table $f(T)$, $P = \{p_1, p_2, \ldots, p_l\}$ is a* **column utility-partition** *of $f(T)$ if P is a column partition of $f(T)$ and $\forall (p_i, p_{i+1} \in P)\{p_i \preceq_{c^*} p_{i+1}\}$.*

Informally, $(P, \preceq_{c^*})$ is a totally ordered set based upon some notion of preserving importance; given two sets of column identifiers, p_i and p_j, if columns identified in p_i are more important than columns identified in p_j, then each column identified in p_i has no more suppressed entries than any column identified in p_j. Thus, columns identified in the last set in P are the least important, columns identified in the first set in P are the most important, and for any $p \in P$, all columns identified in p are equally important. For convenience, in this thesis *columns identified in p_i* will be shortened to *columns in p_i*.

3.3.1.3 Useful Lemmas

This section contains four lemmas which help prove the correctness of several reductions in Chapter 4. Each lemma involves modifying a k-anonymous table in such a way that k-Anonymity is preserved.

Lemma 3.3.9 *Given an $n \times m$ k-anonymous table $f(T)$, if $f(T)$ is transformed into $f(T)'$ by adding a column c, such that all the entries in c are identical, then $f(T)'$ is also k-anonymous.*

Proof: Suppose $f(T)$ is a k-anonymous $n \times m$ table $f(T)$, for a particular k. Because $f(T)$ is k-anonymous, it can be k-partitioned, where each k-group g_i can be further partitioned into m k-segments. Recall that all entries in a k-segment are necessarily identical. Now, transform $f(T)$ into an $n \times (m+1)$ table $f(T)'$ by adding column c, *i.e.*, make all entries a symbol $\diamond \neq \star$ in the $(m+1)^{th}$ k-segment of each g_i. Notice that $f(T)'$ is partitioned into the same k-groups as $f(T)$, except there is one extra k-segment in each k-group. The existence of a k-partition of $f(T)'$ implies that $f(T)'$ is k-anonymous. ∎

Lemma 3.3.10 *Given an $n \times m$ k-anonymous table $f(T)$, if $f(T)$ is transformed into $f(T)'$ by adding a k-group, then $f(T)'$ is also k-anonymous.*

Proof: Suppose $f(T)$ is a k-anonymous $n \times m$ table $f(T)$, for a particular k. Because

$f(T)$ is k-anonymous, it can be k-partitioned. Let G be the k-partition of $f(T)$. Now, transform $f(T)$ into $f(T)'$ by adding a new k-group g. Notice that $G' = G \cup \{g\}$ is a valid k-partition of $f(T)'$ and the existence of a k-partition of $f(T)'$ implies that $f(T)'$ is k-anonymous. ∎

Lemma 3.3.11 *Given an $n \times m$ k-anonymous table $f(T)$, if $f(T)$ is transformed into $f(T)'$ by deleting all columns where suppressed entries occur, then $f(T)'$ is also k-anonymous.*

Proof: Let B be the set of columns from $f(T)$ where suppressed entries occur and let b_l be a particular, but arbitrary, column from B. Since $f(T)$ is k-anonymous, it can be k-partitioned, such that each k-group g_i consists of m k-segments. Recall from Definition 3.3.3 that all entries in a k-segment are necessarily identical. Now, transform $f(T)$ into $f(T)'$ by deleting column b_l (*i.e.*, suppress all entries in the l^{th} k-segment of each g_i). Notice that $f(T)'$ is partitioned into the same k-groups as $f(T)$ because there are still m k-segments in each k-group. This implies that there exists a k-partition of $f(T)'$, hence $f(T)'$ is k-anonymous. Continue in the same manner until all columns in B are deleted. ∎

Lemma 3.3.12 *Given an $n \times m$ k-anonymous table $f(T)$, if $f(T)$ is transformed into $f(T)'$ by deleting all rows where suppressed entries occur, then $f(T)'$ is also k-anonymous.*

Proof: Suppose $f(T)$ is a k-anonymous $n \times m$ table $f(T)$, for a particular k. Let $G = \{g_1, g_2, \ldots, g_l\}$ be a k-partition of $f(T)$. It is necessary to delete all k-groups from

G that have suppressed entries. Because $f(T)$ is k-anonymous, it can be k-partitioned, such that each k-group g_i consists of m k-segments. Recall from Definition 3.3.3 that all entries in a k-segment are necessarily identical. Now, transform $f(T)$ into $f(T)'$ by deleting all k-groups g_i from G that have suppressed entries (*i.e.*, suppress all entries in each k-segment of g_i). Notice that deleting g_i trivially makes all the rows in g_i identical. Notice also that $f(T)'$ is partitioned into the same k-groups as $f(T)$. The existence of a k-partition of $f(T)'$ implies that $f(T)'$ is k-anonymous. ▮

3.3.2 Problems

In this section, a new family of Utility-Preserving k-Anonymous Problems is proposed to overcome deficiencies of the current state of utility-related research (see Section 3.2). Recall from the introduction of Section 3.3 that these problem formulations are superior to previous utility work because they:

1. tailor output to suit specific research needs, and

2. are defined in a way that is clear, concise, and easy to understand.

The aim is to define problems, which are not only useful for researchers, but are tractable as well. Note that we have not mathematically defined utility; however it is treated as the researcher's intuition of utility. Figure 3.4 gives a general abstract overview of solution versions of the family of problems, which are formally defined in Appendix A.

Each member of the family is named following the convention SOL-x-y-z, where x, y, and z are the suppression-location, suppression-quantity, and problem-type variables, respectively. These problems can be broadly categorized by their problem-type variables, such that each problem is either a

- deletion (z =DEL),
- suppression (z =SUP), or
- partition (z =PART) problem.

The region where suppressions may occur in the table is limited by a suppression-location variable, which can restrict the region to a

- number of rows ($x = r$),
- number of columns ($x = c$),
- union of a number of rows and a number of columns ($x = r\text{-}c$), or
- the region may be left unrestricted (x- is omitted).

As problems involving *intersections* of r rows and c columns are just smaller instances of other problems already in the family, they are not considered here. Within each specified region, the amount of suppression is limited by a suppression-quantity variable, which restrict the

- number of suppressed entries ($y = e$),
- number of suppressed entries per row ($y = r'$),
- number of suppressed entries per column ($y = c'$), or
- in the case of a deletion problem, all entries may be suppressed (y- is omitted).

Furthermore, each of these problems can also be broadly categorized by their **orientation**, such that each problem either has a

- row orientation ($x = r$ and/or $y = r'$),
- column orientation ($x = c$ and/or $y = c'$), or
- the orientation may be unspecified ($x \notin \{r, c\}$ and $y \notin \{r', c'\}$).

Several of these problems have already been discussed in this thesis. SOL-e-SUP and SOL-c-DEL appeared in Section 2.2.2 as the two basic suppression problems cited in the literature. SOL-r-DEL was implicitly introduced in Section 3.2 as the problem Samarati [35] refers to as having a cost function which prefers a solution that suppresses the least number of rows. SOL-c'-PART was the problem applied in Figure 3.1 with $k = 2$ and $P = \{\{\texttt{ Age }\}, \{\texttt{ Gender }\}, \{\texttt{ Married }\}, \{\texttt{ Area Code}\}\}$.

As solution problems are related to both decision and optimization problems (see Figure 2.1) solution versions of this family of problems are formally defined in Appendix A. Decision versions of these problems will be analyzed in the next chapter and a similar analysis of optimization versions of these problems will be analyzed in Chapter 5.

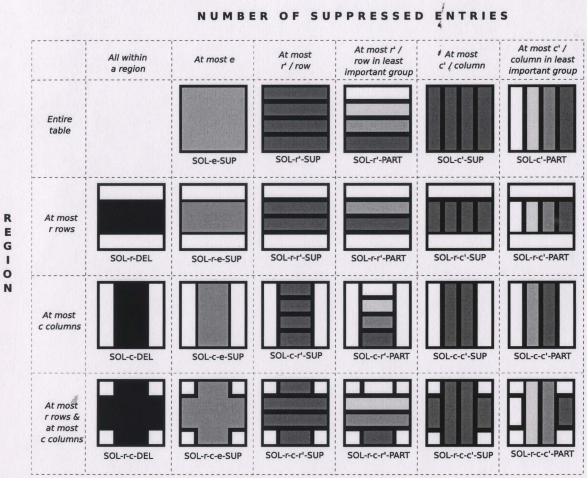


Figure 3.4: A Family of Utility-Preserving k-Anonymity Problems. This is a complete cover of all problems discussed in this thesis. Each problem is named following the convention SOL-x-y-z, where x, y, and z are the suppression-location, suppression-quantity, and problem-type variables, respectively. The problem of interest is SOL-r-c-c'-PART, *i.e.*, satisfy k-anonymity *and* allow researchers to specify the number of rows (study subjects) required for their study, the number of columns (subject characteristics) of interest required for their study, and/or the relative importance of certain columns/characteristics to the study. All other sub-problems are included in order to facilitate systematic analysis (see Section 2.1.4.2).

Chapter 4

Optimal Solutions

This chapter derives optimal tractability and intractability results for the family of problems defined in the previous chapter. Recall that the new framework for analyzing a family of problems proposed in Section 2.1.4.2 involves completing the following set of tasks:

1. Characterize reduction types.
2. Prove the correctness of the template reductions.
3. Systematically acquire all reductions.
4. Create a web of selected reductions.
5. Find (in)tractability results, focusing efforts on roots of (in)tractability.

At this point, some readers may still be skeptical of this new analysis framework, as it is completely backwards from the traditional way of analyzing complexity. However, the framework's inherent reusability, which is apparent in this chapter and the next, justifies its use.

4.1 Reductions

By the end of this section, the first four tasks of analyzing a family of problems will be complete. In Section 4.1.1 the template reductions and their proofs of correctness are given. In Section 4.1.2 all other family reductions are systematically acquired, some of which are selected to form a web of reductions. Finally, in Section 4.1.3, supplementary reductions and their proofs of correctness are given.

4.1.1 Template Reductions

This section will describe four basic types of reductions. For each type, the following details are provided: (1) a characterization of the reduction with respect to applicable problems, (2) a general discussion of how the type of reduction works (see Figure 4.1), (3) a template reduction, and (4) the template reduction's proof of correctness.

For clarity and consistency, two naming conventions are introduced. First, in a reduction characterization, x is a suppression-location variable from a subset of $\{\varepsilon, r, c, r\text{-}c\}$, y is a suppression-quantity variable from a subset of $\{\varepsilon, e, r', c'\}$, and X is a problem-type variable from a subset of {DEL, SUP, PART}. The major elements in a reduction's characterization are emphasized in bold text. Second, for each template reduction $A \leq_m B$, instance variables for problems A and B are subscripted by the letters A and B, respectively. This subscript notation is used throughout the remainder of this thesis.

<u>**TYPE 1**</u> [x-y-X $\leq_m$**r-c**-y-X] ($x \in \{r, c\}, y \in \{\varepsilon, e, r', c'\}, X \in$ {DEL,SUP,PART}): This type of reduction occurs when any problem with suppressed entries restricted

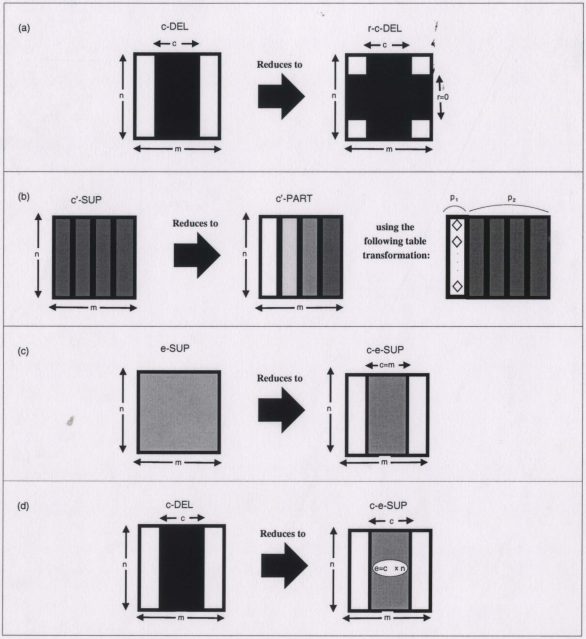


Figure 4.1: Abstract View of Template Reductions. (a) TYPE 1: c-DEL $\leq_m$ r-c-DEL, (b) TYPE 2: c'-SUP $\leq_m$ c'-PART, (c) TYPE 3: e-SUP $\leq_m$ c-e-SUP, (d) c-DEL $\leq_m$ c-e-SUP.

to either c columns or r rows reduces to the same problem, except suppressed entries are restricted to the union of c columns and r rows. The basic idea of the reduction is to either set c or r to zero, so that the constructed instance of the union problem is identical to the strictly r-restricted or c-restricted given instance (see Figure 4.1 (a)). The following is the template reduction and its proof of correctness:

Lemma 4.1.1 c-DEL $\leq_m$ r-c-DEL

Proof: (Proof by restriction) Given an instance $\langle T_A, c_A, k_A \rangle$ of c-DEL, construct the following instance $\langle T_B, r_B, c_B, k_B \rangle$ of r-c-DEL:

$$\begin{aligned} T_B &= T_A \\ r_B &= 0 \qquad (4.1) \\ c_B &= c_A \qquad (4.2) \\ k_B &= k_A \end{aligned}$$

This construction can be done in time polynomial in the size of the given instance of c-DEL. To prove that this reduction is a many-one reduction, it has to be shown that the given instance of c-DEL has a solution $\Leftrightarrow$ the constructed instance of r-c-DEL has a solution.

[$\Rightarrow$] *(If c-DEL = "Yes", then r-c-DEL = "Yes")*

Suppose the given instance of c-DEL has a solution. This implies that there is a table, $f(T)$ that is k-anonymous and that suppressed entries in $f(T)$ can only occur in deleted columns. Let c_{act} be the actual number of deleted columns in $f(T)$. Substitute

c_B from (4.2) into (4.3) to obtain (4.4):

$$c_{act} \leq c_A \tag{4.3}$$

$$c_{act} \leq c_B \tag{4.4}$$

Notice that there are no deleted rows. As there exists a table $f(T)$ that is k-anonymous, the number of deleted rows is at most (actually is equal to) r_B (4.1), and there are up to c_B deleted columns (Equation 4.4), the constructed instance of r-c-DEL has a solution.

[$\Leftarrow$] *(If r-c-DEL = "Yes", then c-DEL = "Yes")*

Suppose the constructed instance of r-c-DEL has a solution. This implies that there is a table, $f(T)$ that is k-anonymous. Suppressed entries only occur in at most c_B deleted columns, since there are no deleted rows (Equation 4.1). Let c_{act} be the actual number of columns where suppressed entries can occur in $f(T)$. Substitute c_A from (4.2) into (4.5) to obtain (4.6):

$$c_{act} \leq c_B \tag{4.5}$$

$$c_{act} \leq c_A \tag{4.6}$$

As we have a table $f(T)$ that is k-anonymous and suppressed entries can only occur in at most c_A deleted columns, the given instance of c-DEL has a solution. ∎

TYPE 2 [x-y-SUP$\leq_m$ x-y-PART] ($x \in \{\varepsilon, r, c\}, y \in \{r', c'\}$): This type of reduction occurs when any suppression problem reduces to its partition version. The basic idea of the reduction is to construct a partition $P = \{p_1, p_2\}$ so that the entire table from the suppression problem's instance makes up p_2. The other partition member p_1 will always have all entries equal to a new symbol $\diamondsuit$ so that no suppressed entries are required to satisfy k-Anonymity (see Figure 4.1 (b)). The following is the template reduction and its proof of correctness:

Lemma 4.1.2 c'-SUP $\leq_m$ c'-PART

Proof: (Proof by quasi-restriction) Given an instance $\langle T_A, c'_A, k_A \rangle$ of c'-SUP, construct the following instance $\langle T_B, P_B, c'_B, k_B \rangle$ of c'-PART:

$$\begin{aligned} c'_B &= c'_A \\ k_B &= k_A \end{aligned} \tag{4.7}$$

Let T_B be an $n_A \times (m_A + 1)$ table, where the last m_A columns of T_B are the m_A columns of T_A and every entry in the first column of T_B is a new symbol $\diamond \notin \Sigma$. Now, let $P_B = \{p_1, p_2\}$ be a column partition of T_B, such that p_2 contains the last m_A columns of T_B and p_1 contains the first column of T_B. This construction can be done in time polynomial in the size of the given instance of c'-SUP. To see that this reduction is a many-one reduction, it has to be shown that the given instance of c'-SUP has a solution $\Leftrightarrow$ the constructed instance of c'-PART has a solution.

[$\Rightarrow$] *(If c'-SUP = "Yes", then c'-PART = "Yes")*

Suppose the given instance of c'-SUP has a solution. This implies that there is a k-anonymous table, $f(T_A)$, such that the number of suppressed entries per column in $f(T_A)$ is at most c'_A. Transform $f(T_A)$ into $f(T_B)$ by adding a new column to $f(T_A)$, where all entries are $\diamond$. According to Lemma 3.3.9, $f(T_B)$ is also k-anonymous. Let

c'_{max} be the maximum number of suppressed entries per column in $f(T_A)$. Substitute c'_B from (4.7) into (4.8) to obtain (4.9):

$$c'_{max} \leq c'_A \tag{4.8}$$

$$c'_{max} \leq c'_B \tag{4.9}$$

Now consider column partition $P_B = \{p_1, p_2\}$; the maximum number of suppressed entries in any column in p_2 has to be c'_B (Equation 4.9). Furthermore, since all the entries in p_1's column are identical, there are no suppressed entries in p_1's column. As there exists a k-anonymous table $f(T_B)$, a partition $P_B = \{p_1, p_2\}$ that is a column utility-partition of $f(T_B)$, and the total number of suppressed entries in any of p_2's columns is at most c'_B, the constructed instance of c'-PART has a solution.

[$\Leftarrow$] *(If c'-PART = "Yes", then c'-SUP = "Yes")*

Suppose the constructed instance of c'-PART has a solution. This implies that there is a k-anonymous table $f(T_B)$ and a column utility-partition of $f(T_B)$, $P = \{p_1, p_2\}$. As P is a column utility-partition, p_1's column has no more than the minimum number of suppressed entries per column in p_2, and the number of suppressed entries in any of p_2's columns is at most c'_B, the number of suppressed entries in any column of T_B is at most c'_B. Let c'_{max} be the maximum number of suppressed entries per column in $f(T_B)$. Substitute c'_A from (4.7) into (4.10) to obtain (4.11):

$$c'_{max} \leq c'_B \tag{4.10}$$

$$c'_{max} \leq c'_A \tag{4.11}$$

As there exists a k-anonymous table $f(T_A)$ and the total number of suppressed entries in any column of $f(T_A)$ is at most c'_A (Equation 4.11), the given instance of c'-SUP has a solution. ∎

Observe that this is a quasi-restriction reduction because the instance of c'-SUP is constructed in such a way that the two instances in the reduction are essentially identical (see Section 2.1.3.2); the constructed instance is exactly the same as the given instance, except $T_B = T_A$+(*one additional column*). We will see in the next section that similar constructions (and hence quasi-restriction reductions) exist when non-partition problems reduce to partition problems. Notice that this is the case for all Type 2 reductions and for several Type 4 reductions.

<u>TYPE 3</u> [y-X $\leq_m$x-y-X] ($x \in \{r, c, r\text{-}c\}, y \in \{e, r', c'\}, X \in \{\text{SUP},\text{PART}\}$): This type of reduction occurs when any problem that has no restriction on the location of suppressed entries reduces to location-restricted version. The basic idea of the reduction is to either set $c = m$ or $r = n$, so that the constructed instance of the location-restricted problem is identical to the unrestricted given instance (see Figure 4.1 (c)). The following is the template reduction and its proof of correctness:

Lemma 4.1.3 e-SUP $\leq_m$ c-e-SUP

Proof: (Proof by restriction) Given an instance $\langle T_A, e_A, k_A\rangle$ of e-SUP, construct the following instance $\langle T_B, c_B, e_B, k_B\rangle$ of c-e-SUP:

$$\begin{aligned} T_B &= T_A \\ c_B &= m_A \end{aligned} \qquad (4.12)$$

$$e_B = e_A \tag{4.13}$$
$$k_B = k_A$$

This construction can be done in time polynomial in the size of the given instance of e-SUP. To see that this reduction is a many-one reduction, it has to be shown that the given instance of e-SUP has a solution $\Leftrightarrow$ the constructed instance of c-e-SUP has a solution.

[$\Rightarrow$] *(If e-SUP = "Yes", then c-e-SUP = "Yes")*

Suppose the given instance of e-SUP has a solution. This implies that there is a table, $f(T)$ that is k-anonymous and the total number of suppressed entries in $f(T)$ is at most e_A. Let e_{act} be the actual number of suppressed entries in $f(T)$ and c_{act} be the actual number of columns where suppressed entries occur in $f(T)$. Substitute e_B from (4.13) into (4.14) to obtain (4.15):

$$e_{act} \leq e_A \tag{4.14}$$
$$e_{act} \leq e_B \tag{4.15}$$

and substitute c_B from (4.12) into (4.16) to obtain (4.17):

$$c_{act} \leq m_A \tag{4.16}$$
$$c_{act} \leq c_B \tag{4.17}$$

As there exists a table $f(T)$ that is k-anonymous, the total number of suppressed entries in $f(T)$ is at most e_B (Equation 4.15), and suppressed entries can only occur in at most c_B columns (Equation 4.17), the constructed instance of c-e-SUP has a solution.

[$\Leftarrow$] *(If* c-e-SUP = *"Yes", then* e-SUP = *"Yes")*

Suppose the constructed instance of c-e-SUP has a solution. This implies that there is a table, $f(T)$ that is k-anonymous and the total number of suppressed entries in $f(T)$ is at most e_B. Let e_{act} be the actual number of suppressed entries in $f(T)$. Substitute e_A from (4.13) into (4.18) to obtain (4.19):

$$e_{act} \leq e_B \tag{4.18}$$

$$e_{act} \leq e_A \tag{4.19}$$

As there exists a table $f(T)$ that is k-anonymous and the total number of suppressed entries is at most e_A (Equation 4.19), the given instance of e-SUP has a solution. ▌

<u>**TYPE 4**</u> [x-DEL$\leq_m$ x-**y**-X] ($x \in \{r, c, r\text{-}c\}, y \in \{e, r', c'\}, X \in \{$SUP,PART$\}$): This type of reduction occurs when any deletion problem reduces to its suppression or partition counterparts. The basic idea of the reduction is to either set e, r', or c' to be the maximum number of suppressed entries allowed within the specified restricted location, so that this constructed instance is identical to deleting rows and/or columns in the given instance (see Figure 4.1 (d)). The following is the template reduction and its proof of correctness:

Lemma 4.1.4 c-DEL $\leq_m$ c-e-SUP

Proof: (Proof by restriction) Given an instance $\langle T_A, c_A, k_A \rangle$ of c-DEL, construct the following instance $\langle T_B, c_B, e_B, k_B \rangle$ of c-e-SUP:

$$T_B = T_A$$

$$c_B = c_A \tag{4.20}$$

$$e_B = c_A \times n_A \tag{4.21}$$

$$k_B = k_A$$

This construction can be done in time polynomial in the size of the given instance of c-DEL. To see that this reduction is a many-one reduction, it has to be shown that the given instance of c-DEL has a solution $\Leftrightarrow$ the constructed instance of c-e-SUP has a solution.

[$\Rightarrow$] *(If c-DEL = "Yes", then c-e-SUP = "Yes")*

Suppose the given instance of c-DEL has a solution. This implies that there is a table, $f(T)$ that is k-anonymous and suppressed entries in $f(T)$ can only occur in deleted columns. Let c_{act} be the actual number of deleted columns in $f(T)$. Substitute c_B from (4.20) into (4.22) to obtain (4.23):

$$c_{act} \leq c_A \tag{4.22}$$

$$c_{act} \leq c_B \tag{4.23}$$

Now, take (4.24) below and multiply both sides by n_A to get (4.25). Substitute e_B from (4.21) into (4.25) to obtain (4.26):

$$c_{act} \leq c_A \tag{4.24}$$

$$c_{act} \times n_A \leq c_A \times n_A \tag{4.25}$$

$$c_{act} \times n_A \leq e_B \tag{4.26}$$

As there exists a table $f(T)$ that is k-anonymous, the total number of suppressed entries in $f(T)$ is at most e_B (Equation 4.26), and suppressed entries can only occur in at most c_B columns (Equation 4.23), the constructed instance of c-e-SUP has a solution.

[$\Leftarrow$] *(If c-e-SUP = "Yes", then c-DEL = "Yes")*

Suppose the constructed instance of c-e-SUP has a solution. This implies that there is a table, $f(T)$ that is k-anonymous, the total number of suppressed entries in $f(T)$ is at most e_B, and suppressed entries can only occur in at most c_B columns. Let c_{act} be the actual number of columns where suppressed entries can occur in $f(T)$. Substitute c_A from (4.20) into (4.27) to obtain (4.28):

$$c_{act} \leq c_B \tag{4.27}$$

$$c_{act} \leq c_A \tag{4.28}$$

Therefore, there exists a table $f(T)$ that is k-anonymous and suppressed entries can only occur in at most c_A columns. Using the construction in Lemma 3.3.11, if $f(T)$ is transformed into $f(T)'$ by deleting a subset of c_A columns, then $f(T)'$ is also k-anonymous. As there exists a table $f(T)'$ that is k-anonymous, suppressed entries only occur in at most c_A columns (Equation 4.28), and the c_A columns are deleted columns (Lemma 3.3.11), the given instance of c-DEL has a solution. ∎

Observe that all template reductions are either by restriction (Lemmas 4.1.1, 4.1.3, and 4.1.4) or quasi-restriction (Lemma 4.1.2). This is an important fact that will be exploited later in Section 5.1.1.

4.1.2 Reduction Web

Recall from Section 2.1.4.2 that the fourth task in analyzing a family of problems is to create a reduction web. To accomplish this task, this section first performs the third task by systematically acquiring all reductions between problems in the family through modification of the template reductions from Section 4.1.1. For each type of reduction, it is explained how to modify the instance construction and proof of correctness (see Lemmas 4.1.1, 4.1.2, 4.1.3, and 4.1.4) to obtain new reductions and their proofs of correctness. Each modification is listed in point-form and is either a substitution (keyword *change*) or addition (keyword *add*) to the template proof. Notice that the modifications follow the same subscript naming convention as was described in the last section. In most cases, the modification is simply a substitution or addition to the instance construction and consequent obvious wording changes in the associated proof of correctness are omitted. The only modifications for correctness proofs that are explicitly stated involve specific equations and lemmas.

The rest of this section is outlined as follows: for each reduction type described in the last section, a list of modification items is given, as well as a table that summarizes the related reductions and their required modifications. The resulting reduction web is discussed thereafter.

<u>TYPE 1: Modifications</u> (Summary in Table 4.1)

- **A**: This modification has two parts: (1) **change** $r_B = 0$ from Equation 4.1 to $r_B = r_A$, and (2) change Equation 4.2 from $c_B = c_A$ to $c_B = 0$. This is necessary because the orientation is changed from columns to rows.

- **B**: Because there is a need to suppress up to e entries instead of suppressing all entries, **add** $e_B = e_A$.

- **C**: Because there is a need to suppress up to r' entries per row instead of suppressing all entries, **add** $r'_B = r'_A$.

- **D**: Because there is a need to suppress up to c' entries per column instead of suppressing all entries, **add** $c'_B = c'_A$.

- **E**: Because these are partition problems instead of deletion problems, **add** $P_B = P_A$.

TYPE 2: Modifications (Summary in Table 4.2)

- **F**: Because the suppressed entries are now restricted to c columns, **add** $c_B = c_A$.

- **G**: Because the suppressed entries are now restricted to r rows, **add** $r_B = r_A$.

- **H**: This modification has four parts: (1) **change** Equation 4.7 from $c'_B = c'_A$ to $r'_B = r'_A$, (2) **change** the construction of T_B to be an $(n_A + k_A) \times m_A$ table, where the last n_A rows of T_B is table T_A and every entry in the first k_A rows of T_B is a new symbol $\diamond \notin \Sigma$, (3) **change** $P_B = \{p_1, p_2\}$ so that p_1 contains the first k_A rows of T_B and p_2 contains the last n_A rows of T_B, and (4) **change** Lemma 3.3.9 to Lemma 3.3.10. This is necessary because the orientation is changed from columns to rows.

Table 4.1: Summary of Type 1 Many-One Reductions

Template	**Type**	**Reduction**	**Modifications**
c-DEL $\leq_m$ r-c-DEL	1	r-DEL $\leq_m$ r-c-DEL	A
		c-e-SUP $\leq_m$ r-c-e-SUP	B
		r-e-SUP $\leq_m$ r-c-e-SUP	A, B
		c-r'-SUP $\leq_m$ r-c-r'-SUP	C
		r-r'-SUP $\leq_m$ r-c-r'-SUP	A, C
		c-c'-SUP $\leq_m$ r-c-c'-SUP	D
		r-c'-SUP $\leq_m$ r-c-c'-SUP	A, D
		c-r'-PART $\leq_m$ r-c-r'-PART	C, E
		r-r'-PART $\leq_m$ r-c-r'-PART	A, C, E
		c-c'-PART $\leq_m$ r-c-c'-PART	D, E
		r-c'-PART $\leq_m$ r-c-c'-PART	A, D, E

Table 4.2: Summary of Type 2 Many-One Reductions

Template	**Type**	**Reduction**	**Modifications**
c'-SUP $\leq_m$ c'-PART	2	c-c'-SUP $\leq_m$ c-c'-PART	F
		r-c'-SUP $\leq_m$ r-c'-PART	G
		r-c-c'-SUP $\leq_m$ r-c-c'-PART	F, G
		r'-SUP $\leq_m$ r'-PART	H
		c-r'-SUP $\leq_m$ c-r'-PART	F, H
		r-r'-SUP $\leq_m$ r-r'-PART	G, H
		r-c-r'-SUP $\leq_m$ r-c-r'-PART	F, G, H

TYPE 3: Modifications (Summary in Table 4.3)

- **I**: Because the orientation is changed from columns to rows, **change** Equation 4.12 from $c_B = m_A$ to $r_B = n_A$.

- **J**: Because the suppressed entries are now restricted to the union of r rows and c columns instead of just c columns, **add** $r_B = n_A$.

- **K**: This modification has two parts: (1) **change** Equation 4.13 from $e_B = e_A$ to $c'_B = c'_A$ and (2) **change** Equation 4.14 from $e_{act} \leq e_A$ to $c'_{max} \leq c'_A$. This is necessary because there is a need to suppress up to c' entries per column instead of suppressing up to e entries anywhere.

- **L**: This modification has two parts: (1) **change** Equation 4.13 from$e_B = e_A$ to $r'_B = r'_A$ and (2) **change** Equation 4.14 from $e_{act} \leq e_A$ to $r'_{max} \leq r'_A$. This is necessary because there is a need to suppress up to r' entries per row instead of suppressing up to e entries anywhere.

- **M**: Because these are partition problems instead of deletion problems, **add** $P_B = P_A$.

TYPE 4: Modifications (Summary in Table 4.4)

- **N**: Because there is a need to suppress up to c' entries per column instead of suppressing up to e entries anywhere, **change** Equation 4.21 from $e_B = c_A \times n_A$ to $c'_B = n_A$.

- **O**: This modification has three parts: (1) **change** the construction of T_B to an $n_A \times (m_A + 1)$ table, where the last m_A columns of T_B are the m_A columns of

Table 4.3: Summary of Type 3 Many-One Reductions

Template	**Type**	**Reduction**	**Modifications**
e-SUP $\leq_m$ c-e-SUP	3	e-SUP $\leq_m$ r-e-SUP	I
		e-SUP $\leq_m$ r-c-e-SUP	J
		c'-SUP $\leq_m$ c-c'-SUP	K
		c'-SUP $\leq_m$ r-c'-SUP	I, K
		c'-SUP $\leq_m$ r-c-c'-SUP	J, K
		r'-SUP $\leq_m$ c-r'-SUP	L
		r'-SUP $\leq_m$ r-r'-SUP	I, L
		r'-SUP $\leq_m$ r-c-r'-SUP	J, L
		c'-PART $\leq_m$ c-c'-PART	K, M
		c'-PART $\leq_m$ r-c'-PART	I, K, M
		c'-PART $\leq_m$ r-c-c'-PART	J, K, M
		r'-PART $\leq_m$ c-r'-PART	L, M
		r'-PART $\leq_m$ r-r'-PART	I, L, M
		r'-PART $\leq_m$ r-c-r'-PART	J, L, M

T_A and every entry in the first column of T_B is a new symbol $\diamond \notin \Sigma$, (2) **add** $P_B = \{p_1, p_2\}$ so that p_1 contains the first column of T_B and p_2 contains the last m_A columns of T_B, and (3) **add** Lemma 3.3.9. This is necessary because the reduction is to a partition problem instead of a suppression problem and it has a column orientation.

- **P**: This modification has three parts: (1) **change** the construction of T_B to an $(n_A + k_A) \times m_A$ table, where the last n_A rows of T_B is table T_A and every entry in the first k_A rows of T_B is a new symbol $\diamond \notin \Sigma$, (2) **add** $P_B = \{p_1, p_2\}$ so that p_1 contains the first k_A rows of T_B and p_2 contains the last n_A rows of T_B and, (3) **add** Lemma 3.3.10. This is necessary because the reduction is to a partition problem instead of a suppression problem and it has a row orientation.

- **Q**: Because there is a need to suppress up to r' entries per row instead of suppressing up to e entries anywhere, **change** Equation 4.21 from $e_B = c_A \times n_A$ to $r'_B = c_A$.

- **R**: This modification has two parts: (1) **change** Equation 4.12 from $c_B = c_A$ to $r_B = r_A$ and (2) **change** Lemma 3.3.11 to Lemma 3.3.12. This is necessary because the orientation is changed from columns to rows.

- **S**: Because the orientation is changed from columns to rows, **change** Equation 4.21 from $e_B = c_A \times n_A$ to $e_B = r_A \times m_A$.

- **T**: Because the orientation is changed from columns to rows and there is a need to suppress up to c' entries per column instead of suppressing up to e entries anywhere, **change** $e_B = c_A \times n_A$ from Equation 4.21 to $c'_B = r_A$.

- **U**: Because the orientation is changed from columns to rows and there is a need to suppress up to r' entries per row instead of suppressing up to e entries anywhere, **change** Equation 4.21 from $e_B = c_A \times n_A$ to $r'_B = m_A$.

- **V**: Because the suppressed entries are now restricted to the union of r rows and c columns instead of just c columns, **change** Equation 4.21 from $e_B = c_A \times n_A$ to $e_B = c_A \times n_A + r_A \times m_A - c_A \times r_A$.

- **W**: This modification has two parts: (1) **add** $r_B = r_A$ and (2) **add** Lemma 3.3.12. This is necessary because the suppressed entries are now restricted to the union of r rows and c columns instead of just c columns.

The resulting many-one reduction web is pictured in Figure 4.2. Notice that not all reductions summarized in Tables 4.1, 4.2, 4.3, and 4.4 are included in the web; recall from Section 2.1.4.2 that in order to complete the fourth task, it suffices to include all problems in the web and enough reductions so that the numbers of intractability and tractability roots are minimized. From the resulting web, e-SUP, c-DEL, c'-SUP, r-DEL, and r'-SUP are the intractability roots. In the next section, we reduce known NP-complete problems to these problems.

4.1.3 Supplementary Reductions

During a back-and-forth analysis within task five, it was discovered that one of the roots of intractability, namely r-DEL, was actually tractable (see Section 4.3). Before dealing with algorithms though, this section accomplishes part of the fifth task in analyzing a family of problems by establishing the supplementary reductions from

Table 4.4: Summary of Type 4 Many-One Reductions

Template	**Type**	**Reduction**	**Modifications**
c-DEL $\leq_m$ c-e-SUP	4	c-DEL $\leq_m$ c-c'-SUP	N
		c-DEL $\leq_m$ c-c'-PART	N, O
		c-DEL $\leq_m$ c-r'-SUP	Q
		c-DEL $\leq_m$ c-r'-PART	P, Q
		r-DEL $\leq_m$ r-e-SUP	R, S
		r-DEL $\leq_m$ r-c'-SUP	R, T
		r-DEL $\leq_m$ r-c'-PART	O, R, T
		r-DEL $\leq_m$ r-r'-SUP	R, U
		r-DEL $\leq_m$ r-r'-PART	P, R, U
		r-c-DEL $\leq_m$ r-c-e-SUP	N, V
		r-c-DEL $\leq_m$ r-c-c'-SUP	V, W
		r-c-DEL $\leq_m$ r-c-c'-PART	O, V, W
		r-c-DEL $\leq_m$ r-c-r'-SUP	U, V
		r-c-DEL $\leq_m$ r-c-r'-PART	P, U, V

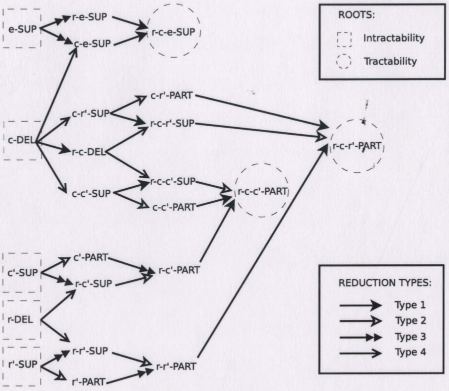


Figure 4.2: Many-One Reduction Web. Selected reductions.

intractable problems for the left-hand side of the reduction web.

In this section, there are four supplementary reductions, accompanied by illustrative examples for the first and fourth reductions (Figures 4.3 and 4.4, respectively). The first reduction below is due to Meyerson and Williams [31]; however, it is restated here for a couple of reasons:

1. There is a mistake (see footnote on page 85) in the conference proceedings [31].

2. The restatement is cleaner and more amenable to reuse in subsequent reductions. For example, a simple modification to the instance construction of the first reduction provides the second and third reductions in this section. Actu-

ally, the second reduction is also due to Meyerson and Williams [31]; however they give an unnecessarily complicated proof sketch. The simplicity of both their proofs of correctness is illustrated in their conference presentation [30], where it is also obvious that the problem they call k-DIMENSIONAL PERFECT MATCHING is actually the following NP-complete problem used in this section.

EXACT COVER BY 3-SETS (X3C) [16, SP2]

Instance: A set X with $|X| = 3q$ and a collection C of 3-element subsets of X.

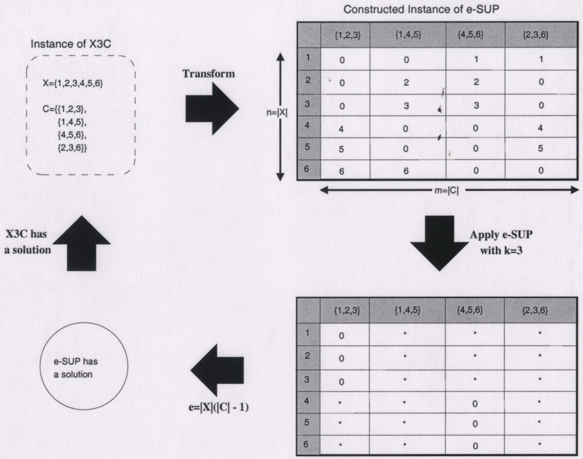
Question: Does C contain an exact cover for X, *i.e.*, a subcollection $C' \subseteq C$ such that every element of X occurs in exactly one member of C'?

The fourth supplementary reduction could not avail of previous work and instead uses the following restricted version of PARTITION INTO TRIANGLES [16, GT11] (The NP-completeness of this restricted problem is noted on Page 2 of [21]):

RESTRICTED PARTITION INTO TRIANGLES (RPT)

Instance: A graph $G = (V, E)$ that has no cliques of size ≥ 4, with $|V| = 3q$ for a positive integer q.

Question: Can the vertices of G be partitioned into q disjoint sets $V_1, V_2, \ldots, V_q$, each containing exactly 3 vertices, such that for each $V_i = \{u_i, v_i, w_i\}$, $1 \leq i \leq q$, all three of the edges $\{u_i, v_i\}$, $\{u_i, w_i\}$, $\{v_i, w_i\}$ belong to E?



	{1,2,3}	{1,4,5}	{4,5,6}	{2,3,6}
1	0	0	1	1
2	0	2	2	0
3	0	3	3	0
4	4	0	0	4
5	5	0	0	5
6	6	6	0	0

	{1,2,3}	{1,4,5}	{4,5,6}	{2,3,6}
1	0	*	*	*
2	0	*	*	*
3	0	*	*	*
4	*	*	0	*
5	*	*	0	*
6	*	*	0	*

Figure 4.3: X3C $\leq_m$ *e*-SUP. Adapted from Meyerson and Williams' conference presentation [30].

Lemma 4.1.5 X3C $\leq_m$ *e*-SUP *[31, Theorem 3.1]*

Proof: Given an instance $\langle X, C\rangle$ of X3C, construct the following instance $\langle T, e, k\rangle$ of *e*-SUP:

$$e = |X|(|C| - 1) \tag{4.29}$$

$$k = 3 \tag{4.30}$$

Construct T as follows. Let the rows of T correspond to the elements of X and let the columns of T correspond to the elements of C. Entry e_{ij} is defined[1] as:

$$e_{ij} = \begin{cases} 0 & \text{if } i \in j \\ i & \text{otherwise} \end{cases}$$

Without loss of generality, assume that there are no repeated 3-sets in C. This construction can be done in time polynomial in the size of the given instance of X3C. To see that this reduction is a many-one reduction, it has to be shown that the given instance of X3C has a solution $\Leftrightarrow$ the constructed instance of e-SUP has a solution.

[$\Rightarrow$] *(If* X3C = *"Yes", then e*-SUP = *"Yes")*

Suppose the given instance of X3C has a solution. This implies that there is a collection C' of 3-sets such that every element of X occurs in exactly one member of C'. Transform T into $f(T)$ as follows: for each of the $|X|$ rows, keep the zero entry corresponding to the element in C' that contains the row and make all other $C - 1$ entries of that row suppressed entries, resulting in exactly $|X|(|C| - 1)$ suppressed entries in total (*i.e.*, the number of rows multiplied by exactly $|C| - 1$ suppressed entries per row). Furthermore, notice that X (and therefore the rows of $f(T)$) can be partitioned into groups of size 3 corresponding to the elements of C'. In $f(T)$, each 3-set is a k-group according to Definition 3.3.2, since each of the rows belonging to a 3-set are identical: zeros in the column corresponding to the 3-set in C' and suppressed entries everywhere else. As $k = 3$ (Equation 4.30), there exists a table $f(T)$ that is k-anonymous and the total number of suppressed entries in $f(T)$ is at most e (Equation 4.29); hence, the constructed instance of e-SUP has a solution.

[1]Meyerson and Williams incorrectly stated "1 otherwise" instead of "i otherwise" in their conference paper [31]; the proof as stated in the original technical report was correct [29].

[$\Leftarrow$] *(If e-SUP = "Yes", then X3C = "Yes")*

Suppose the constructed instance of e-SUP has a solution. This implies that there is a table, $f(T)$ that is 3-anonymous (Equation 4.30) and the total number of suppressed entries in $f(T)$ is at most $|X|(|C|-1)$ (Equation 4.29). Observe, however, that a group of 3 rows must have at least $(|C|-1)$ suppressed entries per row in order for the group to become identical (follows from $e_{i,j} = i$ if $i \notin j$ and the fact that there are no repeated 3-sets in C). This implies that e must equal exactly $|X|(|C|-1)$ and each row in $f(T)$ must have exactly $(|C|-1)$ suppressed entries and therefore one zero. As each row's zero entry corresponds to the 3-set which contains it, the k-partition of $f(T)$ is a partition of X into 3-sets from $|C|$; hence, the given instance of X3C has a solution. ∎

Corollary 4.1.6 X3C $\leq_m$ c-DEL *[31, Theorem 3.2]*

Proof: Given an instance $\langle X, C\rangle$ of X3C, construct the following instance $\langle T, c, k\rangle$ of c-DEL:

$$c = |C| - \frac{|X|}{3} \tag{4.31}$$

$$k = 3 \tag{4.32}$$

Construct T the same way it was constructed in the proof of Lemma 4.1.5, except define entry e_{ij} as:

$$e_{ij} = \begin{cases} 0 & \text{if } i \in j \\ 1 & \text{otherwise} \end{cases}$$

Without loss of generality, again assume that there are no repeated 3-sets in C. This construction can be done in time polynomial in the size of the given instance of X3C. To see that this reduction is a many-one reduction, it has to be shown that the given instance of X3C has a solution $\Leftrightarrow$ the constructed instance of c-DEL has a solution.

[$\Rightarrow$] *(If* X3C = *"Yes", then* c-DEL = *"Yes")*

Suppose the given instance of X3C has a solution. This implies that there is a collection C' of 3-sets such that every element of X occurs in exactly one member of C'. Transform T into $f(T)$ as follows: for each of the $|C|$ columns, delete the column if it is not an element of C'. Notice that X (and therefore the rows of $f(T)$) can be partitioned into groups of size 3 corresponding to the elements of C'. In $f(T)$, each 3-set is a k-group according to Definition 3.3.2, since each of the rows belonging to a 3-set are identical: 0's in the column corresponding to the row's 3-set in C', 1's in the column corresponding to another 3-set in C', and suppressed entries everywhere else. As $k = 3$ (Equation 4.32), there exists a table $f(T)$ that is k-anonymous after deleting columns; hence, the constructed instance of c-DEL has a solution.

[$\Leftarrow$] *(If* c-DEL = *"Yes", then* X3C = *"Yes")*

Suppose the constructed instance of c-DEL has a solution. This implies that there is a table, $f(T)$ that is 3-anonymous (Equation 4.32) and the total number of deleted columns in $f(T)$ is at most $|C| - \frac{|X|}{3}$ (Equation 4.31). Observe, however, that columns not deleted in $f(T)$ must be those which have 0's corresponding to the row's 3-set in C' and 1's corresponding to other 3-sets in C' (follows from the definition of $e_{i,j}$ and the fact that there are no repeated 3-sets in C). As a k-partition of $f(T)$ is a partition of X into 3-sets from $|C|$, the given instance of X3C has a solution. ∎

Corollary 4.1.7 X3C $\leq_m$ r'-SUP

Proof: Given an instance $\langle X, C\rangle$ of X3C, construct the following instance $\langle T, r', k\rangle$ of r'-SUP:

$$\begin{aligned} r' &= |C| - 1 \\ k &= 3 \end{aligned}$$

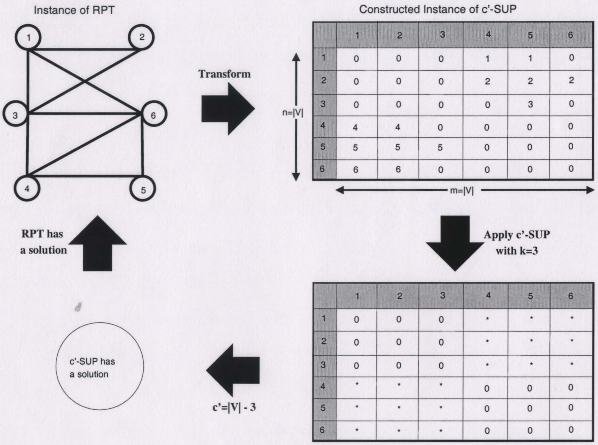
Construct T the same way it was constructed in the proof of Lemma 4.1.5. Observe that in this proof, the constructed instance of e-SUP has a solution if and only if there are exactly $(|C| - 1) = r'$ suppressed entries per row. As the constraint on r' is thus implicit in the proof of correctness of Lemma 4.1.5, the reduction here is correct as well. ∎

Lemma 4.1.8 RPT $\leq_m$ c'-SUP

Proof: Given an instance $\langle G, q\rangle$ of RPT, $G = (V, E)$, construct the following instance $\langle T, c', k\rangle$ of c'-SUP:

$$\begin{aligned} c' &= |V| - 3 && (4.33) \\ k &= 3 \end{aligned}$$

Note that by definition, G is simple and has no complete subgraph with 4 vertices. We construct T as follows. Let both the rows and the columns of T correspond to the $3q$ vertices of G. Entry e_{ij} is defined as:



	1	2	3	4	5	6
1	0	0	0	1	1	0
2	0	0	0	2	2	2
3	0	0	0	0	3	0
4	4	4	0	0	0	0
5	5	5	5	0	0	0
6	6	6	0	0	0	0

	1	2	3	4	5	6
1	0	0	0	*	*	*
2	0	0	0	*	*	*
3	0	0	0	*	*	*
4	*	*	*	0	0	0
5	*	*	*	0	0	0
6	*	*	*	0	0	0

Figure 4.4: RPT $\leq_m$ c'-SUP.

$$e_{ij} = \begin{cases} 0 & \text{if (edge } \{i,j\} \text{ exists) or } (i=j) \\ i & \text{otherwise} \end{cases}$$

Notice that T is effectively an adjacency matrix [18, Section 10.3] for G in which an entry has value i instead of 1 when an edge exists between two vertices. This construction can be done in time polynomial in the size of the given instance of RPT. To see that this reduction is a many-one reduction, it has to be shown that the given instance of RPT has a solution $\Leftrightarrow$ the constructed instance of c'-SUP has a solution.

[$\Rightarrow$] *(If* RPT *= "Yes", then* c'-SUP *= "Yes")*

Suppose the given instance of RPT has a solution. This implies that there is a collection of q disjoint triangles. Consider any triangle (with a, b, and c as the vertices). Transform T into $f(T)$ by making each entry in row x_i and column a_i, $i \in \{a, b, c\}$, a suppressed entry, except the following block of 9 entries: e_{aa},e_{ab},e_{ac},e_{ba},e_{bb},e_{bc},e_{ca},e_{cb},e_{cc}. $f(T)$ then has $|V| - 3$ suppressed entries in each column and all other entries are zeros; that is, q groups of $k = 3$ rows are identical. As we have a table $f(T)$ that is k-anonymous and the total number of suppressed entries per column in $f(T)$ is at most c' (Equation 4.33), the constructed instance of c'-SUP has a solution.

[$\Leftarrow$] *(If* c'-SUP *= "Yes", then* RPT *= "Yes")*

Suppose the constructed instance of c'-SUP has a solution. This implies that there is a table, $f(T)$ that is 3-anonymous and the total number of suppressed entries per column in $f(T)$ is at most $|V| - 3$. Because $f(T)$ is 3-anonymous, the size of any k-group is at least $k = 3$ and at most $2k - 1 = 5$. As T is constructed so that entry values other than zero in each row are different, entry values in at least 3 rows can only match if they are equal to zero. Since there can be at most $c' = |V| - 3$ suppressed entries per column, it follows that there must be at least 3 zeros in every column and hence 3 zeros in every row (rows and columns have the same set of labels). Therefore, each k-group has a block of at least $3 \times 3 = 9$ zeros. Notice that as T is essentially an adjacency matrix, a block of 9 zeros in T defines a triangle in G. For example, a k-group containing rows a, b, and c would have zeros for entries e_{aa},e_{ab}, and e_{ac} in column a, e_{ba},e_{bb}, and e_{bc} in column b, and e_{ca},e_{cb}, and e_{cc} in column c. As a larger block of zeros would imply that G must contain a subgraph which is a complete graph on 4 vertices, the number of suppressed entries in each column must be exactly $|V| - 3$ and the size of each k-group must be exactly 3. Therefore, the rows of $f(T)$ (and hence the rows of T) are necessarily partitioned into $n/3$ k-groups. As this implies that V is partitioned into $|V|/3 = q$ triangles, the given instance of RPT has a solution. ∎

4.2 Intractability Results

Contrary to what one would expect, this section is very short and sums up all intractability results in one theorem:

Theorem 4.2.1 *All k-Anonymity-based decision problems defined in this thesis, except r-DEL, are NP-complete.*

Proof: Proofs of NP-hardness follow from the the polynomial-time many-one reductions from Section 4.1.2, the reducibility properties of Lemma 2.1.14, the NP-

completeness of X3C [16, SP2] and RPT [21, Page 2], and the supplementary reductions given in Lemmas 4.1.5, 4.1.8 and Corollaries 4.1.6, 4.1.7. Note that given a suppression scheme and a set of constraints for any of these problems, the solution can be verified in polynomial-time. As all problems are therefore in **NP**, this completes the proof. ∎

Corollary 4.2.2 *r-DEL is the only k-Anonymity-based decision problem defined in this thesis that has a polynomial-time algorithm, unless* **P=NP**.

Proof: Follows Theorem 4.2.1, the fact that **P⊂NP** unless **P=NP** [16, page 33], and the fact that a problem that is hard for one class cannot have a polynomial-time algorithm from a lower class unless **P=NP** [46, page 14]. ∎

4.3 Algorithms

The fifth and final task in the analysis of a family of problems is to find tractable and intractable problems. Given the intractability results from the previous section, *i.e.*, all problems (except r-DEL) are intractable, a polynomial-time algorithm for r-DEL is described. Unfortunately, as r-DEL is not a tractability root (see Figure 4.2), this does not propagate tractability results to any other problems in the web. In fact, given that all other problems are NP-complete, the best exact algorithms one could hope for (except r-DEL) run in exponential-time[2]. The exponential-time algorithms for each of the reduction web's three tractable roots (*i.e.*, r-c-e-SUP, r-c-r'-PART, and

[2]We know that such exponential-time algorithms exist because for any problem $A \in$ **NP**, there exists a polynomial p such that A can be solved by a deterministic algorithm having time complexity $O(2^{p(n)})$ [16, Theorem 2.1].

r-c-c'-PART) and their time complexities are also described in this section. In doing so, a tighter upper bound is established on the exponential nature of the family of problems.

Lemma 4.3.1 *r-DEL is solvable in $O(nm \log n)$ time.*

Proof: Consider the following algorithm: Given an $n \times m$ table T, sort T, scan T row-wise to look for at least $k-1$ other rows that are identical to the current row, backtrack and mark a total of at most $k-1$ rows if a group of k identical rows is not found, and scan T again to check if there are at most r marked rows. This process runs in time = (*sort time*) + (*scan time*) + (*r-check time*) $= O(nm \log n) + O(nm) + O(nm) = O(nm \log n)$. ∎

This algorithm is essentially a checking routine. Observe that there is only one possible suppression scheme for any particular instance of r-DEL. This is the distinguishing characteristic that makes r-DEL polynomial-time solvable; the above checking routine underlies the exponential-time algorithms described below for other k-Anonymity-based problems, in which all possible suppression schemes are generated and checked. The asymptotic worst-case time complexity analyses for each of those algorithms use the following fact from the sum of the elements in the i^{th} row of Pascal's triangle:

$$\sum_{j=0}^{i} \binom{i}{j} = 2^i$$

Therefore, when $h \leq i$:

$$\sum_{j=0}^{h} \binom{i}{j} \leq 2^i$$

Due to the complicated nature of problems involving unions of rows and columns, when analyzing the complexity of these problems, we split each suppression scheme over a union-selection into two pieces, ensuring that any given suppression scheme is not generated and hence counted in the analysis more than once. Figure 4.5 gives an abstract view of how suppression schemes are split for each analyses.

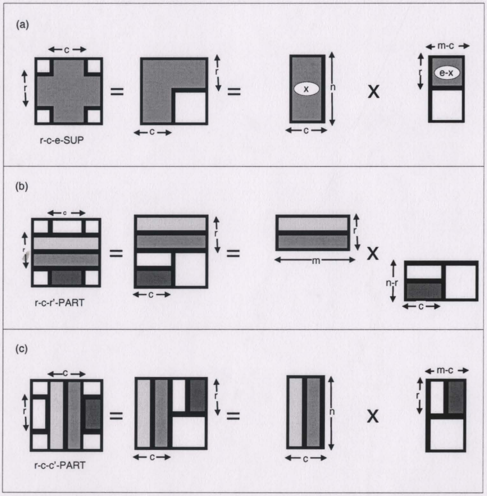


Figure 4.5: Splitting Suppression Schemes to Analyze Time Complexity. This figure demonstates how to reorder and split suppression schemes into two parts in order to simplify the task of determining the total number of possible suppression schemes for (a) r-c-e-SUP, (b) r-c-r'-PART, and (c) r-c-c'-PART.

Lemma 4.3.2 r-c-e-SUP *is solvable in* $O(n^2 m 2^{3nm})$ *time.*

Proof: Consider the following algorithm: Given an $n \times m$ table T, generate all the possible ways to select a union of r rows and c columns from T. For each union-selection, split it into two pieces as shown in Figure 4.5 (a). Generate all the possible ways to suppress up to $x = \min(nc, e)$ entries from the first piece and for each of these suppression schemes, generate all the possible ways to suppress up to $x' = \min(r(m-c), e-x) \leq e-x$ entries from the second piece. Notice that $nc + r(m-c) \leq e \leq nm$. Sort each of the above suppression schemes over the combination of both pieces, and scan each sorted scheme to see if it satisfies k-Anonymity. This process runs in time $=$ (*# possible rows*) (*# possible columns*) (*# possible suppression schemes*) (*sort & scan time*), which is

$$\begin{aligned}
&= \binom{n}{r}\binom{m}{c}\left(\sum_{x=0}^{min(nc,e)}\binom{nc}{x}\sum_{x'=0}^{min(r(m-c),e-x}\binom{r(m-c)}{x'}\right)O(nm\log n)\\
&\leq \binom{n}{r}\binom{m}{c}\left(\sum_{x=0}^{nc}\binom{nc}{x}\sum_{x'=0}^{r(m-c)}\binom{r(m-c)}{x'}\right)O(nm\log n)\\
&\leq (n^r)(m^c)(2^{nc}2^{r(m-c)})O(nm\log n)\\
&= n^r m^c 2^{nc+r(m-c)}O(nm\log n)\\
&\leq 2^n 2^m 2^{nm}O(nm\log n)\\
&\leq 2^{nm+n+m}O(nm\log n)\\
&= O(n^2 m 2^{3nm})
\end{aligned}$$

∎

Lemma 4.3.3 *r-c-r'-PART is solvable in $O(n^2 m 2^{3nm})$ time.*

Proof: Consider the following algorithm: Given an $n \times m$ table T and a row partition $P = \{p_1, p_2, \ldots, p_l\}$, add two new columns a_1 and a_2 to T. Column a_1 will contain the number of suppressed entries in each row and column a_2 will contain the index of the partition member from P that each row belongs to. Generate all the possible ways to select a union of r rows and c columns from T. For each union-selection, split it into two pieces as shown in Figure 4.5 (b). For each row x_i in the first piece, select up to r' entries out of m possible entries and record the number of suppressed entries for row x_i in entry e_{i1}. Similarly, given each of these first-piece selections, for each row in the second piece, select min(c, r') possible entries and record the number of suppressed entries for row x_i in entry e_{i1}. Sort each of the above suppression schemes over the combination of both pieces, and scan each sorted scheme to see if it satisfies k-Anonymity. If this scheme does satisfy k-Anonymity, then for each row x_i, scan P and record the index of the partition member from P that x_i belongs to in entry e_{i2}. Sort a_2 and use it with a_1 to see if the maximum number of suppressed entries for any row in partition member p_j is no more than the minimum number of suppressed entries for any row in partition member p_{j+1}. Notice that $|P| \leq n$, $r \leq n$, and $c \leq m$. As checking for a utility-partition takes time $O(n|P|) + O(n \log n) + O(n) \leq O(n^2)$, this process runs in time = (*# possible rows*) (*# possible columns*) (*# possible suppression schemes*) [(*sort & scan time*) +

(*utility-partition check time*)], which is

$$\begin{aligned}
&= \binom{n}{r}\binom{m}{c}\left(\left(\sum_{i=0}^{r'}\binom{m}{i}\right)^{r}\left(\sum_{i=0}^{min(c,r'}\binom{c}{i}\right)^{n-r}\right)[O(nm\log n)+O(n^2)] \\
&\leq \binom{n}{r}\binom{m}{c}\left(\left(\sum_{i=0}^{r'}\binom{m}{i}\right)^{r}\left(\sum_{i=0}^{c}\binom{c}{i}\right)^{n-r}\right)[O(nm\log n)+O(n^2)] \\
&\leq (n^r)(m^c)\left(2^{mr}2^{c(n-r)}\right)[O(nm\log n)+O(n^2)] \\
&\leq (2^n)(2^m)\left(2^{nm}2^{nm}\right)[O(nm\log n)+O(n^2)] \\
&\leq 2^{2nm+n+m}[O(m(n\log n))+O(n^2)] \\
&\leq 2^{3nm}[O(mn^2)+O(n^2)] \\
&= O(n^2m2^{3nm})
\end{aligned}$$

∎

Lemma 4.3.4 *r-c-c'-PART is solvable in $O(n^2m^22^{3nm})$ time.*

Proof: Consider the following algorithm: Given an $n \times m$ table T and a column partition $P = \{p_1, p_2, \ldots, p_l\}$, add two new rows x_1 and x_2 to T. Row x_1 will contain the number of suppressed entries in each column and row x_2 will contain the index of the partition member from P that each column belongs to. Generate all the possible ways to select a union of r rows and c columns from T. For each union-selection, split it into two pieces as shown in Figure 4.5 (c). For each column a_i in the first piece, select up to c' entries out of n possible entries and record the number of suppressed entries for column a_i in entry e_{1i}. Similarly, given each of these first-piece selections, for each column in the second piece, select min(r, c') possible entries and record the number of suppressed entries for column a_i in entry e_{1i}. Sort each of the above suppression schemes over the combination of both pieces, and scan each sorted scheme to see if it satisfies k-Anonymity. If this scheme does satisfy k-Anonymity, then for each column a_i, scan P and record the index of the partition member from P that a_i belongs to in entry e_{2i}. Sort x_2 and use it with x_1 to see if the maximum number of suppressed entries for any column in partition member p_j is no more than the minimum number of suppressed entries for any column in partition member p_{j+1}. Notice that $|P| \leq m$, $r \leq n$, and $c \leq m$. As checking for a utility-partition takes time $O(m|P|) + O(m \log m) + O(m) \leq O(m^2)$, this process runs in time = (*# possible rows*) (*# possible columns*) (*# possible suppression schemes*) [(*sort & scan time*) + (*utility-partition check time*)], which is

$$\begin{aligned}
&= \binom{n}{r}\binom{m}{c}\left(\left(\sum_{i=0}^{c'}\binom{n}{i}\right)^{c}\left(\sum_{i=0}^{min(r,c'}\binom{r}{i}\right)^{m-c}\right)[O(nm\log n) + O(m^2)] \\
&\leq \binom{n}{r}\binom{m}{c}\left(\left(\sum_{i=0}^{c'}\binom{n}{i}\right)^{c}\left(\sum_{i=0}^{r}\binom{r}{i}\right)^{m-c}\right)[O(nm\log n) + O(m^2)] \\
&\leq (n^r)(m^c)\left(2^{nc}2^{r(m-c)}\right)[O(nm\log n) + O(m^2)] \\
&\leq (2^n)(2^m)\left(2^{nm}2^{nm}\right)[O(nm\log n) + O(m^2)] \\
&\leq 2^{2nm+n+m}[O(m^2 n\log n) + O(m^2)] \\
&\leq 2^{3nm}O(m^2 n\log n) \\
&= O(n^2m^22^{3nm})
\end{aligned}$$

∎

Theorem 4.3.5 *All NP-complete k-Anonymity-based decision problems have asymptotic worst-case time complexity $O(n^2m^22^{12nm})$.*

Proof: Observe that a reduction in the Utility-Preserving k-Anonymity family of problems is either by restriction or quasi-restriction. For each restriction reduction $A \leq_m B$, the $n_B \times m_B$ table T_B in the constructed instance of problem B is identical to the $n_A \times m_A$ table T_A from the given instance of problem A. If the asymptotic worst-case time complexity of B is only in terms of n_B and m_B, A has the same asymptotic worst-case time complexity. Recall from Section 4.1.1 that quasi-restriction reductions are required when non-partition problems reduce to partition problems, which is the case for all Type 2 reductions and for several Type 4 reductions. For each quasi-restriction reduction $A \leq_m B$, if A and B are column-oriented, then $m_B = m_A + 1$, and if A and B are row-oriented, then $n_B = n_A + k_A$, where k_A is the k-Anonymity factor for the given instance of A. These quasi-restrictions only occur when suppression or deletion problems reduce to partition problems. Furthermore, only one such reduction appears on any path from a tractability root to an intractability root in the polynomial-time many-one reduction web (Figure 4.2). Given the above restriction and quasi-restriction properties and the asymptotic worst-case time complexity for tractability roots from Lemmas 4.3.2, 4.3.3, and 4.3.4, and $k \leq n$, all Utility-Preserving k-Anonymity problems are solvable in $O((n+k)^2(m+1)^2 2^{3(n+k)(m+1)}) = O(n^2m^2 2^{3(2n)(m+1)}) = O(n^2m^2 2^{6nm+6m)}) = O(n^2m^2 2^{12nm})$. ∎

4.4 Discussion

This chapter was devoted to systematically proving reductions and summarizing traditional complexity results for the family of problems defined in Chapter 3. As these results are specific to suppression (special case of generalization; see Section 2.2.2), the most natural direction for future research is to extend these results to generalization.

The following is a list of other possible future research:

- Section 4.3 identifies the only existing polynomial-time solvable k-Anonymity-based problem, namely r-DEL. Recall that the solution for an instance of this problem is derived by deleting patient records which are not identical to at least $k-1$ other records. As this solution provides a dataset that may not be representative of the entire patient population, it may not be useful for all types of research; however, given that all proposed k-Anonymity-based problems are NP-complete, the existence of this polynomial-time solvable problem provides insight into how one might refine the frontier of intractability.
- Theorem 4.3.5 states that all NP-complete k-Anonymity-based decision problems have asymptotic worst-case time complexity $O(n^2m^22^{12nm})$. Note that if aspects n and m are small in practice, then these algorithms would run *nearly* in polynomial-time. As n is the number of patient records in a table, it is unlikely that this aspect is small; however, the number of released attributes, m, is very likely to be small in practice. Futhermore, the time complexity expression for any particular problem may not need to be exponential purely in n and m; it may be exponential in other aspects that are small in practice. For example, it may be exponential in k, which is usually a small constant (around 5 or 6) [31, Section 4]. Such research would most naturally fit into the framework of Parameterized Complexity (see [12, 14, 32] for good overviews).

While the reductions in this section were relatively simple, as a collection, they prove to be a powerful tool. This will be substantiated in the next chapter, where approximate solutions for Utility-Preserving k-Anonymity are analyzed.

Chapter 5

Approximate Solutions

This chapter analyzes the approximation complexity of optimization versions of the problems defined in Chapter 3 using the family-analysis framework proposed in Section 2.1.4.2. The power of this new family-analysis framework becomes apparent here; as the many-one reduction web from the last chapter was structured to permit multiple types of complexity results for selected problems to propagate through the family, many of the following results were obtained with minimal additional effort. Section 5.1 addresses the first four family-analysis tasks, Sections 5.2 and 5.3 address the fifth family-analysis task, and Section 5.4 discusses the results obtained and future research.

5.1 Reductions

Adhering to the same process as Chapter 4, by the end of this section, the first four tasks of analyzing a family of problems will be complete. The family-analysis framework proposed in Section 2.1.4.2 allows the many-one reduction types from

Section 4.1.1 to be reused in Section 5.1.1 and the many-one reduction web from Section 4.1.2 to be reused to lay the foundation for a metric reduction web and an L-reduction web in Section 5.1.2. In Section 5.1.3, supplementary reductions and their proofs of correctness are given for intractability roots of these two new reduction webs.

5.1.1 Template Reductions

This section uses the same four basic types of reductions previously described in Chapter 4 (see Section 4.1.1 for details and naming conventions):

- **TYPE 1** [x-y-X $\leq_m$**r**-**c**-y-X] ($x \in \{r, c\}, y \in \{e, r', c'\} \cup \phi, X \in \{$DEL,SUP,PART$\}$)
- **TYPE 2** [x-y-SUP$\leq_m$ x-y-PART] ($x \in \{r, c\} \cup \phi, y \in \{r', c'\}$)
- **TYPE 3** [y-X $\leq_m$**x**-y-X] ($x \in \{r, c, r\text{-}c\}, y \in \{e, r', c'\}, X \in \{$SUP,PART$\}$)
- **TYPE 4** [x-DEL$\leq_m$ x-**y**-X] ($x \in \{r, c, r\text{-}c\}, y \in \{e, r', c'\}, X \in \{$SUP,PART$\}$)

Given that the template many-one reductions from Section 4.1.1 are either by restriction or quasi-restriction, the following is true:

Lemma 5.1.1 *The following metric reductions and L-reductions are correct:*

- **TYPE 1**: MIN-COST-c-DEL = r-MIN-COST-c-DEL, MIN-c-DEL $\leq_L$ r-MIN-c-DEL
- **TYPE 2**: MIN-COST-c'-SUP = MIN-COST-c'-PART, MIN-c'-SUP $\leq_L$ MIN-c'-PART
- **TYPE 3**: MIN-COST-e-SUP = c-MIN-COST-e-SUP, MIN-e-SUP $\leq_L$ c-MIN-e-SUP
- **TYPE 4**: MIN-COST-c-DEL = e-MIN-COST-c-SUP, MIN-c-DEL $\leq_L$ e-MIN-c-SUP

Proof: Follows from the many-one restriction or quasi-restriction reductions (Lemmas 4.1.1, 4.1.2, 4.1.3, and 4.1.4), and the fact that these reductions are trivially metric reductions and L-reductions (Lemma 2.1.17). ∎

Note that to preserve the correctness of these reductions, the costs which are being minimized in both problems must be the same, *i.e.*, the variables immediately to the right of MIN or MIN-COST in each problem's name must be the same. To see why, take MIN-c-DEL $\leq_L$ e-MIN-c-SUP as an example. Even though a proof of correctness may exist for MIN-c-DEL $\leq_L$ e-MIN-c-SUP, a proof of correctness may not necessarily exist for MIN-c-DEL $\leq_m$ c-MIN-e-SUP.

5.1.2 Reduction Web

Recall from Section 4.1.2 that the fourth task in the family-analysis framework is to create a reduction web. To accomplish this task, we first perform the third task by systematically acquiring all L-reductions between pairs of minimization problems belonging to the problem family, which correspond to the many-one reductions (all by either restriction or quasi-restriction) summarized in Section 4.1.2. In this way, L-reductions with $\alpha = \beta = 1$ are easily derived and their proofs of correctness follow from Lemma 2.1.17. Summaries of these L-reductions are given in Tables 5.1, 5.2, 5.3, 5.4 and the resulting L-reduction web is shown in Figure 5.1. All possible minimization problems are included in the web; however, only enough L-reductions are included so that the number of (in)tractability roots are minimized.

Table 5.1: Summary of Type 1 L-reductions

Template	**Type**	**Reduction**
MIN-c-DEL $\leq_L$ r-MIN-c-DEL	1	MIN-r-DEL $\leq_L$ c-MIN-r-DEL
		c-MIN-e-SUP $\leq_L$ r-c-MIN-e-SUP
		r-MIN-e-SUP $\leq_L$ r-c-MIN-e-SUP
		c-MIN-r'-SUP $\leq_L$ r-c-MIN-r'-SUP
		r-MIN-r'-SUP $\leq_L$ r-c-MIN-r'-SUP
		c-MIN-c'-SUP $\leq_L$ r-c-MIN-c'-SUP
		r-MIN-c'-SUP $\leq_L$ r-c-MIN-c'-SUP
		c-MIN-r'-PART $\leq_L$ r-c-MIN-r'-PART
		r-MIN-r'-PART $\leq_L$ r-c-MIN-r'-PART
		c-MIN-c'-PART $\leq_L$ r-c-MIN-c'-PART
		r-MIN-c'-PART $\leq_L$ r-c-MIN-c'-PART
		e-MIN-c-SUP $\leq_L$ r-e-MIN-c-SUP
		e-MIN-r-SUP $\leq_L$ c-e-MIN-r-SUP
		r'-MIN-c-SUP $\leq_L$ r-r'-MIN-c-SUP
		r'-MIN-r-SUP $\leq_L$ c-r'-MIN-r-SUP
		r'-MIN-c-PART $\leq_L$ r-r'-MIN-c-PART
		r'-MIN-r-PART $\leq_L$ c-r'-MIN-r-PART
		c'-MIN-c-SUP $\leq_L$ r-c'-MIN-c-SUP
		c'-MIN-r-SUP $\leq_L$ c-c'-MIN-r-SUP
		c'-MIN-c-PART $\leq_L$ r-c'-MIN-c-PART
		c'-MIN-r-PART $\leq_L$ c-c'-MIN-r-PART

Table 5.2: Summary of Type 2 L-reductions

Template	**Type**	**Reduction**
MIN-c'-SUP $\leq_L$ MIN-c'-PART	2	c-MIN-c'-SUP $\leq_L$ c-MIN-c'-PART
		r-MIN-c'-SUP $\leq_L$ r-MIN-c'-PART
		r-c-MIN-c'-SUP $\leq_L$ r-c-MIN-c'-PART
		MIN-r'-SUP $\leq_L$ MIN-r'-PART
		c-MIN-r'-SUP $\leq_L$ c-MIN-r'-PART
		r-MIN-r'-SUP $\leq_L$ r-MIN-r'-PART
		r-c-MIN-r'-SUP $\leq_L$ r-c-MIN-r'-PART
		c'-MIN-c-SUP $\leq_L$ c'-MIN-c-PART
		c'-MIN-r-SUP $\leq_L$ c'-MIN-r-PART
		r-c'-MIN-c-SUP $\leq_L$ r-c'-MIN-c-PART
		r'-MIN-c-SUP $\leq_L$ r'-MIN-c-PART
		r'-MIN-r-SUP $\leq_L$ r'-MIN-r-PART
		r-r'-MIN-c-SUP $\leq_L$ r-r'-MIN-c-PART
		c-c'-MIN-r-SUP $\leq_L$ c-c'-MIN-r-PART
		c-r'-MIN-r-SUP $\leq_L$ c-r'-MIN-r-PART

Table 5.3: Summary of Type 3 L-reductions

Template	**Type**	**Reduction**
MIN-e-SUP $\leq_L$ c-MIN-e-SUP	3	MIN-e-SUP $\leq_L$ r-MIN-e-SUP
		MIN-e-SUP $\leq_L$ r-c-MIN-e-SUP
		MIN-c'-SUP $\leq_L$ c-MIN-c'-SUP
		MIN-c'-SUP $\leq_L$ r-MIN-c'-SUP
		MIN-c'-SUP $\leq_L$ r-c-MIN-c'-SUP
		MIN-r'-SUP $\leq_L$ c-MIN-r'-SUP
		MIN-r'-SUP $\leq_L$ r-MIN-r'-SUP
		MIN-r'-SUP $\leq_L$ r-c-MIN-r'-SUP
		MIN-c'-PART $\leq_L$ c-MIN-c'-PART
		MIN-c'-PART $\leq_L$ r-MIN-c'-PART
		MIN-c'-PART $\leq_L$ r-c-MIN-c'-PART
		MIN-r'-PART $\leq_L$ c-MIN-r'-PART
		MIN-r'-PART $\leq_L$ r-MIN-r'-PART
		MIN-r'-PART $\leq_L$ r-c-MIN-r'-PART

Table 5.4: Summary of Type 4 L-reductions

Template	**Type**	**Reduction**
MIN-c-DEL $\leq_L$ e-MIN-c-SUP	4	MIN-c-DEL $\leq_L$ c'-MIN-c-SUP
		MIN-c-DEL $\leq_L$ c'-MIN-c-PART
		MIN-c-DEL $\leq_L$ r'-MIN-c-SUP
		MIN-c-DEL $\leq_L$ r'-MIN-c-PART
		MIN-r-DEL $\leq_L$ e-MIN-r-SUP
		MIN-r-DEL $\leq_L$ c'-MIN-r-SUP
		MIN-r-DEL $\leq_L$ c'-MIN-r-PART
		MIN-r-DEL $\leq_L$ r'-MIN-r-SUP
		MIN-r-DEL $\leq_L$ r'-MIN-r-PART
		r-MIN-c-DEL $\leq_L$ r-e-MIN-c-SUP
		r-MIN-c-DEL $\leq_L$ r-c'-MIN-c-SUP
		r-MIN-c-DEL $\leq_L$ r-c'-MIN-c-PART
		r-MIN-c-DEL $\leq_L$ r-r'-MIN-c-SUP
		r-MIN-c-DEL $\leq_L$ r-r'-MIN-c-PART
		c-MIN-r-DEL $\leq_L$ c-e-MIN-r-SUP
		c-MIN-r-DEL $\leq_L$ c-c'-MIN-r-SUP
		c-MIN-r-DEL $\leq_L$ c-c'-MIN-r-PART
		c-MIN-r-DEL $\leq_L$ c-r'-MIN-r-SUP
		c-MIN-r-DEL $\leq_L$ c-r'-MIN-r-PART

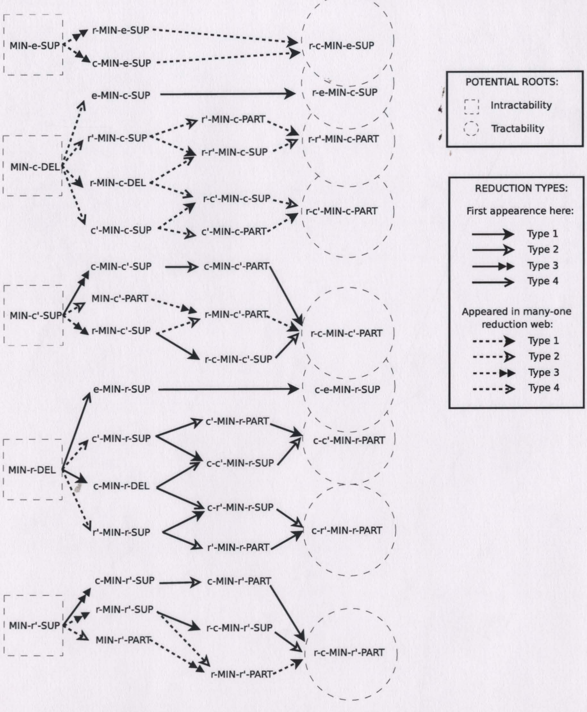


Figure 5.1: L-reduction Web. Selected reductions.

Note that there are many more problems (and hence, reductions) in this web than there were in the many-one reduction web. This is because a decision problem like r-c-e-SUP actually has three corresponding minimization problems, namely r-cMIN-e-SUP, r-eMIN-c-SUP, and c-eMIN-r-SUP. Similarly, this decision problem has three corresponding optimal-cost evaluation problems. The L-reductions between pairs of minimization problems above also apply to similarly-named metric reductions between pairs of optimal-cost evaluation problems (*i.e.*, replace MIN with MIN-COST). As the metric reduction web can be easily derived from the L-reduction web by name replacment, it is not explicitly given here.

In order to complete the fourth task, it suffices to include all problems in the web and enough reductions so that the numbers of intractability and tractability roots are minimized. From the resulting web, MIN-e-SUP, MIN-c-DEL, MIN-c'-SUP, and MIN-r'-SUP are the intractability roots[1]. In the next section, we give three sets of polynomial-time approximation intractability results for these roots.

5.1.3 Supplementary Reductions

This section accomplishes part of the fifth task in the family-analysis framework by establishing supplementary reductions into the left-hand side of the metric reduction and L-reduction webs. Recall the NP-complete decision problems used in the supplementary many-one reductions of the previous chapter, namely EXACT COVER BY 3-SETS (X3C) and RESTRICTED PARTITION INTO TRIANGLES (RPT). We use the following optimization problem counterparts for these decision problems:

[1]Although MIN-r-DEL is in the position of a root, minor changes to the polynomial-time algorithm for its associated decision problem, r-DEL, gives an efficient optimal-solution algorithm for MIN-r-DEL (see Lemma 4.3.1).

BOUNDED MAXIMUM 3-SET PACKING (MAX 3SP-B) [22]

Instance: A set X with $|X| = 3q$ and a collection C of 3-element subsets of X and every element $x \in X$ is contained in at most B subsets in C.

Solution: The largest packing of X, *i.e.*, a subcollection of mutually disjoint sets $C' \subseteq C$.

BOUNDED MAXIMUM TRIANGLE PACKING (MAX TP-B) [22]

Instance: A graph $G = (V, E)$ that has no cliques of size ≥ 4, with $|V| = 3q$ for a positive integer q.

Solution: The largest triangle packing of G, *i.e.*, a subcollection of mutually disjoint sets $V' \subseteq V$, each set containing exactly 3 vertices, such that for each $V_i = \{u_i, v_i, w_i\}$, $1 \leq i \leq q$, all three of the edges $\{u_i, v_i\}$, $\{u_i, w_i\}$, $\{v_i, w_i\}$ belong to E.

Johnson [21, Section 2] describes in more detail how partitioning, covering, and packing problems are often closely related. In fact, Kann [22] originally called the above packing problems (MAX 3SP-B and MAX TP-B) covering problems.

Lemma 5.1.2 *The following are metric reductions:*

1. MAX-COST 3SP-B $= \frac{1}{3(m-1)}$MIN-COST-e-SUP

2. MAX-COST 3SP-B $= \frac{n}{3(m-1)}$MIN-COST-r'-SUP

3. MAX-COST 3SP-B $= m-$MIN-COST-c-DEL

4. MAX-COST 3SP-B $= 1 - 3m + \frac{1}{3}$MIN-COST-c'-SUP

Proof: For any k-Anonymity-based problems, let n and m be the rows and columns of a table, respectively.

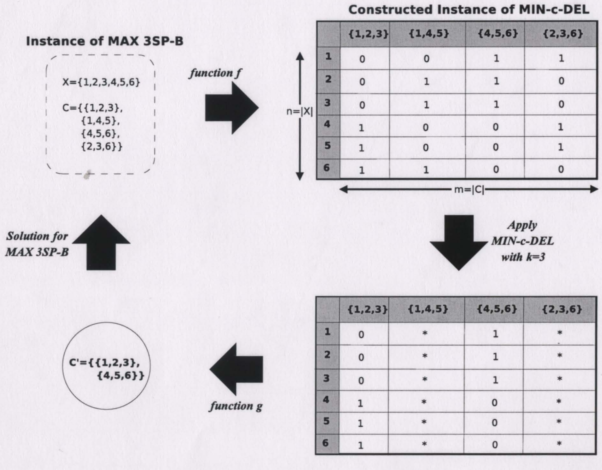
Proof of (1): For any given instance x of MAX 3SP-B, construct instance $R(x)$ of MIN-e-SUP the same way as in Lemma 4.1.5. The cost of any solution y of $R(x)$ is the total number of possible entries to suppress minus the entries corresponding to 3-sets packed into C', *i.e.*, $c_{esup} = nm - 3c_{max3sp-B}$. The metric reduction follows from this equation and optimal solutions $OPT_{max3sp-B} = \frac{n}{3}$ and $OPT_{esup} = n(m-1)$ (*i.e.*, when all elements in set X of instance x are packed into C').

Proof of (2): For any given instance x of MAX 3SP-B, construct instance $R(x)$ of MIN-r'-SUP the same way as in Corallary 4.1.7. The cost of any solution y of $R(x)$ is the total number of possible entries to suppress per row minus the entries corresponding to 3-sets packed into C', such that for the optimal case $c_{r'sup} = m-1$ when $c_{max3sp-B} = \frac{n}{3}$. The metric reduction follows from these optimal solutions (*i.e.*, when all elements in set X of instance x are packed into C').

Proof of (3): For any given instance x of MAX 3SP-B, construct instance $R(x)$ of MIN-c-DEL the same way as in Corallary 4.1.6. The cost of any solution y of $R(x)$ is the total number of possible columns to suppress, which corresponds to 3-sets not packed into C', *i.e.*, $c_{cdel} = m - c_{max3sp-B}$. The metric reduction follows directly from this equation (optimal solutions $OPT_{max3sp-B} = \frac{n}{3}$ and $OPT_{esup} = m - \frac{n}{3}$ occur when all elements in set X of instance x are packed into C').

Proof of (4): For any given instance x of MAX TP-B, construct instance $R(x)$ of MIN-c'-SUP the same way as in Lemma 4.1.8. The cost of any solution y of $R(x)$ is the total number of possible entries to suppress per column minus the entries corresponding to triangles packed into G, such that for the optimal case $c_{c'sup} = n - 3$ when $c_{maxtp-B} = \frac{n}{3}$, *i.e.*, the following metric reduction is correct MAX-COST TP-B $=$ (MIN-COST-c'-SUP $+3)/3$. Substituting this into the metric reduction MAX-COST SP3-B $=$ MAX-COST TP-B$-3m$ [16, page 68] completes the proof. ∎

Figure 5.2 gives an abstract view of the proof of correctness for the next reduction.



	{1,2,3}	{1,4,5}	{4,5,6}	{2,3,6}
1	0	0	1	1
2	0	1	1	0
3	0	1	1	0
4	1	0	0	1
5	1	0	0	1
6	1	1	0	0

	{1,2,3}	{1,4,5}	{4,5,6}	{2,3,6}
1	0	*	1	*
2	0	*	1	*
3	0	*	1	*
4	1	*	0	*
5	1	*	0	*
6	1	*	0	*

Figure 5.2: MAX 3SP-B $\leq_m$ MIN-c-DEL.

Lemma 5.1.3 MAX 3SP-B $\leq_L$ MIN-c-DEL *with* $\alpha = (B-1)$ *and* $\beta = 1$.

Proof: Given an instance $x = \langle X, C \rangle$ of MAX 3SP-B, let function f transform x into an instance $f(x) = \langle T, k \rangle$ of c-DEL in the following way:

$$k \;=\; 3 \tag{5.1}$$

Construct T as follows: let the n rows of T correspond to the elements of $X = \{x_1, x_2, \ldots, x_n\}$ $(n = |X|)$ and let the m columns of T correspond to the elements of $C = \{c_1, c_2, \ldots, c_m\}$ $(m = |C|)$. Entry e_{ij} is defined as:

$$e_{ij} = \begin{cases} 0 & \text{if } x_i \in c_j \\ 1 & \text{otherwise} \end{cases}$$

Without loss of generality, assume that there are no repeated 3-sets in C. Now it must be shown that if the given instance of MAX 3SP-B has a solution, then a solution also exists for the constructed instance of MIN-c-DEL.

Suppose the given instance of MAX 3SP-B has a solution. This implies that there is a collection C' of 3-sets such that every element of X occurs in exactly one member of C'. Transform T into $f(T)$ as follows: for each of the $|C|$ columns, delete the column if it is not an element of C'. Notice that X (and therefore the rows of $f(T)$) can be partitioned into groups of size 3 corresponding to the elements of C'. In $f(T)$, each 3-set is a k-group according to Definition 3.3.2, since each of the rows belonging to a 3-set are identical: 0's in the column corresponding to the row's 3-set in C', 1's in the column corresponding to another 3-set in C', and suppressed entries everywhere else. As $k = 3$ (Equation 5.1), there exists a table $f(T)$ that is k-anonymous after deleting columns, hence the constructed instance of MIN-c-DEL has a solution.

For any instance x of MAX 3SP-B and for any solution y for instance $f(x)$ of MIN-c-DEL, let g be a function which transforms y into a solution $g(x, y)$ for MAX 3SP-B by setting C' to be the collection of labels for columns not deleted in y. Clearly the transformations f and g can be done in time polynomial in the size of the given instance of MAX 3SP-B.

To complete this proof, the following must be shown:

1. $OPT_{3sp-B} \leq \alpha(OPT_{cdel})$

2. $|OPT_{cdel} - c_{cdel}| \leq \beta(|OPT_{3sp-B} - c_{3sp-B}|)$

Since MAX 3SP-B is a maximization problem and MIN-c-DEL is a minimization problem, (2) can be rewritten as: $OPT_{3sp-B} - c_{3sp-B} \leq \beta(c_{cdel} - OPT_{cdel})$. Note the following facts that will be useful in the proofs of (1) and (2) below:

- $n = |X|$.

- $m = |C|$.

- The optimal solution for MAX 3SP-B is when all elements of X are packed into C', *i.e.*, $OPT_{3sp-B} = \frac{|X|}{3}$.

- Given instance $f(x)$, the optimal solution for MIN-c-DEL is also when all elements of X are packed into C', *i.e.*, $OPT_{cdel} = m - \frac{n}{3}$.

- $|C| \leq B(OPT_{3sp-B})$ [22, Proof of Corollary 5].

- $c_{3sp-B} = m - c_{cdel}$ is always true since the cost for any given instance of 3SP-B corresponds to the number of columns not deleted and the cost for any constructed instance of c-DEL corresponds to the number of deleted columns out of a total of m columns.

Proof of (1):

$$\begin{aligned} OPT_{cdel} &= m - \frac{n}{3} \\ &= |C| - \frac{|X|}{3} \\ &\leq B(OPT_{3sp-B}) - \frac{|X|}{3} \\ &\leq B(OPT_{3sp-B}) - OPT_{3sp-B} \\ &\leq (B-1)(OPT_{3sp-B}) \qquad (5.2) \end{aligned}$$

Proof of (2):

$$\begin{aligned} OPT_{3sp-B} - c_{3sp-B} &\leq \beta(c_{cdel} - OPT_{cdel}) \\ c_{3sp-B} &\geq OPT_{3sp-B} - \beta(c_{cdel} - OPT_{cdel}) \\ m - c_{cdel} &\geq OPT_{3sp-B} - \beta(c_{cdel} - OPT_{cdel}) \\ m - c_{cdel} &\geq \frac{|X|}{3} - \beta(c_{cdel} - OPT_{cdel}) \\ m - c_{cdel} &\geq \frac{n}{3} - \beta(c_{cdel} - OPT_{cdel}) \\ m - c_{cdel} &\geq \frac{n}{3} - \beta(c_{cdel} - (m - \frac{n}{3})) \\ m - c_{cdel} - \frac{n}{3} &\geq -\beta(c_{cdel} - m + \frac{n}{3}) \\ -(c_{cdel} - m + \frac{n}{3}) &\geq -\beta(c_{cdel} - m + \frac{n}{3}) \qquad (5.3) \end{aligned}$$

By the above, $\alpha = (B-1)$ and $\beta = 1$, completing the proof. ▮

We speculate that L-reductions MAX 3SP-B$\leq_m$MIN-e-SUP and MAX 3SP-B$\leq_m$MIN-r'-SUP do not exist because their associated supplementary metric reductions in Lemma 5.1.2 have non-constant multiplicative terms. For example, consider MAX 3SP-B$\leq_m$MIN-e-SUP and its associated metric reduction MAX-COST 3SP-B $= \frac{1}{3(m-1)}$MIN-COST-e-SUP. In the proof of correctness for this L-reduction, the equation $OPT_{esup} \leq \alpha(OPT_{max3sp-B})$, would require that constant $\alpha = \frac{1}{3(m-1)}$. We could substitute $B(OPT_{3sp-B})$ in for m, but that would result in a squared term. As L-reduction MAX TP-B$\leq_m$MIN-c'-SUP has an associated metric reduction with a constant multiplicative term, it may be correct; however, as the cost c_{tp-B} cannot not be stated in terms of $c_{c'sup}$, the equation $|OPT_{tp-B} - c_{tp-B}| \leq \beta(|OPT_{c'sup} - c_{c'sup}|)$ is difficult to solve for β.

5.2 Intractability Results

Theorems and corollaries in this section give the first algorithm-independent polynomial-time approximation intractability results for k-Anonymity-based problems.

Theorem 5.2.1 *All k-Anonymity-based NP-hard optimal-cost evaluation problems are $OptP[O(\log n)]$-hard.*

Proof: Follows from the polynomial-time metric reductions in Section 5.1.2, the reducibility property of Lemma 2.1.15, the $OptP[O(\log n)]$-completeness of MAX 3SP [45, page 88], and the supplementary reductions given in Lemma 5.1.2. ∎

Corollary 5.2.2 *No k-Anonymity-based NP-hard optimization problem has a polynomial-bounded absolute approximation algorithm unless* **P=NP**.

Proof: Follows Theorem 5.2.1 and the fact that no $OptP[O(\log n)]$-hard optimal-cost evaluation problem can have a polynomial-bounded absolute approximation algorithm unless **P=NP** [45, Theorem 39]. ∎

Theorem 5.2.3 *No k-Anonymity-based NP-hard optimization problem can have an FPTAS algorithm unless* **P=NP***.*

Proof: Follows from the fact that all these k-Anonymity-based optimization problems are polynomially bounded (*i.e.*, given instance x, which includes an $n \times m$ table T, no solution for x that minimizes e, r', c, or c' can have a cost greater than nm, m, m, n, respectively) and the fact that no NP-hard polynomially bounded optimization problem belongs to the class **FPTAS** unless **P=NP** [7, Theorem 3.15]. ∎

We can can find an even better lower bound on possible approximation algorithms for several k-Anonymity-based problems in the following theorem:

Theorem 5.2.4 *All k-Anonymity-based NP-hard optimization problems branching out from* MIN-*c*-DEL *(including* MIN-*c*-DEL*) in the L-reduction web (Figure 5.1) are APX-hard.*

Proof: Follows from the polynomial-time L-reduction in Section 5.1.2, the reducibility properties of Lemma 2.1.16, the APX-completeness of MAX 3SP-B (the actual statement of APX-completeness for this problem is dispersed throughout the *Com-*

ment section of Ausiello *et al.* [7, SP2]), and the supplementary reduction given in Lemma 5.1.3. ▮

Corollary 5.2.5 *No k-Anonymity-based NP-hard optimization problem branching out from* MIN-*c*-DEL *(including* MIN-*c*-DEL*) in the L-reduction web (Figure 5.1), has a PTAS algorithm unless* **P=NP**.

Proof: Follows Theorem 5.2.4, the fact that **PTAS⊂APX** unless **P=NP** [11, Theorem 6], and the fact that a problem that is hard for one class cannot have an approximation algorithm from a lower class unless **P=NP** [46, page 14]. ▮

5.3 Algorithms

The last task in the analysis of a family of problems is to find tractability results, focusing efforts on tractability roots of tractability. At present, we have only been able to establish polynomial-time approximation algorithms for the problems on the left-hand side of the L-reduction web, *i.e.*, the intractability roots. These algorithms either directly use or modify the best known approximation result for MIN-*e*-SUP, namely the $O(k)$-approximate algorithm due to G. Aggarwal *et al.* [3].

Essentially, G. Aggarwal *et al.*'s algorithm is as follows (see [3, Section 6] for more details): given a table T of MIN-*e*-SUP, create a weighted complete graph $G = (V, E)$, such that each $v_i \in V$ represents a row $x_i \in T$ and each weight $w(e)$ of an edge $e = \{u, v\}$ is the number of attributes not in common between the rows represented by u and v. The **charge** of a vertex is the number of suppressed entries in the row which it represents. The FOREST algorithm [3, Section 6.1] produces a forest

$F = \{T_1, T_2, \ldots, T_r\}$ such that each tree T_i is a directed connected graph which contains no cycles. T_i has at least k vertices and the outdegree of each vertex is ≤ 1. The DECOMPOSE-COMPONENT algorithm [3, Section 6.2] breaks any T_i with $|T_i| > 3k - 3$ into two components of size $\geq k$. The resulting forest is a $(3k-3)$-approximate algorithm for MIN-e-SUP.

Lemma 5.3.1 *[3, Theorem 3] MIN-e-SUP has a polynomial-time $O(k)$-approximate algorithm.*

Proof: As details are relevant for subsequent corollaries and lemmas, a proof sketch (see [3, Theorem 3] for a complete proof) is stated here. Let $E(T_i), V(T_i)$ respectively denote the set of arcs and vertices in tree T_i. The charge on any vertex $v \in V(T_i)$ is at most the total weight of the tree, *i.e.*, $W(T_i) = \sum_{e \in E(T_I)} w(e)$. This upper bound is correct since any attribute for which two rows differ appears on the path between the two vertices that represent those rows. This means that $c_{esup} \leq \sum_i (|T_i| W(T_i))$. Since the size of the largest T_i in F is no more than $3k-3$, $c_{esup} \leq \sum_i ((3k-3)W(T_i)) = (3k-3)(\sum_i W(T_i))$. By construction of the forest, the cost of any feasible forest divided by the total weight of the forest can be no more than the largest tree in the forest, *i.e.*, $c_{esup} / \sum_i W(T_i) \leq OPT_{esup}$; hence, $\sum_i W(T_i) \leq OPT_{esup}$ and therefore $c_{esup} \leq (3k-3)(OPT_{esup})$. As this proves that the cost will always be at most $3k-3$ times the optimal solution's cost, this is a $O(k)$-approximate algorithm. ▮

Corollary 5.3.2 MIN-r'-SUP *has a polynomial-time $O(n)$-approximate algorithm.*

Proof: Given the algorithm in [3, Section 6] and that $\sum_i W(T_i) \leq OPT_{esup}$ from Lemma 5.3.1, let $\frac{OPT_{esup}}{n}$ be the average number of suppressed entries per row, which is less than or equal to the maximum number of suppressed entries per row. Therefore $OPT_{esup} \leq n(OPT_{r'sup})$ and $\sum_i W(T_i) \leq n(OPT_{r'sup})$. The cost for any instance of MIN-r'-SUP is the maximum number of suppressed entries per row in that solution, which must be $\leq max(W(T_i)) \leq \sum_i W(T_i)$; hence, $c_{r'sup} \leq (n)(OPT_{r'sup})$. As this proves that the cost will always be at most n times the optimal solution's cost, this is a $O(n)$-approximate algorithm. ∎

Corollary 5.3.3 MIN-c'-SUP *has a polynomial-time $O(mk)$-approximate algorithm.*

Proof: Given the algorithm in [3, Section 6] and that $c_{esup} \leq (3k-3)(OPT_{esup})$ from Lemma 5.3.1, let $\frac{OPT_{esup}}{m}$ be the average number of suppressed entries per column, which is less than or equal to the maximum number of suppressed entries per column. Therefore $OPT_{esup} \leq m(OPT_{c'sup})$. The cost for any instance of MIN-c'-SUP is the maximum number of suppressed entries per column in that solution, which cannot be more than the total number of suppressed entries, *i.e.*, $c_{c'sup} \leq c_{esup} \leq (3k - 3)(OPT_{esup})$; hence, $c_{c'sup} \leq m(3k-3)(OPT_{c'sup})$. As this proves that the cost will always be at most mk times the optimal solution's cost, this is a $O(mk)$-approximate algorithm. ∎

Corollary 5.3.4 MIN-c-DEL *has a polynomial-time $O(n)$-approximate algorithm.*

Proof: Given the algorithm in [3, Section 6], make the following modification: on each edge $e = \{u, v\} \in E$, along with $w(e)$, record the set of attributes s_{uv} that vertices u and v have in common. For example, if u and v have attributes a_1, a_2, and a_5 in common from the set of all m attributes $A = \{a_1, a_2, a_3, a_4, a_5\}$, then $w(e) = 2$ and $s_{uv} = \{a_1, a_2, a_5\}$. Let $S(T_i) = \bigcap_{u,v \in T_i}(s_{uv})$ be the intersection of all common attributes of tree T_i. Notice that as the cost of any solution to MIN-c-DEL $c_{cdel} = m - |\bigcap_i S(T_i)|$, $w(e) = m - s_{uv}$, and $m - |\bigcap_i S(T_i)| \leq \sum_i W(T_i)$, $c_{cdel} \leq \sum_i W(T_i)$. Because a deleted column is a column with suppressed entries in all n entries, the optimal solution for MIN-e-SUP $OPT_{esup} \leq n(OPT_{cdel})$. Recall from [3, Section 6] that $\sum_i W(T_i) \leq OPT_{esup} \leq n(OPT_{cdel})$. Given this equation and the one above, $c_{cdel} \leq n(OPT_{cdel})$. As this proves that the cost will always be at most n times the optimal solution's cost, this is a $O(n)$-approximate algorithm. ∎

5.4 Discussion

This chapter was devoted to systematically proving reductions and summarizing approximation complexity results for the family of problems defined in Chapter 3. Finding approximation algorithms for problems on the right-hand of the L-reduction web and extending results to generalization problems are the most natural directions for future research. The following is a list of other possible future research:

- Section 5.2 reveals the sort of heuristic approaches to k-Anonymity to stay away from. Recall from Section 2.2.2 that MIN-c-DEL is the optimization-suppression counterpart to one of the basic k-Anonymity problems studied in the literature, namely SOL-k-ANONYMITY ON ATTRIBUTES [39]. As MIN-c-DEL does not have a PTAS and is a special case of (and hence, reduces to) MIN-k-ANONYMITY ON ATTRIBUTES, MIN-k-ANONYMITY ON ATTRIBUTES does not have a PTAS either. This may have implications for the heuristic algorithms proposed in the literature for k-Anonymity-based problems (*e.g.*, [8, 15, 20, 44, 47]). These heuristics need closer examination and more work is required to prove APX-hardness for the other intractability roots, starting with MIN-e-SUP, to investigate if *any* work proposed so far for k-Anonymity has a PTAS.

- Corollary 5.3.4 gives a $O(n)$-approximate solution for MIN-c-DEL. As n is very large in practice and Meyerson and Williams [30, page 19] conjecture that there is an $O(k)$-approximate solution for this problem, a better algorithmic approach than the one taken in this corollar may exist.

- Section 5.3 provides algorithms that put their associated optimization problems in **poly-APX**; however, the APX-hardness result from Section 5.2 only classifies the problem as not being in **PTAS**. A future task is to close this gap for all k-Anonymity-based problems (*i.e.*, for each of these problems, either prove *poly-APX*-hardness, or give an approximation algorithm that will always provide a solution whose cost is at least a constant factor of the optimal cost, thereby placing the associated optimization problem in **APX**).

In the literature, the focus has been minimizing the number of suppressed entries e. Figure 5.1 displays many open optimization problems that have not been analyzed. Although it has been shown for k-Anonymity-based problems that minimizing the number of columns c is not a viable option, perhaps problems which minimize other aspects, such as the number of suppressed entries per row r' or the number of suppressed entries per column c', may be good alternatives. Moreover, to date, optimization problems which maximize k have not been considered at all.

Chapter 6

Conclusions

In this thesis, the problem of preserving personal health information privacy in our growing digital society has been discussed. The concept of a k-anonymous release of this information has been proposed in the literature as a privacy-preserving solution. All known k-Anonymity reseach to date has been unified, so that this thesis may be used as a concrete reference for future research. This thesis shows that previous k-Anonymity-based solutions lack sufficient ability to meet specific researcher needs. To this end, new Utility-Preserving k-Anonymity-based problems have been defined, which better capture the tradeoff between ensuring patient privacy and providing researchers with worthwhile data.

Also, a framework for systematically analyzing the complexity of a family of related problems has been proposed. Many results for k-Anonymity-based problems are systematically derived through this analysis, including two of particular interest: (1) the first known polynomial-time solvable k-Anonymity-based problem and (2) the first known algorithm-independent polynomial-time approximation intractability results for k-Anonymity-based problems.

There are many open k-Anonymity-based problems. Future research particular to traditional and approximation complexity theory has already been described in Sections 4.4 and 5.4, respectively. The following is a list of other potential avenues of research.

- Investigate alternate formulations of k-Anonymity-based problems related to clustering problems on strings (see [13] for a good overview of the area), using some concept of distance between rows in a database table (*e.g.*, Hamming distance).

- Investigate new utility-preserving k-Anonymity heuristics. Heuristics that may be of particular interest are those which only compute the *cost.* For example, a problem whose solution answers the question"What is the k-level?" may be solvable in polynomial time. Results for optimal-cost evaluation problems would have real implications for **privacy advisor** technologies; perhaps optimal solutions are not even required (*e.g.*, research by Krentel [24]).

Finally, it must be remembered that all complexity results derived here are for suppression-only versions of Utility-Preserving k-Anonymity problems. As suppression is a special case of generalization (see Section 2.2.2), all hardness results also apply for their corresponding generalization versions; however, it is very unlikely that the optimal- and approximate-solution algorithms given in Sections 4.3 and 5.3 apply directly in a similar fashion. Hence, another direction for future research is establishing the precise degree of approximate-solution tractability of the generalization versions of the problems treated in this thesis.

Bibliography

[1] C. C. Aggarwal. On k-Anonymity and the curse of dimensionality. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005)*, pages 901–909. ACM Press, 2005.

[2] G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, and A. Zhu. Achieving anonymity via clustering. In *Proceedings of the 25th ACM Symposium on Principles of Database Systems (PODS 2006)-To appear.* ACM Press, 2006.

[3] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Anonymizing tables. In *Proceedings of the 10th International Conference on Database Theory (ICDT 2005)*, volume 3363 of *Lecture Notes in Computer Science*, pages 246–258. Springer, 2005.

[4] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD International Conference of Management of Data*, pages 439–450. ACM Press, 2000.

[5] K. S. Ahamed. *Improved anonymization of data sets.* Honours thesis, Memorial University of Newfoundland, 2006.

[6] M. Atzori, F. Bonchi, F. Giannotti, and D. Pedreschi. Blocking anonymity threats raised by frequent itemset mining. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, pages 561–564. IEEE Computer Society, 2005.

[7] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Potasi. *Complexity and Approximation*. Springer, 1999.

[8] R. Bayardo and R. Agrawal. Data privacy through optimal k-Anonymization. In *Proceedings of 21st International Conference on Data Engineering (ICDE 2005)*, pages 217–228. IEEE Computer Society, 2005.

[9] S. Boggan. *Q. What could a boarding pass tell an identity fraudster about you? A. Way too much.* Guardian unlimited special reports, 2006. `http://www.guardian.co.uk/idcards/story/0,,1766266,00.html`.

[10] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu. Tools for privacy preserving distributed data mining. *Newsletter of the ACM Special Interest Group on Knowledge Discovery and Data Mining*, 4(2):28–34, 2002.

[11] P. Crescenzi, C. Fiorini, and R. Silvestri. Completeness in approximation classes. *Information and Control*, 93:241–262, 1991.

[12] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[13] P. Evans, A. Smith, and T. Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306(1-3):407–430, 2003.

[14] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[15] B. C. M. Fung, K. Wang, and P. S. Yu. Top-down specialization for information and privacy preservation. In *Proceedings of 21st International Conference on Data Engineering (ICDE 2005)*, pages 205–216. IEEE Computer Society, 2005.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[17] W. Gasarch, M. Krentel, and K. Rappoport. OptP as the normal behavior of NP-complete problems. *Mathematical Systems Theory*, 28(6):487–514, 1995.

[18] E. G. Goodaire and M. M. Parmenter. *Discrete Mathematics with Graph Theory, Second Edition.* Prentice Hall, 2002.

[19] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Academic Press, 2001.

[20] V. S. Iyengar. Transforming data to satisfy privacy constraints. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2002)*, pages 279–288. ACM Press, 2002.

[21] D. S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 3(2):182–195, June 1982.

[22] V. Kann. Maximum bounded 3-dimensional matching is *MAX SNP*-complete. *Information Processing Letters*, 37(1):27–35, January 1991.

[23] D. Kifer and J. Gehrke. Injecting utility into anonymized datasets. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data-To appear.* ACM Press, 2006.

[24] M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

[25] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-Anonymity. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 49–60. ACM Press, 2005.

[26] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-Anonymity. In *Proceedings of the 22nd IEEE International Conference on Data Engineering (ICDE 2006)-To appear.* IEEE Computer Society, 2006.

[27] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-Anonymity. In *Proceedings of the 22nd IEEE International Conference on Data Engineering (ICDE 2006)-To appear.* IEEE Computer Society, 2006.

[28] B. Malin. *Trail Re-identification and Unlinkability in Distributed Databases.* PhD thesis, Carnegie Mellon University, 2006.

[29] A. Meyerson and R. Williams. *General k-Anonymization is hard.* Technical Report CMU-CS-03-113. Carnegie Mellon University, 2003.

[30] A. Meyerson and R. Williams. *On the complexity of optimal k-Anonymity.* PODS 2004 presentation, 2004. `http://www.aladdin.cs.cmu.edu/workshops/lamps04/presentations/williams.pdf`.

[31] A. Meyerson and R. Williams. On the complexity of optimal k-Anonymity. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems (PODS 2004)*, pages 223–228. ACM Press, 2004.

[32] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[33] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[34] C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

[35] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

[36] P. Samarati and L. Sweeney. *Protecting Privacy when Disclosing Information: k-Anonymity and its Enforcement through Generalization and Suppression.* Technical Report SRI-CSL-98-04. SRI International, Computer Science Laboratoy, 1998.

[37] L. Sweeney. Guaranteeing anonymity when sharing medical data, the datafly system. In *Conference of the American Medical Informatics Association, Annual Fall Symposium (AMIA1997).* Hanley & Belfus, Inc.

[38] L. Sweeney. *Computational Disclosure Control: A Primer on Data Privacy Protection.* PhD thesis, Massachusetts Institute of Technology, 2001.

[39] L. Sweeney. Achieving k-Anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):571–588, 2002.

[40] L. Sweeney. k-Anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[41] S. Vinterbo. *A note on the hardness of the k-Ambiguity problem.* Technical Report 2002-006. Decision Systems Group, Brigham and Women's Hospital, Harvard Medical School, Boston, 2002.

[42] S. Vinterbo. Privacy: A machine learning approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):939–948, 2004.

[43] K. Wagner and G. Wechsung. *Computational Complexity Theory.* Reidel, 1986.

[44] K. Wang, P. S. Yu, and S. Chakraborty. Bottom-up generalization: A data mining solution to privacy protection. In *Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004)*, pages 249–256. IEEE Computer Society, 2004.

[45] H. T. Wareham. On the computational complexity of inferring evolutionary trees. Master's thesis, Memorial University of Newfoundland, 1993.

[46] H. T. Wareham. *Systematic Parameterized Complexity Analysis in Computational Phonology.* PhD thesis, University of Victoria, 1999.

[47] W. Winkler. *Using simulated annealing for k-Anonymity.* Technical Report 2002-07. US Census Bureau Statistical Research Division, 2002.

[48] X. Xiao and Y. Tao. Personalized privacy preservation. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data-To appear.* ACM Press, 2006.

[49] C. Yao, X. S. Wang, and S. Jajodia. Checking for k-Anonymity violation by views. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005)*, pages 910–921. ACM Press, 2005.

[50] S. Zhong, Z. Yang, and R. N. Wright. Privacy enhancing k-Anonymization of customer data. In *Proceedings of the 24th ACM Symposium on Principles of Database Systems (PODS 2005)*, pages 139–147. ACM Press, 2005.

Appendix A

Utility-Preserving k-Anonymity Problem Definitions

This appendix gives the definitions for the new family of solution problems described in Section 3.3.2. An abstract overview of these problems is illustrated in Figure 3.4.

SOL-e-SUPPRESSION (SOL-e-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers e and k.

Solution: Any k-anonymous table $f(T)$ that has at most e suppressed entries.

SOL-r'-SUPPRESSION (SOL-r'-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers r', k.

Solution: Any k-anonymous table $f(T)$ that has at most r' suppressed entries per row.

SOL-r'-PARTITION (SOL-r'-PART)

Instance: An $n \times m$ table T over an alphabet Σ, a row partition $P = \{p_1, p_2, \ldots, p_l\}$ of T, and positive integers r', k.

Solution: Any k-anonymous table $f(T)$ having a row utility-partition $f(P) = \{p_1, p_2, \ldots, p_l\}$ (generated by applying P to $f(T)$) such that there are at most r' suppressed entries per row in p_l.

SOL-c'-SUPPRESSION (SOL-c'-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers c', k.

Solution: Any k-anonymous table $f(T)$ that has at most c' suppressed entries per column.

SOL-c'-PARTITION (SOL-c'-PART)

Instance: An $n \times m$ table T over an alphabet Σ, a column partition $P = \{p_1, p_2, \ldots, p_l\}$ of T, and positive integers c', k.

Solution: Any k-anonymous table $f(T)$ having a column utility-partition $f(P) = \{p_1, p_2, \ldots, p_l\}$ (generated by applying P to $f(T)$) such that there are at most c' suppressed entries per column in p_l.

SOL-r-DELETION (SOL-r-DEL)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers r, k.

Solution: Any k-anonymous table $f(T)$ such that suppressed entries only occur in at most r deleted rows.

SOL-r-e-SUPPRESSION (SOL-r-e-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers r, e, k.

Solution: Any k-anonymous table $f(T)$ that has at most e suppressed entries such that the suppressed entries only occur in at most r rows.

SOL-r-r'-SUPPRESSION (SOL-r-r'-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers r, r', k.

Solution: Any k-anonymous table $f(T)$ that has at most r' suppressed entries per row such that the suppressed entries only occur in at most r rows.

SOL-r-r'-PARTITION (SOL-r-r'-PART)

Instance: An $n \times m$ table T over an alphabet Σ, a row partition $P = \{p_1, p_2, \ldots, p_l\}$ of T, and positive integers r, r', k.

Solution: Any k-anonymous table $f(T)$ having a row utility-partition $f(P) = \{p_1, p_2, \ldots, p_l\}$ (generated by applying P to $f(T)$) such that there are at most r' suppressed entries per row in p_l and the suppressed entries only occur in at most r rows.

SOL-r-c'-SUPPRESSION (SOL-r-c'-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers r, c', k.

Solution: Any k-anonymous table $f(T)$ that has at most c' suppressed entries per column such that the suppressed entries only occur in at most r rows.

SOL-r-c'-PARTITION (SOL-r-c'-PART)

Instance: An $n \times m$ table T over an alphabet Σ, a column partition $P = \{p_1, p_2, \ldots, p_l\}$ of T, and positive integers r, c', k.

Solution: Any k-anonymous table $f(T)$ having a column utility-partition $f(P) = \{p_1, p_2, \ldots, p_l\}$ (generated by applying P to $f(T)$) such that there are at most c' suppressed entries per column in p_l and the suppressed entries only occur in at most r rows.

SOL-c-DELETION (SOL-c-DEL)

Instance: An $n \times m$ table T over an alphabet Σ and positives integers c and k.

Solution: Any k-anonymous table $f(T)$ that has at most c deleted columns[1].

SOL-c-e-SUPPRESSION (SOL-c-e-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers c, e, k.

Solution: Any k-anonymous table $f(T)$ that has at most e suppressed entries such that the suppressed entries only occur in at most c columns.

SOL-c-r'-SUPPRESSION (SOL-c-r'-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers c, r', k.

Solution: Any k-anonymous table $f(T)$ that has at most r' suppressed entries per row such that the suppressed entries only occur in at most c columns.

SOL-c-r'-PARTITION (SOL-c-r'-PART)

Instance: An $n \times m$ table T over an alphabet Σ, a row partition $P = \{p_1, p_2, \ldots, p_l\}$ of T, and positive integers c, r', k.

Solution: Any k-anonymous table $f(T)$ having a row utility-partition $f(P) = \{p_1, p_2, \ldots, p_l\}$ (generated by applying P to $f(T)$) such that there are at most r' suppressed entries per row in p_l and the suppressed entries only occur in at most c columns.

SOL-c-c'-SUPPRESSION (SOL-c-c'-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers c, c', k.

Solution: Any k-anonymous table $f(T)$ that has at most c' suppressed entries per column such that the suppressed entries only occur in at most c columns.

SOL-c-c'-PARTITION (SOL-c-c'-PART)

Instance: An $n \times m$ table T over an alphabet Σ, a column partition $P = \{p_1, p_2, \ldots, p_l\}$ of T, and positive integers c, c', k.

Solution: Any k-anonymous table $f(T)$ having a column utility-partition $f(P) = \{p_1, p_2, \ldots, p_l\}$ (generated by applying P to $f(T)$) such that there are at most c' suppressed entries per column in p_l and the suppressed entries only occur in at most c columns.

SOL-r-c-DELETION (SOL-r-c-DEL)

Instance: An $n \times m$ table T over an alphabet Σ and a positive integers r, c, k.

Solution: Any k-anonymous table $f(T)$ such that suppressed entries only occur in a region defined by the union of at most c deleted columns and at most r deleted rows.

SOL-r-c-e-SUPPRESSION (SOL-r-c-e-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers r, c, e, k.

Solution: Any k-anonymous table $f(T)$ that has at most e suppressed entries such that the suppressed entries only occur in a region defined by the union of at most c columns and at most r rows.

SOL-r-c-r'-SUPPRESSION (SOL-r-c-r'-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers r, c, r', k.

Solution: Any k-anonymous table $f(T)$ that has at most r' suppressed entries per row such that the suppressed entries only occur in a region defined by the union of at most c columns and at most r rows.

SOL-r-c-r'-PARTITION (SOL-r-c-r'-PART)

Instance: An $n \times m$ table T over an alphabet Σ, a row partition $P = \{p_1, p_2, \ldots, p_l\}$ of T, and positive integers r, c, r', k.

Solution: Any k-anonymous table $f(T)$ having a row utility-partition $f(P) = \{p_1, p_2, \ldots, p_l\}$ (generated by applying P to $f(T)$) such that there are at most r' suppressed entries per row in p_l and the suppressed entries only occur in a region defined by the union of at most c columns and at most r rows.

SOL-r-c-c'-SUPPRESSION (SOL-r-c-c'-SUP)

Instance: An $n \times m$ table T over an alphabet Σ and positive integers r, c, c', k.

Solution: Any k-anonymous table $f(T)$ that has at most c' suppressed entries per column such that the suppressed entries only occur in a region defined by the union of at most c columns and at most r rows.

SOL-r-c-c'-PARTITION (SOL-r-c-c'-PART)

Instance: An $n \times m$ table T over an alphabet Σ, a column partition $P = \{p_1, p_2, \ldots, p_l\}$ of T, and positive integers r, c, c', k.

Solution: Any k-anonymous table $f(T)$ having a column utility-partition $f(P) = \{p_1, p_2, \ldots, p_l\}$ (generated by applying P to $f(T)$) such that there are at most c' suppressed entries per column in p_l and the suppressed entries only occur in a region defined by the union of at most c columns and at most r rows.