EXPRESSIVENESS, EFFICIENCY, AND PRIVACY IN ADVERTISING AUCTIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

David John Martin

January 2009

EXPRESSIVENESS, EFFICIENCY, AND PRIVACY IN ADVERTISING AUCTIONS

David John Martin, Ph.D.

Cornell University 2009

Internet search results are a growing and highly profitable advertising platform. Search providers auction advertising slots to advertisers on their search result pages. Due to the high volume of searches and the users' low tolerance for search result latency, it is important to resolve these auctions quickly. Current approaches restrict the expressiveness of bids in order to achieve fast winner determination, which is the problem of allocating slots to advertisers so as to maximize the expected revenue given that advertisers are charged what they bid. The goal of this work is to permit more expressive bidding, thus allowing advertisers to achieve complex advertising goals, while still providing fast and scalable techniques for winner determination. To this end, we allow advertisers to submit programs that express complex and dynamic bidding strategies. We provide techniques for reducing the amount of program evaluation necessary to solve the winner determination problem, and we study the complexity of sharing aggregation computations between these programs. In addition, we also examine the problem of providing advertisers with data about search auctions without disclosing too much about any individual. We provide algorithms for both checking and enforcing privacy in this context.

BIOGRAPHICAL SKETCH

David Martin was born in Bombay, India in 1979. He attended Campion High School and Jai Hind Junior College there. Between 1997 and 2001, he majored in Computer Science at Carnegie Mellon University in Pittsburgh, Pennsylvania, where he also earned an M.S. in Mathematics through the Honors Program. He worked at Microsoft in Redmond, Washington, for two years before attending graduate school at Cornell University in Ithaca, New York. After finishing his Ph.D. in Computer Science under Joseph Halpern, he will move to Mountain View, California, to work at Google.

To Milli and Mole

ACKNOWLEDGEMENTS

I am extremely grateful to my advisor Joe Halpern and my co-advisor Johannes Gehrke for their patient supervision, great advice, and constant support over the last few years. I simply could not have done this without them. I would also like to thank Richard Shore and Dexter Kozen whose classes I have enjoyed immensely. Thanks to my officemates and peers at Cornell for their company and friendship. I am indebted to James Cummings, Ernest Schimmerling, Bill Hrusa, and Victor Mizel at Carnegie Mellon University for starting me on this path. And, of course, my deepest thanks to my family: to my sister, for being my best friend and the most amazingly wonderful person I know; to my mother, for always believing in me and seeing the best in me, no matter what; to my brother, for his constant readiness to help me out, even if it means driving 3,000 miles; to my father, for introducing me to computers and programming when I was eight; and to my grandmother, for her prayers which have apparently worked.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

OVERVIEW

With the huge number of Internet searches performed every day, search result pages have become a thriving advertising platform. The results of a search query are presented to the user as a web page that contains a limited number of slots for advertisements (typically between four and twenty). On each search result page, major search engines, like Google and Yahoo, sell these slots to advertisers via an auction mechanism that charges an advertiser only if a user clicks on his ad. Most of Google's multi-billion dollar revenue, and more than half of Yahoo's revenue, comes from these so-called sponsored search auctions [30]; and this market is growing quickly. By 2008, spending by US firms on sponsored search is expected increase by $3.2 billion from 2006 and will exceed $9.6 billion, the amount spent on all of online advertising in 2004 [31]. Furthermore, 44% of the current search engine advertisers joined the market within the last two years [31]. With the increasing market size in mind, it is natural to approach sponsored search auctions from a database perspective in order to tackle issues of scalability and expressiveness. This thesis is a step in this direction.

Due to the high volume of searches and the users' low tolerance for search result latency, it is imperative to resolve these auctions fast. Current approaches restrict the expressiveness of bids in order to achieve fast winner determination, which is the problem of allocating slots to advertisers so as to maximize the expected revenue given the advertisers' bids. In Chapter 2, we look at the problem of permitting more expressive bidding, thus allowing advertisers to achieve complex advertising goals, while still providing fast and scalable techniques for winner determination. We allow advertisers to submit bidding programs and provide techniques for finding the winning programs efficiently. The material in Chapter 2 is joint work with Johannes Gehrke and Joseph Halpern. A preliminary version of the work in Chapter 2 appears in [58].

In Chapter 3, we consider various extensions to the techniques proposed in Chapter 2, such as modeling the case where the probability of receiving a click depends on the advertisers placed in surrounding slots, allowing advertisers to bid on blocks of slots in various slot layouts, and dealing with the budget uncertainty arising from ads that have been displayed from previous auctions but have not received clicks yet. We also examine applications of our techniques to other settings for advertisement auctions, such as massively multiplayer online games, and map searches. The material in Chapter 3 is joint work with Johannes Gehrke and Joseph Halpern, and portions of this work appears in [58].

The high volume of searches presents an opportunity for sharing the work required to resolve multiple auctions that occur simultaneously. In Chapter 4, we introduce the problem of shared winner determination and provide techniques for sharing work between multiple search auctions using shared aggregation and shared sort. Our analysis suggests a general framework which we use to study the complexity of optimally sharing various abstract aggregation operators that might be used in bidding programs. The material in Chapter 4 is joint work with Joseph Halpern, with Mingsheng Hong and Walker White providing many useful discussions.

In order to enable advertisers to improve their bidding strategies, it would be helpful if search providers release historical search and auction data. However, they must do so in such a way as to limit the disclosure about any individual advertiser while still providing as much useful information as possible. In

Chapter 5, we consider the problem of controlling the release of data to advertisers so that they can learn about the market they are participating in and improve their bidding strategies, while at the same time not disclosing too much information about any given advertiser. The material in Chapter 5 is joint work with Daniel Kifer, Ashwin Machanavajjhala, Johannes Gehrke, and Joseph Halpern. A preliminary versions of this work appear in [59] and [60].

The work on this thesis was supported in part by NSF under Grants IIS-0534064, IIS-0534404, and IIS-0725260, and by grants from Microsoft.

CHAPTER 2

TOWARD EXPRESSIVE AND SCALABLE SPONSORED SEARCH AUCTIONS

2.1 Introduction

With the huge number of Internet searches performed every day, search result pages have become a thriving advertising platform. The results of a search query are presented to the user as a web page that contains a limited number of slots for advertisements (typically between four and twenty). On each search result page, major search engines, like Google and Yahoo, sell these slots to advertisers via an auction mechanism that charges an advertiser only if a user clicks on his ad. Most of Google's multi-billion dollar revenue, and more than half of Yahoo's revenue, comes from these so-called sponsored search auctions [30]; and this market is growing quickly. By 2008, spending by US firms on sponsored search is expected increase by \$3.2 billion from 2006 and will exceed \$9.6 billion, the amount spent on all of online advertising in 2004 [31]. With the increasing market size in mind, it is natural to approach sponsored search auctions from a database perspective in order to tackle issues of scalability and expressiveness. This work is a first step in this direction.

Sponsored search auctions currently work as follows:

1. **Bid submission.** Advertisers submit bids on clicks for certain keywords offline.
2. **User search.** A user submits a search query.

3. **Winner determination.** Slots are assigned to advertisers by the search provider based on the advertisers' bids.

4. **User action.** The search result page is returned to the user who may now click on one or more of the sponsored links.

5. **Pricing and payment.** The search provider charges an advertiser according to some pricing rule if the user clicks on the advertiser's sponsored link.

The speed of the winner determination in Step 3 is crucial. Since the winning ads are displayed on the search result page, winner determination must be done before the page can be returned to the user. In current sponsored search auctions, this winner determination can be done quickly because advertisers are limited to submitting a single bid on whether or not the user clicks on their ad.

Unfortunately, the limited bidding in current sponsored search auctions is insufficient to meet advertisers' needs in two respects:

1. **Bidding on Multiple Features.** Once the advertisers' ads are displayed on the search results page, the user who submitted the query may click on the ad and may even make a purchase as a result. Advertisers clearly value purchases because they represent immediate revenue. They also value clicks on their ads because they indicate potential customers. However, even if the user does not click on or buy something, advertisers might place value on having their ads displayed simply because this increases their chance to make an impression on the customer. Advertisers who value brand awareness may wish their ads to be placed in prominent positions. Such advertisers may prefer their ads to be displayed near the top

or bottom of the list, but not in the middle. Other advertisers whose goals are to be perceived as the leaders in their markets may wish their ads to be displayed in the topmost slot or not displayed at all. Thus it is clear that advertisers have valuations on clicks, purchases, and slot positions.

Unfortunately, in current search advertising platforms, advertisers are restricted to bidding only on whether they receive a click on their ad. We call this a *single-feature auction*, since the advertisers can express their valuations on only one feature, namely, receiving a click. Our goal is to support *multi-feature auctions* that would allow advertisers to express valuations on multiple features, namely, clicks, purchases, and slot positions. Extending bidding to multiple features is non-trivial; whereas previously the advertiser submitted a single value as depicted in Figure 2.1, now the advertiser can submit a whole table of values for the different combinations of features, as depicted in Figure 2.2. The fast algorithms for winner determination that are currently used by Google and Yahoo! do not extend to non-trivial multi-feature auctions. Moreover, even for single-feature auctions, these algorithms can correctly deal with only a restricted situation, namely, one where the expected number of clicks on an ad is "separable" into the product of an advertiser-specific factor and a slot-specific factor.

2. **Dynamic Bidding Strategies.** The language that search providers such as Google and Yahoo currently use to let advertisers express their bidding preferences is rather limited. While the language does allow advertisers to specify a limited number of parameters to constrain their bids (such as a daily budget, and geographic targets), the language is often insufficiently expressive for serious advertisers to express their preferences and how they change over time. To deal with this, advertisers employ the ser-

vices of various third-party ad-campaign management companies (such as iProspect, SureHits, Atlas, etc.) that monitor the outcomes of auctions and periodically resubmit bids on behalf of the advertiser in an attempt to approximate the advertisers' preferences as much as possible. The kinds of goals that they try to achieve include maintaining a specified slot position during certain hours of the day, maintaining a slot position above a specified competitor, and equalizing the return on investment (ROI) across multiple keywords. The success of such ad-campaign management companies demonstrates the desire among advertisers for more complex expressive bidding in search auctions. Again, advertisers want these, but can only pick from a set of pre-defined strategies that these companies provide.

With the increasing market size in mind, our goal is to design a framework that allows huge numbers of advertisers to bid on a richer set of features using dynamic bidding strategies while simultaneously allowing the search provider to determine winners quickly. We approach sponsored search auctions from a database perspective, and tackle issues of scalability and expressiveness. Our main contribution is an efficient and scalable infrastructure that permits much more expressive bidding than is currently available. In particular, we provide

- a simple but rich language that allows advertisers to express their high-level bidding strategies as *bidding programs* which take as input the search query and various statistics about auction history and performance, and output bids on clicks, purchases, and slot positions (Section 2.2);
- an efficient, scalable, and parallelizable algorithm to solve winner determination given the bids output by the bidding programs (Section 2.3); and

- techniques to reduce the amount of work necessary for evaluating the bidding programs of multiple advertisers (Section 2.4).

This gives advertisers direct and fine-grained control over their advertising strategies, as opposed to limiting them to a menu of pre-defined goals, without sacrificing speed and scalability of solving winner determination. We evaluate our techniques experimentally in Section 2.5, and we conclude in Section 2.6.

2.2 Bidding Strategies as Programs

In this section, we formalize the notion of bidding on multiple features, and we propose a simple language for dynamic strategies that bid on these features.

2.2.1 Multiple Features

Recall that traditionally an advertiser could only bid on one property of the outcome, namely, whether his ad received a click. Now we would like to allow advertisers to bid on additional properties as well, namely whether a purchase was made, and whether his ad was displayed within a desired set of slots. To each advertiser, we make available the following predicates that indicate whether or not the outcome has one of these desired properties.

1. $Slot_j$, indicating that the advertiser gets slot j, for $j \in \{1, \ldots, k\}$, with k being the number of slots.
2. $Click$, indicating that the user clicked on the advertiser's ad.

Table 2.1: Single-feature Valuation

Click	value
Y	3

3. *Purchase*, indicating that the user made a purchase via a link from the advertiser's ad.

Conceptually, the advertiser associates a value with each truth assignment to these predicates, as depicted in Figure 2.2. However, the size of such a representation is exponential in the number of predicates. So we represent bids as OR-bids on Boolean combinations of predicates instead. That is, we let the advertiser fill in a *Bids table* where each row corresponds to a Boolean formula of predicates and the amount that he is willing to pay should that formula be true. If multiple formulas are true, the advertiser can be charged the sum of the corresponding amounts. For example, the Bids table depicted in Table 2.3 indicates that the advertiser is willing to pay 5 cents if he gets a purchase; 2 cents if his ad is displayed in either positions 1 or 2; and 7 cents if he gets a purchase *and* his ad is displayed in positions 1 or 2.

2.2.2 Dynamic Strategies

As we said, we are interested in designing a programming language that lets advertisers express more complex preferences, which may change over time. Instead of providing advertisers with a pre-defined selection of advertising strategies, we let the advertisers submit their bidding strategies as programs for the search provider to run. Conceptually, each time a user submits a search query

Table 2.2: Multi-feature Valuation

$Purchase$	$Click$	$Slot_1$	$Slot_2$	$Slot_3$	value
Y	Y	Y	N	N	7
N	Y	Y	N	N	2
⋮	⋮	⋮	⋮	⋮	⋮
Y	Y	N	N	Y	5
N	Y	N	N	Y	0
⋮	⋮	⋮	⋮	⋮	⋮

Table 2.3: Bids Table

formula	value
$Purchase$	5
$Slot_1 \vee Slot_2$	2

to the search provider, these programs are triggered. The main purpose of these programs is to output bids on clicks, purchases, and slot positions that may result from displaying their ad on the search result page. In order to do so, each program creates a Bids table as described in Section 2.2.1 each time there is a sponsored search auction. These programs have access to several variables pertinent to the current auction and to the advertiser, such as the keywords in the search query, the time of day, the advertiser's remaining budget, the current return on investment for the keywords that the advertiser is interested in, and so on. These variables are stored in tables, some of which are read-only shared between all advertisers (such as the time and location of the search) and some of which are private to each advertiser (such as information about the keywords that the advertiser is interested in). The programs can then be written using simple SQL updates without recursion and side-effects. SQL triggers

Table 2.4: Keywords Table

text	formula	maxbid	roi	bid	relevance
boot	$Click \wedge Slot_1$	5	2	4	0.8
shoe	$Click$	6	1	8	0.2

can be used to activate programs when an auction begins and to notify programs if they received a slot, click, or purchase. Programs can modify their private tables, although commonly used variables, such as amount spent, budget remaining, return on investment for various keywords, etc. can be automatically maintained for each program by the search provider. For example, the advertiser-specific variables related to keywords are stored in a Keyword table, as depicted in Table 2.4 that is private to each advertiser. Each tuple in the Keyword table corresponds to a bid for a keyword that the advertiser is interested. The attributes of the tuple contain, among other things, the formula for the bid, keyword's relevance score in the search query, the return on investment that this keyword has provided the advertiser, the maximum amount that the advertiser is willing to bid on a click by a user who searched for this keyword, and the amount of money that the advertiser is currently bidding for the keyword. The search provider updates the return on investment for a keyword each time a user searches for the keyword and then clicks on the advertiser's ad. The bidding program can be stored with its private tables to improve locality. Since bidding programs use private tables and read-only shared tables, they do not interact with each other when they are triggered by a new search query. Hence they can be distributed across several machines and run in parallel if necessary.

2.2.3 An Example: Equalizing ROI

We now give a concrete example of a dynamic bidding strategy that bids on multiple features. Our example combines the dynamic ROI equalizing heuristic mentioned in Section 2.1 with bidding on two features, clicks and the top slot; the advertiser is interested in receiving clicks for two keywords, "boot" and "shoe", but also wants to be perceived as the leading supplier of boots and so would be willing to pay extra to be shown in the top slot if the search query is highly relevant to boots. In order to control his spending, the advertiser has a target spending rate that he wishes to maintain. The ROI equalizing heuristic, as suggested in [16], tries to dynamically allocate spending across the different keywords and bids so as to maximize the advertiser's "bang for the buck". If the advertiser is underspending (i.e., his current spending rate is lower than his target spending rate), then the advertiser increases the bids on keywords that have been most profitable for him (i.e., those with the highest return on investment). If the advertiser is overspending (i.e., his current spending rate is higher than his target spending rate), then the advertiser decreases the bids on keywords that have been least profitable for him (i.e., those with the lowest return on investment). Return on investment of a bid is the total value gained from the keyword (e.g., number of clicks received in the top slot times the amount the advertiser values a click in the top slot) divided by the total amount spent on it.

Figure 2.1 shows the program for this strategy. Line 1 creates a trigger that waits for a new query to be inserted into the Query table, indicating that a new auction is taking place. If the advertiser notices that he has been underspending (line 3), he increases his tentative bids for all relevant keywords that have provided him with the highest ROI, taking care not to increase the bid past its

```
 1 CREATE TRIGGER bid AFTER INSERT ON Query
 2 {
 3   IF amtSpent / time < targetSpendRate THEN
 4     UPDATE Keywords
 5     SET bid = bid + 1
 6     WHERE roi =
 7       ( SELECT MAX( K.roi )
 8         FROM Keywords K )
 9       AND relevance > 0
10       AND bid < maxbid;
11   ELSEIF amtSpent / time < targetSpendRate
12   THEN
13     UPDATE Keywords
14     SET bid = bid - 1
15     WHERE roi =
16       ( SELECT MIN( K.roi )
17         FROM Keywords K )
18       AND relevance > 0
19       AND bid > 0;
20   ENDIF;
21
22   UPDATE Bids
23   SET value =
24     ( SELECT SUM( K.bid )
25       FROM Keywords K
26       WHERE K.relevance > 0.7
27         AND K.formula = Bids.formula );
28 }
```

Figure 2.1: Equalize ROI Strategy

maximum value (lines 4–10). Similarly, lines 13–19 decrease his bids for relevant keywords with the lowest ROI if he is overspending (line 11), taking care not to decrease his bid below zero. Next, he updates the values in the Bids table with the sum of his tentative bids for the corresponding formulas for all sufficiently relevant keywords, namely, those with a relevance score higher than 0.7 in the user-submitted search query (lines 22–27). For example, if the Keywords table is as depicted in Table 2.4 after running lines 1–20, then the output Bids table

Table 2.5: Bids Table for Example Program

formula	value
$Click \wedge Slot_1$	4
$Click$	0

will be as depicted in Table 2.5.

2.3 Winner Determination

Having empowered the advertisers with a language for expressing dynamic bidding strategies to bid on a rich set of features, we now seek efficient and scalable techniques for the search provider to perform winner determination.

All sponsored search auction mechanisms currently in use (see, for example, [3, 5, 30, 81]) first solve the winner-determination problem and assign slot positions according to the winning allocation, and then use some method of charging prices for the positions, such as charging each advertiser their social opportunity cost (this is known as *Vickrey pricing* [21, 39, 82]), or charging advertiser in the kth slot the amount bid by the next-highest bidder (this is known as *generalized second-pricing* [30]). Note that, with most pricing schemes, a provider's revenue is *not* the revenue that is computed in the winner-determination problem. Nevertheless, the first step in all these auctions is to do winner determination. Furthermore, given winner determination as a subroutine, the pricing schemes used in these auctions (i.e., Vickrey pricing, generalized second-pricing, etc.) can all be expressed as very simple computations. In our work, therefore, we focus on optimizing the winner-determination computation.

2.3.1 How Winner Determination Works

The *winner-determination problem* is to compute the allocation of slots to advertisers that results in the highest expected revenue for the search engine provider, under the assumption that advertisers actually pay what they bid. In keeping with Google and Yahoo policy, we restrict the slot allocations to those in which no advertiser gets assigned more than one slot. This prevents extremely wealthy advertisers from monopolizing all the available slots. We call this the *assignment restriction*; in Chapter 3, we return to this issue and lift the restriction.

In order to compute the expected revenue resulting from an allocation, we need the advertisers' bids on clicks, purchases, and slot positions as specified in their Bids tables. For now, we assume that we actually run all of the advertisers' bidding programs to get their resulting Bids tables. In Section 2.4, we give techniques that require us to run only a small subset of programs under certain conditions.

In order to compute the expected revenue resulting from an allocation, we also need the probabilities that the formulas in the Bids tables are true in the final outcome. We thus consider the set of all possible outcomes that describe which slot was allocated to which advertiser together with which advertisers received clicks and purchases. The probabilities of clicks and purchases depend on the search provider's allocation of slots to advertisers. For example, ads placed at the top are more likely to be noticed and clicked on than those placed in the middle of the page [65]. As a reasonable first-order approximation, we assume that the probability that a given advertiser gets a click depends only on the slot allocated to him, and that the probability that he gets a purchase depends only on whether he got a click and on the slot allocated to him. Furthermore, we assume that the search provider has (or can estimate, using data it has collected) these click and purchase probabilities for each advertiser and each slot allocation to that advertiser.

Note that a complete representation of the probabilities of all possible formulas for each advertiser is exponential in the number of features. Although this is not too large in our setting, the complete set of probabilities should be stored in a database separate from the run-time system, which itself should store only probabilities for the formulas mentioned in the bidding programs and Keyword tables, since these are the only probabilities that are used. Furthermore, the probabilities can be partitioned by advertiser and should be stored with the advertiser's bidding program and private tables to improve locality.

2.3.2 Complexity

Given the assumptions on slot allocations and distributions above, we look at the complexity of solving the winner-determination problem given bids in our language. Recall that a bidding program's output is an OR-bid represented by a Bids table whose rows contain bids of the form "Pay \$$d_1$ for E_1", ..., "Pay \$$d_m$ for E_m", where $E_1, \ldots, E_m$ are Boolean combinations of the $Slot_j$, $Click$, and $Purchase$ predicates. Recall that, in addition, we assume that, for any allocation, we have a distribution on outcomes, conditional on that allocation. Each formula E_i can be identified with an *event* on the set of possible outcomes, namely, the set of outcomes in which E_i is true. Toward proving that winner determination is tractable for bids in our language, we introduce the following definition.

Definition 2.3.1 (m-dependent event) *An event is m-*dependent *if there are at* *most m advertisers such that probability of the event given any allocation depends only on the placement of those m advertisers.*

That is, an event is m-dependent if it is independent of the slots assigned to all but m advertisers. For example, the event that a given advertiser gets a click is 1-dependent, since we assumed that the probability of an advertiser getting a click depends only on the slot position of that advertiser. Similarly, the event that a given advertiser is in either the top slot or the bottom slot is 1-dependent, since it depends only on the slot assigned to that advertiser. However, given two advertisers, the event that one gets the top position and the second gets the bottom is 2-dependent, since it depends on the slots assigned to both those advertisers.

We assume that the representation of each m-dependent event includes the labels of the m advertisers on whose slot assignment the event depends. The following theorem says that winner determination is tractable for 1-dependent events.

Theorem 2.3.2 *For OR-bids that contain formulas corresponding to 1-dependent events, the winner-determination problem is in polynomial time.*

Proof Consider any bid of \$$d$ on E, where E is a formula corresponding to a 1-dependent event that depends on the slot assigned to only one advertiser, say i. If advertisers pay what they bid, then, in all outcomes, this bid contributes exactly the same amount to the revenue as the OR-bid of \$$d$ on $E \wedge Slot_1^i$, \$$d$ on $E \wedge Slot_2^i$, $\dots$, \$$d$ on $E \wedge Slot_k^i$, and \$$d$ on $E \wedge (\wedge_j \neg Slot_j^i)$, where $Slot_j^i$ is a propositional variable that is true iff advertiser i gets slot j. This is because

$Slot^i_1, \ldots, Slot^i_k$ correspond to mutually exclusive events, given that the allocations are restricted to at most one slot per advertiser. We can thus fill out a table of advertisers versus slots where the entry for the ith advertiser and the jth slot is the sum of the total expected revenue from bids on formulas of form $E \wedge Slot^i_j$, assuming that advertisers pay what they bid. If we interpret this table as the edge-weight matrix of a bipartite graph between advertisers and slots, then the winner-determination problem is the problem of finding a maximum-weight bipartite matching for this graph, which can be done in polynomial time [53]. $\square$

It follows that winner determination for bids represented by a Bids table can be solved in polynomial time, since our assumptions in Section 2.3.1 guarantee that any Boolean combination of predicates in the set $\{Click, Purchase, Slot_1, \ldots, Slot_k\}$ corresponds to a 1-dependent event.

A natural question to ask is whether we can extend our tractability results to a language that allows advertisers to bid on formulas corresponding to m-dependent events, for $m \geq 2$. The next result says that winner determination is *APX-hard* if we allow bids to be placed on formulas corresponding to 2-dependent events, such as the event that one advertiser is displayed above another. APX is the class of NP optimization problems that have polynomial-time constant-factor approximation algorithms [46].

Theorem 2.3.3 *For OR-bids that contain formulas corresponding to* 2*-dependent events, the winner-determination problem is APX-hard.*

Proof We reduce the winner-determination problem to the maximum-weight feedback arc set problem by using bids on formulas corresponding to 2-

dependent events to encode the edges in a given weighted directed graphs on advertisers. Consider any weighted directed graph on n advertisers. Let $w_{i,i'}$ be the weight of the edge from advertiser i to advertiser i'. Let $Slot_j^i$ be the propositional variable that is true iff advertiser i gets assigned slot j. For two advertisers i and i', let $E_{i>i'}$ be shorthand for $\vee_j(Slot_j^i \wedge ((\vee_{j'>j} Slot_{j'}^{i'}) \vee (\wedge_{j'} \neg Slot_{j'}^{i'}))$, which corresponds to the event that advertiser i gets a slot and is placed above advertiser i' who may or may not get a slot. Then $E_{i>i'}$ corresponds to a 2-dependent event, since it depends on the slots assigned to advertisers i and i'. Let each advertiser i place the following bids: for each $i' \neq i$, bid $w_{i,i'}$ on $E_{i>i'}$. Then, assuming advertisers pay what they bid, revenue of $w_{i,i'}$ will be generated if and only if advertiser i is placed above advertiser i'. Then winner determination is equivalent to the problem of finding the maximum-weight feedback arc set over all size-k subgraphs, which is APX-hard in n and k [46].

In our reduction, each formula in an advertiser's OR-bid corresponds to a 2-dependent event. This does not preclude the set of all advertisers that these events depend on from being large (e.g., an advertiser's OR-bid could contain $n-1$ formulas of the form $E_{i>i'}$ for each $i' \neq i$). However, the reduction above gives us an NP-hardness result even if the OR-bids are restricted so that all the events corresponding to formulas depend on at most two other advertisers *in total* (e.g., if each advertiser's OR-bid contains at most 2 formulas of the form $E_{i>i'}$ with $i' \neq i$). This is because the maximum-weight feedback arc set problem is NP-hard even if the input graphs have degree 3 [64, 48]. $\square$

Figure 2.2: Separable and Non-separable Click Probabilities

	$Slot_1$	$Slot_2$
Nike	0.8	0.4
Adidas	0.6	0.3

	$Slot_1$	$Slot_2$
Nike	0.7	0.4
Adidas	0.6	0.3

2.3.3 Existing Allocation Algorithms

The allocation algorithms used by Google and Yahoo, as well as those studied in the literature [5, 3, 30, 81], deal with the issue of scalability by assuming that the probability of a click resulting from assigning a slot to an advertiser is *separable*, that is, it can be written as the product of an advertiser-specific factor and a slot-specific factor. To illustrate this notion of separability, we provide examples of separable and non-separable click probabilities in Figure 2.2. The left matrix in Figure 2.2 is separable because the entries in the matrix can be split into the product of advertiser-specific factors (namely, 4 for Nike and 3 for Adidas) and slot specific-factors (namely, 0.2 for slot 1, and 0.1 for slot 2).

When the click probabilities are separable, it is easy to see that winner determination can be performed by assigning the advertisers with jth highest advertiser-specific factor to the slot with the jth highest slot-specific factor.[1] This can be done in time $O(n \log k)$.

Note that the assumption of separability implicitly assumes that the event that an advertiser gets a click is 1-dependent. Indeed, it assumes the event that an advertiser gets a click depends on only that advertiser's slot assignment. But separability requires much more 1-dependence; it requires that the ratio of the expected number of clicks on one advertiser in a slot and the expected number

[1]A more detailed description of this algorithm is given in Chapter 4.

of clicks on another advertiser in the same slot is the same for all slots. Thus, separability is a very brittle property; if we change one number in a separable matrix, it will no longer be separable (as seen, for example, in Figure 2.2).

Not only is separability a much stronger requirement than 1-dependence, but the techniques for fast winner determination that use this assumption do not suffice to deal with our bidding language. In particular, they cannot deal with the situations described in Section 2.1 where one advertiser wants to be displayed in the top slot or not displayed at all, while another wants to be displayed in either the top or bottom slots but not in the middle slots. (Bids representing these preferences can be easily expressed in our language.)

2.3.4 Maximum-Weight Bipartite Matching

We proved Theorem 2.3.2 by showing that winner determination in this case is equivalent to maximum-weight bipartite matching between advertisers and slots, where the edge-weight between an advertiser and a slot is the expected revenue obtained by assigning that slot to that advertiser. The fastest known (non-parallel) algorithm to solve this is the Hungarian algorithm, invented by Kuhn [53] (also known as the Kuhn-Munkres algorithm after being revised by Munkres [62]); it finds the best matching in time $O(nk(n + k))$ where n is the number of advertisers and k is the number of slots. Since this is quadratic in n, this will not scale well. We want to deal with situations where n can be quite large (possibly in tens to hundreds of thousands). To make the problem scalable, we need it to be *linear* in n, the number of advertisers. There are parallel algorithms for maximum-weight matching [36], but these require prohibitively

Table 2.6: Expected Revenue Matrix

	$Slot_1$	$Slot_2$
Nike	**9**	5
Adidas	**8**	**7**
Reebok	7	**6**
Sketchers	7	4

large numbers (typically $\Omega(n^2)$) of processing units in order to achieve linear running time.

2.3.5 Our Algorithm

We now give a scalable winner-determination algorithm that takes advantage of the fact that k, the number of slots, is quite small (say less than 20) compared to n, the number of advertisers. Indeed, n is growing rapidly every year while k remains the same. We can modify the Hungarian algorithm to get a $O(nk \log k + k^5)$ algorithm by considering only those advertisers whose values are in the top k highest for some slot. That is, for each slot, we consider the k advertisers who would produce the top k expected revenue if placed in that slot. We take the union of these advertisers over all the k slots, and consider the bipartite subgraph containing only these advertisers along with all the k slots. We then solve maximum-weight bipartite matching problem for this reduced bipartite graph.

As an example, consider the expected revenue matrix as depicted in Table 2.6. There are two slot positions available and four advertisers. The top two

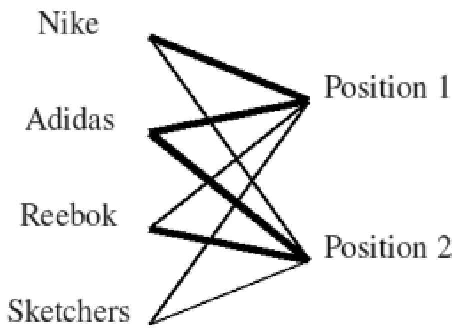

Figure 2.3: Bipartite Graph

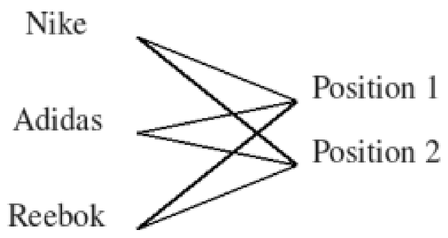

Figure 2.4: Reduced Graph

expected revenues for the first slot come from Nike and Adidas, while the top two expected revenues for the second slot come from Adidas and Reebok. The corresponding edges in the original bipartite graph between advertisers and slots have been depicted in bold in Figure 2.3. This bipartite graph is then reduced to contain only those advertisers with an adjacent bold edge as depicted in Figure 2.4.

We observe that the maximum matching for the original problem must occur for this smaller problem, since if a maximum matching in the original problem assigned a slot to an advertiser who was not among the top k bidders for that slot, we can simply reassign that slot to one of these top k bidders who is not assigned any slot. Note that, since there are only $k-1$ other slots, at least one advertiser in the top k is guaranteed to remain unassigned.

Finding the relevant advertisers takes time $O(nk \log k)$ because, for each slot, we can find the top k bidders for that slot in time $O(k + n \log k)$ by maintaining a priority heap of size at most k. There are at most k^2 such advertisers, since in the worst case we will have a distinct set of k advertisers for each of the k slots. Hence, running the Hungarian algorithm on the reduced graph takes time $O(k^5)$ for a total running time of $O(nk \log k + k^5)$ for our algorithm.

Parallelization. Our technique lends itself very well to parallelization. Note that in our setting there is typically already a high amount of parallelized infrastructure present, since the bids are collected from advertisers in a distributed way. We construct k networks of computers each in the form of a binary tree of height $O(\log n)$ with n leaves. We can compute a maximum matching in time $O(k \log n + k^5)$ as follows. For each slot j, we consider the jth binary tree network, which will ultimately compute the top k bidders for that slot at the root:

1. The ith leaf node in the jth network starts out with the expected revenue from assigning slot j to advertiser i.
2. Each internal node gathers the top k bidders (along with their corresponding bids) from its two children, and combines them into a single list of top k bidders. This takes time $O(k)$ for each of the $O(\log n)$ levels of the tree, since each level of the tree works in parallel.
3. The root nodes in each of the j-networks take the union of their lists of bidders and compute the maximum-weight matching of these bidders with the k slots using the Hungarian algorithm. This takes time $O(k^5)$, since there are k slots and at most k^2 bidders considered.

Note that we can mix sequential processing with parallel processing by running more than one program sequentially on each machine, computing the top k bids, and then aggregating using a tree network as before. If we have a binary tree network with p nodes, then the total running time becomes $O(\frac{n}{p} k \log k + k \log p + k^5)$.

Finally the $O(k^5)$ part of the algorithm (i.e., the part resulting from running the Hungarian algorithm on the reduced bipartite graph) can be reduced to

$O(k^2)$ using a parallel algorithm, such as that of Fayyazi et al. [36]. The number of parallel processing units required is $O(k^5)$, which is independent of n.

2.4 Top-k Program Evaluation

In Section 2.3.5, we showed that, in order to solve winner determination quickly, we need to find the advertisers with the top k expected revenue for each slot. We can easily do this if we have the bids output by the advertisers' programs. However, getting these bids for a given search query requires, in the worst case, running each advertiser's program for that query. This itself can be quite expensive. An obvious step toward alleviating this problem is for search providers to use their proprietary keyword matching algorithms to prune away advertisers who are not interested in the search keywords for the current auction. However, this is not enough if the search query contains a popular keyword, such as "music" or "book", where the set of interested advertisers can still be large. In this section, we show that we can further reduce the amount of work by taking advantage of knowledge of the structure of the advertiser's programs. To simplify exposition, we assume that advertisers' programs output bids on only $\mathit{Click} \wedge \mathit{Slot}_1, \ldots, \mathit{Click} \wedge \mathit{Slot}_k$. It is easy to incorporate bids on other formulas, since both Click and $\mathit{Purchase}$ are assumed to be 1-dependent events.

2.4.1 Threshold Algorithm

We start by considering a situation where the only difference between the programs used by different advertisers is in the values of certain advertiser-specific parameters. More precisely, for each slot $j \in [k]$, suppose that each advertiser's bids depends on a set of (numeric) parameters X_j in a monotonic way. That is, there is a monotonic function $f_j : X_j \to \mathbb{R}^+$ that takes as input a value for each parameter in X_j and outputs a bid for a click in slot j. We allow some subset of the parameters Y_j to be advertiser-specific: these can vary from advertiser to advertiser (e.g., the amount that they value a particular keyword, the amount of budget remaining, etc.).

Suppose further that these parameters Y_j are updated only by programs that win the auction. In Section 2.4.3, we consider the case where all programs can update their state; nonetheless, restricting updates to winning programs is not unreasonable, since most useful advertiser-specific quantities (such as number of auctions won, amount spent so far, return on investment for a given keyword, etc.) change only when the advertiser wins an auction.

The rest of the parameters $Z_j = X_j \setminus Y_j$ can be thought of as public global parameters, and are the same for all advertisers (e.g., the keyword scores associated with the user's search query, the time and date, the number of times the keywords in search query have appeared today). As an example, consider the situation where advertisers all use the same general strategy of starting each day by bidding low and then gradually increasing their bids as the end of the day approaches. However, they each start with a different amount and might increase their bids at different rates. The starting amounts and the rate of increase would be advertiser-specific parameters in Y_j, and the time of day would be a global parameter in Z_j.

For each advertiser i and each slot j, let the edge weight between advertiser i and slot j be $w_{i,j} \times f_j(y_{i,j}, z_j)$, where $w_{i,j}$ is the probability of advertiser i getting a click in slot j, $y_{i,j} \in Y_j$ are the values of the advertiser-specific parameters, and $z_j \in Z_j$ are the values of the global parameters. We previously showed that we can solve the maximum-weight matching in time $O(nk \log k + k^5)$. Under the assumptions above, we can further reduce the $O(nk \log k)$ portion that finds the top k bidders for each slot as follows. For a given slot j, we also store a list of bidders sorted by $w_{i,j}$ and incrementally maintain $|Y_j|$ lists of bidders, each sorted by one of the parameters in Y_j. We can then run the *threshold algorithm* [34] with these lists as input to find the top k advertisers with the highest values of $w_{i,j} \times f_j(y_{i,j}, z_j)$. Note that we do not need to maintain lists for the parameters in Z_j since all advertisers have the same value for these parameters. Since f_j is monotonic, the threshold algorithm is *instance optimal* for the class of algorithms that find the advertisers with the top k values of $f_j(x_{i,j})$ without making "wild guesses" (i.e., the algorithms must not access an advertiser until that advertiser is encountered via a sequential scan of one of the lists). Instance optimality means that, for any input, the threshold algorithm finds the top k values within a constant factor of the time it takes the fastest algorithm that avoids wild guess on that input. Given these top k advertisers for each slot, we take $O(k^5)$ further time to compute the winners as described in Section 2.3.5. To maintain the sorted lists, once the k winners have been computed, we update their Y_j parameters and accordingly update their positions in the sorted lists, which takes $O(|Y_j| k \log n)$ time.

2.4.2 Bid Range Tracking

In the previous section, we examined programs with the same basic structure. We now examine programs with differing structures. Again, we assume that the programs are such that only programs that win the auction update their private state. Our goal in this section is to reduce work by performing the top k computation over a restricted set of advertisers in most rounds, while only periodically having to evaluate the bidding programs of all advertisers.

We use the following observation. Let $U_{i,j}$ and $L_{i,j}$ be upper and lower bounds on output bid of advertiser i's program for slot j. Then we only need to consider those programs i for which $U_{i,j}$ is at least as high as the kth highest $L_{i,j}$ value.

The algorithm maintains a partition the advertisers' programs into $\mathrm{DefiniteLosers}$ and $\mathrm{PossibleWinners}$, keeping track of upper bounds on the expected revenues for all advertisers and of lower bounds on the expected revenue for just those advertisers in $\mathrm{PossibleWinners}$. At every step, the partition is updated by running the following procedure:

1. Compute the expected revenue only for programs in $\mathrm{PossibleWinners}$.
2. Compute the top k expect revenue amongst the exact bids from programs in $\mathrm{PossibleWinners}$.
3. Update the upper and lower bounds for the expected revenue for the programs in $\mathrm{PossibleWinners}$.
4. Update the upper bounds for advertisers in $\mathrm{DefiniteLosers}$.
5. Move an advertiser from $\mathrm{DefiniteLosers}$ to $\mathrm{PossibleWinners}$ if her upper bound is at least as high as the kth-highest lower bound from $\mathrm{PossibleWinners}$.
6. Move an advertiser from $\mathrm{PossibleWinners}$ to $\mathrm{DefiniteLosers}$ if her upper bound is less than the kth-highest lower bound from $\mathrm{PossibleWinners}$.

We can store the programs in DefiniteLosers in a max-heap, prioritized by the upper bound for the expected revenue from that program, in order to perform steps 5 and 6 efficiently. If PossibleWinners is much smaller than DefiniteLosers, the major savings come from finding an upper bound that does not change if an advertiser loses the auction. This eliminates the need to perform step 4.

As an example, we take the inputs to the program to be: geographic relevance, temporal relevance, user profile relevance, keyword relevance, number of clicks received so far today, amount paid for clicks so far today, and daily budget remaining. We make the reasonable assumption that the output bid is increasing with respect to the first four parameters. Notice that the last three parameters do not change for losing bidders. (They also happen to be monotonic with respect to time for a given day, but we do not take advantage of that here.) Thus we can use, as an upper bound, the value of the bidding function supplied with the maximum possible values for the relevance parameters and current values for the last three parameters. For a lower bound, we can supply the minimum values for the relevance parameters that result in a non-zero bid.

2.4.3 Logical Updates

We now consider the case where all program update their state, not just the winners. In certain situations, it is possible to reduce the amount of work done in this case as well. Consider a situation where many programs update their state using an operation that maintains their relative bid ordering. For example, suppose that many bidders are using the ROI heuristic described in Section 2.2.3, each with possibly different target spending rates and maximum bids. As long as certain conditions hold (namely, the bid is above zero and the spending rate is above the target spending rate), the heuristic will decrement its bid for a given keyword. Thus, if we can maintain a *decrement list*—that is, a list of programs, sorted by their bid, that are currently decrementing their bid for a given keyword—we can avoid explicitly decrementing each program's bid, by instead performing a single *logical* decrement in constant time. That is, the decrement list is associated with a single *adjustment variable*, initially zero. A program's bid is then the sum of the adjustment variable and the program's stored bid. So, in order to decrement the bids of all programs in the list, we simply decrement the adjustment variable. The sorted order is maintained because all programs in the list adjust their bids by the same amount.

Of course, the ROI heuristic eventually stops decrementing the bid and starts to increment it (if the spending rate drops below the target) or keep it constant (if the bid is zero) instead. At this point we must move the program to an *increment list* or a *constant list* as appropriate (similar to a decrement list, except that the adjustment variable respectively increments or remains constant). At first glance, this would seem to involve checking checking the conditions for each program at every auction. However, we observe that such conditions can often be reduced to waiting for a shared monotonic variable (such as time, or the number of times a given keyword has occurred) to reach a *critical value*. For example, in the ROI heuristic, the spending rates of losing programs decreases with time, since their amount spent remains constant. We can thus compute the next "critical" time that a program would have to stop decrementing and start incrementing assuming it continued to lose. Similarly, we can compute the number of auctions for given keyword necessary before its bid would be decremented to zero and it would have to remain constant at zero. We maintain

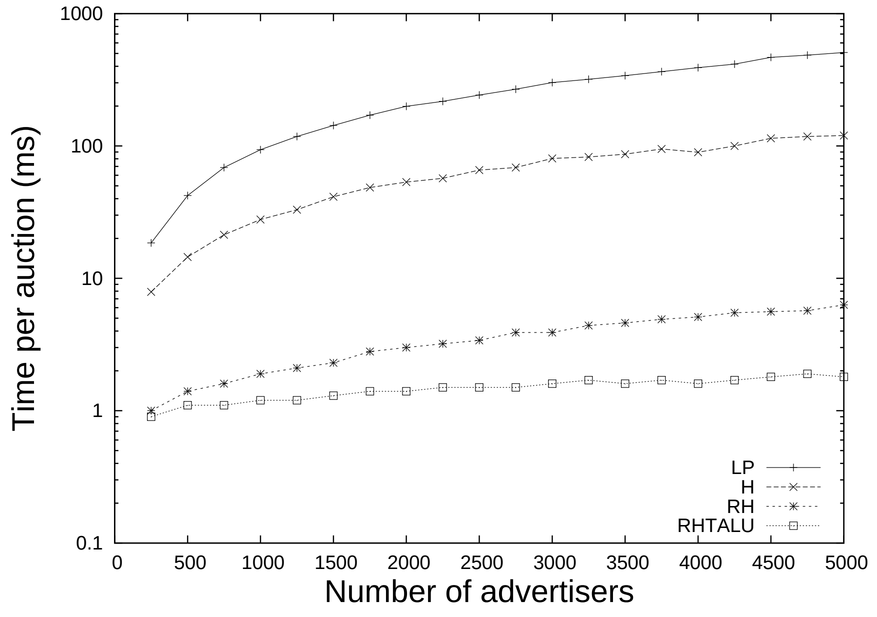

Figure 2.5: Winner Determination Performance

a list of *triggers* for the relevant shared monotonic variables, sorted by critical value, that when activated move a bidding program to the appropriate increment, decrement, or constant list, and insert the appropriate new triggers. This way, we do work only for programs that win an auction and for triggers whose critical values have been reached.

2.5 Experiments

To evaluate our fast winner-determination algorithm, we compare the performance of four methods for solving the winner-determination problem. The first method (LP) solves the linear program formulation of the winner-determination problem. We can prove that this linear program is guaranteed to have an inte-

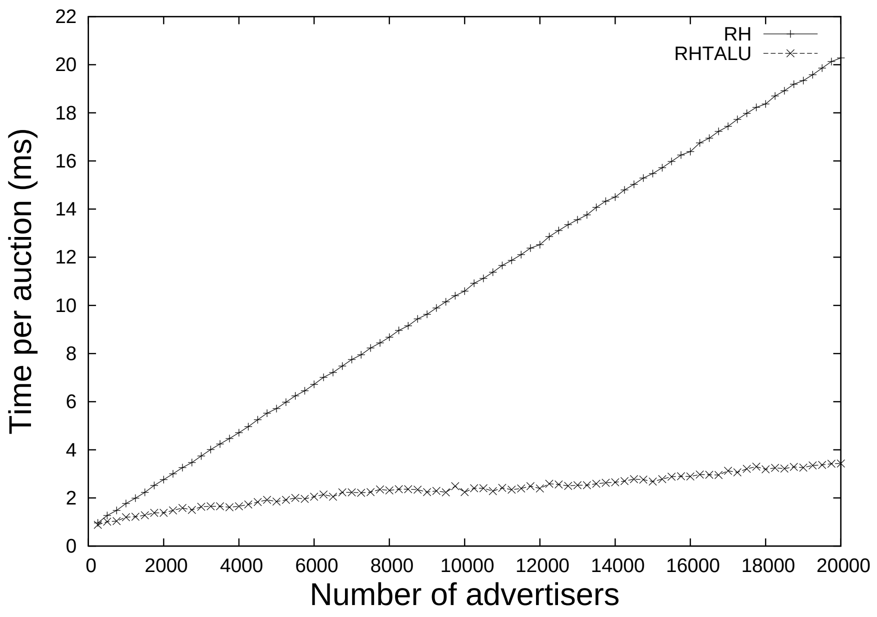

Figure 2.6: Reducing Program Evaluation

ger optimum using a theorem of Chvátal [20], by showing that the rows of the constraint matrix represent the maximal cliques of a perfect graph. The second method (H) uses the Hungarian algorithm in a straightforward way to compute the maximum-weight bipartite matching in the bipartite graph with advertisers on the left and slots on the right, where the weight of an edge from an advertiser to a slot is the expected revenue from assigning that slot to that advertiser. The third method (RH) is our winner-determination technique from Section 2.3.5, which first reduces the bipartite graph. The fourth method (RHTALU) augments RH with the techniques for reducing program evaluation from Section 2.4 using the threshold algorithm together with logical updates with triggers.

We used 15 slots in all cases. For simplicity, search queries were generated at a constant rate, each containing one keywords chosen uniformly at random out of 10 keywords. That chosen keyword was given a relevance score of 1 for that query, while other keywords had a relevance score of 0. All bidders used the ROI heuristic described in Section 2.2.2. For each keyword, the bidders' value for a click was generated uniformly at random between 0 and 50 (subject to each bidder having at least one non-zero click value). The target spending rates were chosen uniformly at random between 1 and the bidder's maximum value over all keywords. The interval $[0.1, 0.9]$ was partitioned into 15 disjoint intervals, with the $(j + 1)$-highest interval associated with slot j. The probability of a given advertiser getting a click in a given slot was generated uniformly at random within that slot's interval. We used a slight generalization of generalized second-pricing to charge the advertisers who received clicks.

The entire auction system, including the ROI heuristic, was implemented in C++. We used the GNU Linear Programming Kit to solve the linear program via the simplex method.[2] We ran the experiments on an AMD Athlon 64 3800+ processor with 1GB of RAM.

Figure 2.5 shows, for each of the four methods, the average time taken per auction (over 100 auctions) as we increase the number of bidders. We observed roughly an order of magnitude improvement of the Hungarian method over naive linear programming solution, and further order of magnitude improvement using our reduced bipartite graph technique. Figure 2.6 compares the performance of methods RH and RHTALU in more detail. It plots the average time taken per auction (over 1000 auctions) as we increase the number of bidders. We observe that our techniques for reducing program evaluation from Section 2.4 give a significant further improvement in performance.

[2]We found that the library's interior point method was much slower than the simplex method for our workloads.

2.6 Conclusions

In this chapter, we highlight the need for more expressive bidding in sponsored search auctions. To address this, we propose a framework that empowers advertisers with an expressive bidding language, and we provide efficient, scalable, and parallelizable techniques for performing winner determination given bids expressed in our language. In the next chapter, we build upon the material presented here, and discuss several interesting extensions to our techniques, as well as applications to other advertising scenarios.

CHAPTER 3

EXTENSIONS AND APPLICATIONS TO OTHER ADVERTISING AUCTIONS

In this chapter, we present some interesting extensions of the techniques we developed in the previous chapter. In Section 3.4, we discuss applications of these techniques to other kinds of advertising auctions.

3.1 Beyond 1-dependence

So far, our results have assumed that the probability that an advertiser receives a click or a purchase depends only on the slot to which that advertiser was assigned. However, it is easy to think of situations where this assumption might not be true. For example, if the slot assigned to an advertiser for a small company is just below a very large and popular competitor, then it is likely that the competitor will receive a substantial portion of user clicks that might otherwise have gone to the smaller advertiser had the competitor not been present.

Thus, the probability of receiving a click (or a purchase) would depend on who else displays an ad and in what position. In the worst case, the probability would depend on the entire slot assignment. The representation of such a general probability distribution would be quite large ($O(kn^k)$); it is not clear that we can determine winners much better than with the brute force algorithm that considers each of the possible $\binom{n}{k}k!$ assignments.

Moreover, advertisers could value two assignments differently even if both assignments may give the advertiser the same slot. For example, consider two assignments, both of which assign an advertiser slot 2. However, in the first assignment, slot 1 is given to a very famous company, while in the second assignment, slot 1 is given to a relatively unknown company. Then the advertiser in slot 2 would naturally prefer the second assignment to the first, since the famous company poses a serious threat to the advertiser in terms of diverting away clicks. Representing such general valuations would also require large space ($O(kn^{k-1})$) in general.

We now consider two extensions to our existing framework, both of which allow events that are not 1-dependent but, at the same time, do not require us to store such large distributions and valuations.

3.1.1 Heavyweights and Lightweights

For a given search auction, suppose that the advertisers are classified into either *heavyweights* (famous advertisers) or *lightweights* (relatively unknown advertisers). One way for the search provider to decide which advertisers are heavyweights is to select those advertisers with the most clicks so far.

We now allow the probability that a given advertiser gets a click (or a purchase) to depend not only on his slot position, but also on which slots have heavyweight advertisers and which slots have lightweight advertisers. We also allow advertisers to place bids that depend on which slots get heavyweights and which slots get lightweights, in addition to placing bids on click, purchases, and slot positions as before. Thus, an advertiser might bid 3 cents if he gets slot 2 and if there is a lightweight advertiser in slot 1. Advertisers could even place more complex bids, such as bidding on having no heavyweights within 3 slot positions above or below his slot in addition to having no more than 2 heavyweights appear anywhere else. The representation of the probability distributions and valuations now become $O(k2^{k-1})$, and does not depend on n.

In order to solve the winner-determination problem, we must find an assignment of slots to advertisers to maximize expected revenue (assuming advertisers pay what they bid) given these new valuations and distributions. Suppose that we knew exactly which slots get heavyweight advertisers in such a revenue maximizing assignment. We call these slots *heavyweight slots*, and call the remaining slots *lightweight slots*. Then we can solve the winner-determination problem by simply solving two disjoint maximum-weight bipartite matching problems: one matching the heavyweight advertisers to the heavyweight slots, and the other matching the lightweight advertisers to lightweight slots. And if we do this for each possible way to choose heavyweight slots, we can find the assignment that maximizes expected revenue over all possible assignments. Moreover, the maximum-weight bipartite matching problems for different choices of heavyweight slots can be solved independently and in parallel. Therefore, since there are 2^k ways to choose heavyweight slots, we can solve winner determination in time $O(2^k(n \log k + k^5))$ in series, or in time $O(n \log k + k^5)$ in parallel using 2^k processing units. Note that the number of parallel processing units is independent of the number of advertisers n. Therefore, this approach becomes practical when k is small, say 10 or less, which seems reasonable in practice.

3.1.2 Types

Often the search keyword might interest advertisers who are not competing with each other for sales. For example, a keyword like “games” might interest both sports stores and a video game stores. A large video game advertiser placed would not drain clicks away from a sports store advertisement placed just below it as much as it would drain clicks from another video game advertiser. We could therefore extend our model to incorporate the notion of *types*. Each advertiser has a type (e.g., ‘video game store’, ‘sports store’, etc.), and if advertisers of the same type are displayed, they affect each other’s click-through and purchase rates, whereas advertisers of different types do not affect each others click-through and purchase rates.

We now allow the probability that a given advertiser gets a click (or a purchase) to depend on his slot position as well as on the set of slots that have advertisers who are of the same type. We also allow advertisers to place bids on the set of slots that are assigned to advertisers of their own type, in addition to placing bids on click, purchases, and slot positions as before. Thus, an advertiser might bid 3 cents if he gets slot 2 and if there is a competitor in slots 1 or 3. Typically, an advertiser would bid less for an assignment with many other advertisers of his type, since these are the advertisers that give him the most competition. In general, the storage requirement for each advertiser’s valuations and click and purchase probability distributions is $O(k2^{k-1})$, similar to the heavyweight-lightweight model.

For winner determination, we generalize the technique employed in computing winners for the heavyweight-lightweight model. That is, for each possible way to assign types to slots, we run a separate winner-determination computation, giving us a running time of $O(g^k(n \log k + k^5))$ in series, or in time $O(n \log k + k^5)$ in parallel using g^k processing units, where g is the number of types of advertisers bidding in the auction.

3.2 Beyond the Assignment Restriction

In Chapter 2, we restricted the types of slot allocations to those in which each advertiser got no more than one slot. In this section, we examine what happens when we remove this restriction. If we allow advertisers to receive multiple slots, then the winner-determination problems looks rather easy: we simply assign each slot to the advertiser with the highest expected revenue in that slot.

Unfortunately, such an approach has an important shortcoming: it ignores the fact that consecutive slots in an array of slots are often *complementary goods*. That is, advertisers might value winning *both* of two adjacent slots more than the sum of the values of winning each slot alone. The reason for this is that adjacent slots give advertisers the advantage of being able to combine the two slots to create a larger ad, perhaps even using a larger font size, thus making their advertisement stand out more. In such case, the value of winning multiple adjacent slots in not additive with respect to the value of winning the individual slots by themselves. Furthermore, the click-through rate of such an ad in the larger combined slot is no longer a simple combination of independent click-through rates from the two adjacent slots. We thus extend our model to allow for valuations and click and purchase probability distributions on ads that span *blocks* of adjacent slots.

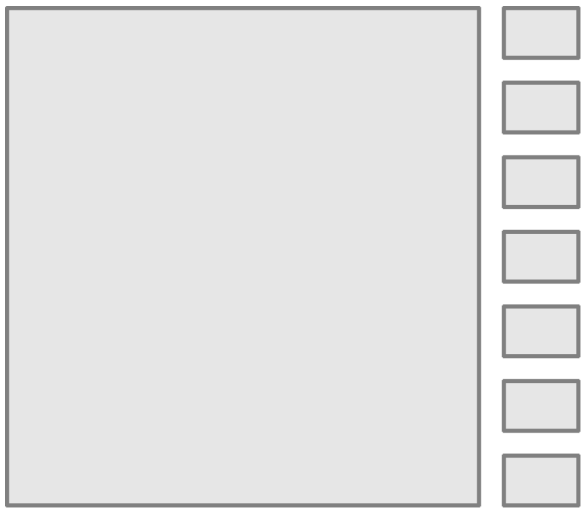

Figure 3.1: List Layout

3.2.1 List Layout

For the moment, we restrict our attention to one-dimensional arrays of slots, as depicted in Figure 3.1. We call such a layout of advertisement slots a *list layout*. List layouts are currently used by most of the major search engines such as Google and Yahoo. In this setting, the blocks are *slot intervals*, comprising of a set of one or more adjacent slots in the list. Let m be the number of such slot interval blocks. Then $m \in \theta(k^2)$, where k is the number of slots. We allow advertisers to place OR-bids on these slot intervals instead of on individual slots. Accordingly, we add predicates of the form $Slot_{x-y}$ for each $1 \leq x \leq y \leq k$ that the advertiser can use in their OR-bid table, in addition to the predicates described in the previous chapter. The predicate $Slot_{x-y}$ indicates that the advertiser gets all the slots between and including slots x and y. An example OR-bid table is depicted in Table 3.1, where the advertiser is willing to pay \$2 for slot 1 alone, \$1 for slot 2 alone, but \$4 if he gets both slots 1 and 2.

We maintain separate click and purchase probabilities for each advertiser in

Table 3.1: Interval Bids Table

formula	value
$Slot_1$	2
$Slot_2$	1
$Slot_{1-2}$	4

each slot interval. Therefore, the space requirements for storing an advertiser's bids and click- and purchase-probabilities increase from $O(k)$ to $O(k^2)$.

Winner Determination

We can solve the winner-determination problem as follows. With each slot interval, we associate the advertiser who would provide the highest expected revenue if displayed in that interval. This takes time $O(nm)$, where m is the number of blocks or slot intervals. Now construct a weighted undirected graph where each node corresponds to a slot interval, and the weight of the node is the expected revenue associated with the advertiser associated with that slot as computed above. Two nodes have an edge between them iff the intervals corresponding to the nodes intersect. By definition, this graph is an *interval graph*, i.e., a graph whose nodes represent intervals, with an edge between two nodes whose corresponding intervals overlap. If we had a maximum-weight independent set of this graph, then we could assign the slot intervals corresponding to nodes in this independent set to the highest advertisers associated with these intervals. Such an allocation would be feasible, since the independent set corresponds to a set of mutually disjoint intervals, and hence no slot would be assigned to more than one advertiser. Furthermore, such an allocation would be a solution to the winner-determination problem, since a maximum-weight independent set corresponds to an allocation with maximum expected revenue.

Thus, winner determination boils down to finding the maximum-weight independent set of an interval graph containing $O(k^2)$ nodes. Finding a maximum-weight independent set of the intersection graph is equivalent to finding a maximum-weight clique of the complement of the intersection graph. This is the graph whose nodes are blocks and whose edges connect nodes corresponding to disjoint blocks. We call this the *non-intersection graph*. Suppose we could impose directions on the edges of the non-intersection graph such that they satisfy transitivity. Note that for a list layout, we can do this by directing the edge between two disjoint intervals from the lower interval to the higher interval, assuming, without loss of generality, that the list is oriented vertically. Then we can solve winner determination in time $O(m^3)$, where $m \in \theta(k^2)$ is the number of blocks, using the following algorithm proposed by Even et al. [32]. We proceed inductively, by considering each node in an order corresponding to any linearization of the topological ordering. For each such node, we compute the maximum-weight clique in the subgraph induced by all nodes considered up to this point and containing this node. We can find such a clique by finding the maximum-weight clique from the set of cliques obtained by unioning the node under consideration with each of the cliques associated with the preceding nodes that have edges into the node under current consideration. Note that transitivity guarantees that the union of the node and a clique containing a preceding node is also a clique. Once we have encountered all nodes, we can find a clique of overall maximum weight in the non-intersection graph by considering the clique associated with each node and picking the one with the highest weight. Furthermore, if we store the slot intervals sorted by their endpoints, then we can find a maximum-weight independent set in time $O(m) = O(k^2)$ [37]; hence, winner determination takes time $O(nk^2)$.

3.2.2 Other Layouts

We now turn our attention to layouts other than lists. For example, instead of displaying a vertical list of ads on the right-hand side of the page, the search provider might display ads all around the border of the page, as depicted in Figure 3.2. Alternatively, the search provider might display ads as a vertical list with horizontal rows of ads protruding out of the list in-between various sections of the page, as depicted in Figure 3.4. With layouts such as these, the winner determination algorithm described above no longer works, since it is specific to linear layouts.

Consider the graph whose nodes are slots and whose edges connect nodes corresponding to adjacent slots. We call this the *layout graph* of the slots. A *block* of slots is any set of slots corresponding to a connected subgraph of the layout graph. Let m be the number of such blocks. Following the approach from the previous section, we construct the *intersection graph* of the blocks that the advertisers can bid for. This is the graph whose nodes represent blocks and whose edges connect nodes corresponding to blocks that intersect. As in the previous section, once we have associated each block with the advertiser who produces the highest expected revenue in that block, the solution to winner determination is given by finding a maximum-weight independent set of the intersection graph and then assigning each block in this independent set to its associated advertiser.

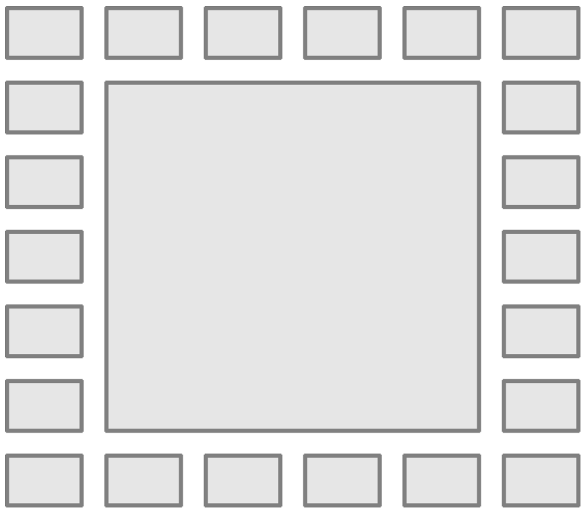
Figure 3.2: Ring Layout

Ring Layout

We say that a layout is a *ring layout* if the layout graph is a simple cycle, as depicted in Figure 3.2. In this case, the number of blocks, m is $\theta(k^2)$, and the intersection graph of blocks is a circular arc graph (i.e., one which is the intersection graph of arcs of a circle), for which finding the maximum-weight independent set is known to be solvable in time $O(m^2)$ [40, 74]. Thus, winner determination for a ring layout can be solved in time $O(nk^4)$.

Grid Layout

We say that a layout is a $k_1 \times k_2$ grid layout if the layout graph is a $k_1 \times k_2$ grid, as depicted in Figure 3.3. These present a problem because the number of blocks is exponential. Even if we restrict ourselves to rectangular blocks, for which there are $\theta((k_1 k_2)^2)$ blocks, the problem of finding a maximum-weight independent set is NP-hard and is in fact inapproximable to within a $o(\log k)$

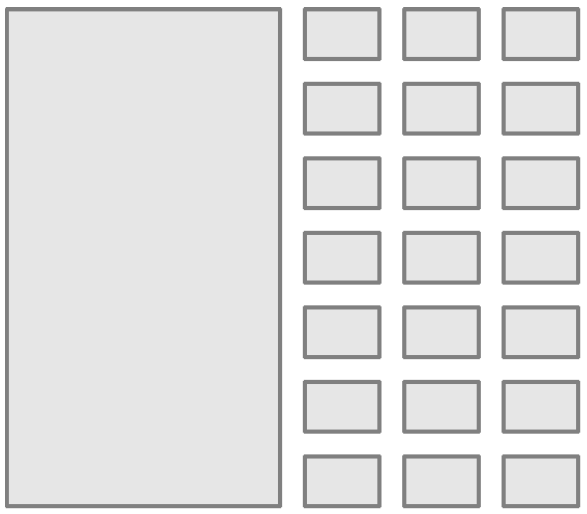

Figure 3.3: Grid Layout

factor of optimal by any polynomial-time algorithm unless $P = NP$ [19]. This means that selling 2-dimensional blocks of screen real-estate is fundamentally harder than selling blocks in a linear list. Nevertheless, if there a small number of blocks (e.g., if at least one dimension of the grid is small), then we can use an $O(2^m)$-time brute force solution, where m is the number of blocks, resulting in an $O(nm + 2^m)$-time winner determination algorithm.

If we restrict the assignment of blocks to ones where the blocks form an axis-aligned recursive subdivision of the grid, then winner determination can be achieved in polynomial time. Moreover, the restriction to subdivisions leads to a more comprehensible display of advertisements than if the ads tiled the grid in an arbitrary manner. We call an independent set of a grid's intersection graph a *subdivision set* if the blocks corresponding to the nodes in the independent set form an axis-aligned subdivision of the grid. To solve winner determination, we must find a maximum-weight subdivision set for the intersection graph. We can do so with a dynamic programming algorithm that computes maximum-weight subdivision set for all rectangular sub-grids in increasing or-

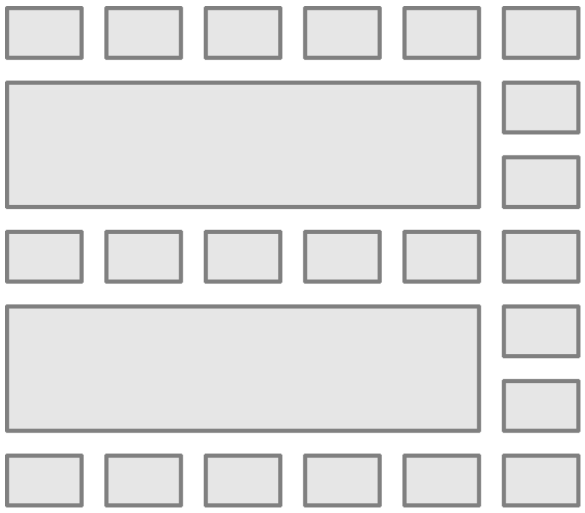

Figure 3.4: Tree Layout

der of area. To find the maximum-weight subdivision set of a given sub-grid, we try all $O(k_1 + k_2)$ ways to split the sub-grid into two pieces, and combine the subdivision sets of the two smaller sub-grids formed by the split. Since there are $O({k_1}^2{k_2}^2)$ rectangular sub-grids of the original grid, this algorithm runs in time $O((k_1k_2)^2(k_1 + k_2))$.

Tree Layout

The last type of layout we consider a *tree layout* (i.e., one where the layout graph is a tree, as in Figure 3.4). Here again, we run into the problem of having exponentially many blocks. For example, in a star tree, there are $\Omega(2^{k-1})$-many blocks, since every subset of leaves defines a connected subgraph of slots when connected through the central node. Therefore, we restrict our attention to trees with bounded degree d.

However, even in trees with bounded degree, there can still be exponentially many blocks. For example, in a full binary tree, just the number of connected subgraphs containing the root is $\Omega(2^{k/2})$. This is because each subgraph containing the root is the result of taking the original binary tree and subtracting off various subtrees, each of which corresponds to a disjoint interval of the $\theta(k/2)$ leaf nodes; and there are $\Omega(2^{k/2})$-many different ways to partition the leaf nodes into disjoint intervals. Therefore, in order to limit the number of blocks, we further restrict our attention to blocks containing at most b slots. Then the number of blocks will be $O(kd^b)$, since each slot can belong to no more than d^b blocks, given that the degree of the tree layout graph is bounded by d and the block size is bounded by b.

Now the intersection graph of the blocks has a natural tree decomposition given by associating each node in the tree layout graph with the set of blocks containing that node. It is easy to see that this tree decomposition yields a treewidth of $O(d^b)$, since each slot can belong to no more than d^b blocks. For a graph with $O(m)$ nodes and $O(d^b)$ treewidth, a maximum-weight independent set can be found in time $O(2^{d^b}m) = O(2^{d^b}kd^b)$ [15]. Thus, the total time for winner determination for a tree layout with degree bound d and block size bound b is $O(kd^b(n + 2^{d^b}))$.

3.3 Dealing with Budget Uncertainty

In most existing systems, advertisers can specify a daily budget, which represents the maximum amount of money the advertiser is willing to spend per day. The search provider is required to respect this constraint, and must therefore never charge an advertiser more than his daily budget on any given day. In order to perform winner determination correctly, we need to take this budget into consideration. What makes this tricky is that the amount of budget remaining is often uncertain. With the high rate of searches, an advertiser may well be interested in a new auction before he has to pay for his winnings from a previous auction. Since advertisers pay for clicks only after a user clicks on their ad, if the user from the first auction has not yet clicked on the advertiser's ad by the time the second auction occurs, there will be uncertainty about the amount of budget that the advertiser has remaining, since the first user may still click on the ad at some time in the future.

Suppose we were to ignore the budget issue during winner determination and simply not charge the advertiser if the user clicks after the advertiser's budget has been depleted. Consider an advertiser who is interested in a popular keyword, such as music, whose budget is almost exhausted. Until he receives enough clicks to completely exhaust his budget, we would allow him to bid his remaining budget on every music-related search query that occurs. He may win m auctions, but only have enough money in his budget to pay for $m' < m$ clicks. If he gets more than m' clicks, payment for the extra clicks would be forgiven. Thus, the advertiser would get more than his budget's worth of clicks. This constitutes lost revenue, since the slots could have been assigned to competing advertisers who had less chance of depleting their budgets. We now propose a principled solution to this problem by taking into account the outstanding ads that are awaiting clicks, and computing appropriately throttled bids for advertisers who are likely to go over budget.

3.3.1 Throttling Bids

To start with, consider an advertiser i for whom there are no outstanding ads awaiting clicks from users. Denote the i's remaining budget, i.e., his daily budget minus the amount he has paid to the search provider for clicks that have already occurred, as β_i. Suppose that in the current round, the advertiser takes part in m_i auctions, and that his current bid for a click is b_i. Rather than using b_i directly as his bid, we use a modified bid $\hat{b}_i$ instead. If the advertiser can afford to pay his stated bid of b_i for each of the m_i auctions, then we take $\hat{b}_i$ to be b_i; otherwise, we use the highest possible bid that the advertiser could still afford to pay for each auction. In other words, we let $\hat{b}_i = \min(b_i, \beta_i/m_i)$.

Now suppose that there are some, say l_i, outstanding ads of advertiser i that are awaiting clicks. For each outstanding ad j, suppose the price for a click on that ad was determined to be π_j and the probability of that ad getting clicked (given the time elapsed since the ad was displayed) is ctr_j. We make no assumptions about the value of ctr_j, but we point out that it is reasonable to model ctr_j as decreasing over time, and furthermore, that it reaches 0 after a specified time limit has passed; this will enable us to discard outstanding ads that have received no clicks in a long time. Let X_j be the random variable for the amount eventually paid for ad j. For any $l \in \{1, \ldots, l_i\}$, let $S_l = \sum_{j=1}^{l} X_j$. Thus, the amount of budget remaining once the debts for outstanding ads have been cleared is $\max(0, \beta_i - S_{l_i})$. We would like to take $\hat{b}_i$ to be the highest possible bid that the advertiser could still afford once his debts for outstanding ads have been cleared. That is,

$$\hat{b}_i = \begin{cases} b_i & \text{if } S_{l_i} < \beta_i - m_i b_i \\ 0 & \text{if } S_{l_i} \geq \beta_i \\ (\beta_i - S_{l_i})/m_i & \text{otherwise} \end{cases}$$

or, written another way, $\hat{b}_i = \min(b_i, \max(0, \beta_i - S_{l_i})/m_i)$. However, since the values of the X_js are uncertain because the ads are still awaiting clicks, we use the expected value at the time of winner determination. That is, we let $\hat{b}_i = \mathrm{E}(\min(b_i, \max(0, \beta_i - S_{l_i})/m_i))$.

3.3.2 Computing Bounds for Throttled Bids

Let ω_l denote $\sum_{j=1}^{l} \pi_j$, where π_j is the price for a click on the jth outstanding ad. Note that $S_{l_i} \leq \omega_{l_i}$, since each X_j is either π_j with probability ctr_j, or else is 0 with the remaining probability. Thus, if $\omega_{l_i} \leq \beta_i - m_i b_i$, then $\hat{b}_i = b_i$. Otherwise, if $\omega_{l_i} > \beta_i - m_i b_i$, we can compute $\hat{b}_i$ as follows. Note that $\mathrm{E}(\min(b_i, \max(0, \beta_i - S_{l_i})/m_i)) = \mathrm{E}(\min(m_i b_i, \beta_i - \min(\beta_i, S_{l_i})))/m_i$. Thus, in order to compute $\hat{b}_i$, we can compute the distribution of $\min(\beta_i, S_{l_i})$ and then take the expected value of $\min(m_i b_i, \beta_i - \min(\beta_i, S_{l_i}))$ over that distribution. This takes time $O(\min(2^{l_i}, \beta_i))$, assuming that β_i is written in the lowest denomination of currency. However, observe that during the winner-determination phase, we do not need the precise values of $\hat{b}_i$. We simply need the ability to compare $\hat{b}_i$ with $\hat{b}_{i'}$ for advertisers i and i' in order to find the top k advertisers. Of course, once winner determination is over, we will need the precise values of $\hat{b}_i$ for the winning advertisers in order to compute the prices for clicks. But there are only k winning advertisers at this point, so the amount of computation is a lot less than computing the precise $\hat{b}_i$ values for all n advertisers.

Now, in order to compare the $\hat{b}_i$ and $\hat{b}_{i'}$, we use Hoeffding bounds to compute successively tighter upper and lower bounds for $\hat{b}_i$ and $\hat{b}_{i'}$ until the upper bound is lower than the lower bound for the other at which point we can resolve the comparison test with certainty. In order to do this, notice that $\hat{b}_i$ can be rewritten as

$$b_i \Pr(S_{l_i} < \beta_i - m_i b_i) + \frac{1}{m_i} \operatorname{E}((\beta_i - S_{l_i}) 1_{\beta_i - m_i b_i \leq S_{l_i} < \beta_i})$$

We will denote upper and lower probability bounds as $\overline{\Pr}(\dots)$ and $\underline{\Pr}(\dots)$ respectively, and we denote upper and lower expectation bounds as $\overline{\operatorname{E}}(\dots)$ and $\underline{\operatorname{E}}(\dots)$ respectively. Let μ_l denote $\operatorname{E}(S_l) = \sum_{j=1}^{l} ctr_j \pi_j$ by linearity of expectation, and let σ_l denote $\sqrt{\operatorname{Var}(S_l)} = \sqrt{\sum_{j=1}^{l} ctr_j (1 - ctr_j) \pi_j^2}$. Using Hoeffding's inequality [41], which upper-bounds the probability that the sum of bounded independent random deviates from its expected value, we can derive the following bounds for $\Pr(S_l < x)$ for any $x > 0$,

$$\underline{\Pr}(S_l < x) = \begin{cases} 1 & \text{if } \omega_{l_i} \leq x \\ \max(0.5, 1 - \exp(-2(x - \mu_{l_i})^2 / \sum_{j=1}^{l_i} \pi_j^2)) & \text{if } \mu_{l_i} \leq x < \omega_{l_i} \\ 0 & \text{if } x < \mu_{l_i} \leq \omega_{l_i} \end{cases}$$

and

$$\overline{\Pr}(S_l < x) = \begin{cases} 1 & \text{if } \mu_{l_i} \leq x \\ \min(0.5, \exp(-2(\mu_{l_i} - x)^2 / \sum_{j=1}^{l_i} \pi_j^2)) & \text{if } x < \mu_{l_i} \leq \omega_{l_i} \end{cases}$$

Using these bounds, we can derive bounds for $\Pr(x \leq S_l < y)$ as $\underline{\Pr}(x \leq S_l < y) = \max(0, \min(1, \underline{\Pr}(S_l < y) - \overline{\Pr}(S_l < x)))$ and $\overline{\Pr}(x \leq S_l < y) = \max(0, \min(1, \overline{\Pr}(S_l < y) - \underline{\Pr}(S_l < x)))$. Now for $0 < x < y$, we can bound $\operatorname{E}(S_l 1_{x \leq S_l < y})$ from above and below by $x \Pr(x \leq S_l < y)$ and $y \Pr(x \leq S_l < y)$ respectively. Using the bounds that we have just derived, we can bound the value of $b_i \Pr(S_{l_i} < \beta_i - m_i b_i) + \frac{\beta_i}{m_i} \Pr(\beta_i - m_i b_i \leq S_{l_i} < \beta_i) + \frac{1}{m_i} \operatorname{E}(S_{l_i} 1_{\beta_i - m_i b_i \leq S_{l_i} < \beta_i})$ and hence that of $\hat{b}_i$.

If the bounds for $\hat{b}_i$ and $\hat{b}_{i'}$ as computed above are insufficient to decide the comparison, we can expand $\Pr(S_l < x)$ and $\mathrm{E}(S_l 1_{x \leq S_l < y})$ in terms of expressions involving S_{l-1}, π_l, and ctr_l to get tighter bounds. We do this repeatedly until the bounds are tight enough to decide the comparison. $\Pr(S_l < x)$ expands to

$$ctr_l \Pr(S_{l-1} < x - \pi_l) + (1 - ctr_l) \Pr(S_{l-1} < x)$$

and $\mathrm{E}(S_l 1_{x \leq S_l < y})$ expands to

$$\begin{aligned} & ctr_l \, \mathrm{E}(S_{l-1} 1_{x - \pi_l \leq S_{l-1} < y - \pi_l}) \\ + \quad & ctr_l \pi_l \Pr(x - \pi_l \leq S_{l-1} < y - \pi_l)) \\ + \quad & (1 - ctr_l) \, \mathrm{E}(x \leq S_{l-1} < y) \end{aligned}$$

We order the random variables X_j in increasing order of π_j. We expand out variables of high π_j values first, thus quickly eliminating their appearance in the Hoeffding bounds which as can be seen from the equations above leads to tighter bounds. Note that, in the worst case, the running time for getting a precise value for $\hat{b}_i$ is still $O(\max(2^{l_i}, \beta_i))$, but our technique allows us to terminate early once the bounds are tight enough for the purpose of comparison. Furthermore, we can cache the bounds for comparison with other $\hat{b}_{i'}$s and for computing the precise computation of $\hat{b}_i$ should advertiser i be one of the top k advertisers.

3.3.3 Related Work

Related to our work on uncertain budgets, Aggarwal and Hartline propose a related auction known as the knapsack auction, where bidders want to place items of varying sizes in a knapsack of a given capacity [4]. They suggest that this auction can be used to run a single auction to sell advertisement slots for the entire day where each advertiser's budget runs out after receiving exactly one click. In contrast to our approach, their auction fixes the outcome ex ante at the start of the day. Re, Suciu, et al. propose a technique they term 'multisimulation' to find the top-k most probable tuples in the result of a query to probabilistic database [67]. They do this by running Monte-Carlo simulations for all tuples and scheduling the simulations so as to quickly eliminate unlikely contenders.

3.4 Application to Other Settings

Our algorithms are applicable to more than just web search. We now discuss two other kinds of advertising auctions that benefit from the techniques we have developed so far.

3.4.1 Massively Multiplayer Online Games

Beyond web search, another setting to which sponsored ad auctions can be applied is massively multiplayer online games (MMOGs). In-game advertising in these games could prove to be a highly effective advertising platform; recent studies that included eye tracking have shown that 75% of gamers engage with at least one ad per minute across most, but not all, game types, and 81% of gamers engage at least every other minute [38]. Advertising in the form of product placement and in-game billboards is already making its way into the current generation of games, and in-game ad spending could reach $1.8 billion by 2010 [72].

We now show how to adapt our sponsored search auction framework to the auction of in-game billboards to advertisers. The billboard ads shown to a player do not have to be generated statically. The game can select which ad to display dynamically (via an auction) as long as it does so before the billboard is rendered on the player's screen. There are a number of ways to determine *when* to run the ad auction. For example, it can be done just before drawing the first frame in which the billboard is visible. Or it can be done by having the level-designer manually place *trigger areas* on the game map that activate the routine for ad selection. Ideally, the trigger areas would be placed on the map so that the player would have to cross the trigger area before the billboard comes into view. Well-developed techniques such as binary space partitioning [78] can help to automate the process of identifying trigger areas. No matter which method is used to determine when to run the auction, there is a requirement of fast winner determination in order to keep the game running in real-time.

In order to see how bidding would work in the MMOG setting, we need to identify the factors that affect how much an advertiser values being displayed on a given billboard. We model four such factors: player profile, billboard prominence, ad exposure, and player engagement.

Player Profile

Since the advertisements do not affect the gameplay, it would be acceptable for two players looking at the same virtual billboard simultaneously to see different ads. Thus, the game could display different ads for different players even in a shared environment. This opens the door to targeted advertising. Advertisers can bid differently for different player profiles. Accurate statistics about players' in-game activities are already maintained by game servers in order to track players' progress. These statistic can be used to get a picture of the type of player. For example, Bartle [12] proposes four prototypes: explorers, killers, achievers, and socializers. A player is given scores, called *Bartle quotients*, in each of the four types. These Bartle quotients can be used by advertisers to distinguish target market segments. For example, advertisers selling fiction books might bid higher for their ad to be displayed to players who have high explorer quotients. Even more useful, MMOGs contain well-defined social networks such as guilds (large groups of players who share similar goals or virtual professions), parties (smaller groups of players who go on quests together), and personal contact lists (other players who are friends of a player and who often socialize with the player in the game). These social networks can be mined to predict whether or not a new player falls into a certain market segment based on whether or not his friends and fellow guild-members do. Furthermore, all in-game chat is logged and so can be mined for keywords that indicate potential interest in the products that the advertisers sell. Of course, the extent to which the social networks and chat transcripts can be used to build the player profile for advertisers is subject to the privacy policies of the game. However, one could well imagine ad-supported versions of the game that allow players to play for free provided they agree to sharing their in-game social data with advertisers.

Billboard Prominence

One of the most important sources of value of virtual billboard advertising to an advertiser comes from impressions. The location of the billboard within the virtual world can affect the amount of impact its ads can have. Billboards places at eye-level within the gameworld tend to have greater impact. Too many billboards cluttered together can reduce the amount of impression that an ad makes. Beyond spatial positioning of the billboard, gameplay-related distractions present at the location can also affect impressions. For example, if a billboard is placed in an area where there is a lot of intense and immersive gameplay (such as combat with a monster), then the player is not likely to pay much attention to the ad displayed on that billboard. Thus, we can assign each billboard a *prominence score*, based on the visibility of its spatial location and on the amount of distractions present at the location (e.g., other billboards, enemies, etc.). The prominence score can be calculated just before the auction for a billboard (or set of billboards) begins, based on the number of enemies near the billboard at the time the player enters a trigger area, and based on the visibility the billboard would have for a player approaching from the trigger area.

Ad Exposure

Even if an ad is placed on a prominent billboard with few distractions around it, the player may still not see the ad because he just happened to be facing the wrong way. In determining the amount of exposure an ad has to a player, games have a great advantage over web search. It is easy to accurately measure and record various properties that directly affect an ad's exposure. For example, one can measure how long the ad is in the player's field of vision, whether or not the player's view of the ad was obstructed by another object, what angle the ad was viewed from, whether the player was engaged in some other activity (e.g., cycling through his inventory) while the ad was in view, etc. We can combine these measurements into a single exposure score that is accumulated over the course of the game. Note that the exposure score is known only after the player has quit the game and is therefore uncertain at the time of the ad auction.

Player Engagement

Beyond measuring the exposure that virtual billboard ads provide, one may be tempted to implement a mechanism analogous to clicks in sponsored search auctions to take the player to the advertiser's homepage in a separate browser window. However, care must be taken so as to minimize the intrusiveness of such a mechanism on the gameplay. The mechanism should allow players to express interest in an ad, but should not entail a substantial distraction from the immediacy of gameplay. We can use the aiming/targeting system already built into these games for such a non-intrusive mechanism. The idea is to allow players to *bookmark* an ad by "shooting" at the ad. The number of shots fired indicate how much the ad interests the player. Upon quitting the game, the player is then presented with a splash screen containing the list of all the ads he bookmarked, sorted by the extent of his interest.

Adapting Our Auction Framework

In the setting of advertising on in-game billboards, the billboards are analogous to slots. The exposure and engagement scores are similar to clicks and purchases in that they are unknown at auction time, and therefore the game must maintain distributions of exposure and engagement scores for each billboard. These distributions can be based on historical data. An auction for a set of billboards is run when a player enters their trigger area.

As before, advertisers submit programs to bid on their behalf, and these pro-

Table 3.2: Bids Table for In-Game Advertising

formula	value
$\ldots$	$\ldots$
$Billboard_2 \wedge Exp_{(0.8,1]} \wedge Eng_{(0.6,1]}$	3
$\ldots$	$\ldots$

grams are given access to variables relevant to the player's profile (e.g., Bartle types, guild, etc.) and to the current prominence scores for the set of billboards. Now, instead of of bidding on slots, clicks, and purchases, these programs output bids on billboards as well as on intervals of exposure and engagement scores. For examples, an advertiser can bid 3 cents for the second-most prominent billboard if exposure ends up being greater than 0.8 and engagement ends up being greater than 0.6. This would be represented by a bids table as shown in Table 3.2.

The game then computes winners so as to maximize the expected revenue assuming that advertisers pay what they bid. We assume once again that the only billboard that affects the exposure and engagement scores for an advertiser is the billboard to which he is assigned. Then we can use our algorithm from Section 2.3 to solve winner determination efficiently. Moreover, our techniques for reducing program evaluation from Section 2.2.2 still apply in this setting.

3.4.2 Map Routes

To illustrate the use of our more advanced techniques, consider selling advertisements on online map-route searches. In a map-route search, users submit a start address and a destination address; then the search provider returns a page displaying driving directions from the start to the destination, along with an map displaying this route.

We can display ads on map-route search result pages by highlighting segments along the route such that when the user clicks on or hovers over a highlighted segment, an advertiser's ad will be displayed in a balloon similar to the balloons used to expand pushpin ads on local business searches in Google Maps. Highlighted segments that are adjoining each other can be distinguished by the use of different colors. Moreover, the advertisers' ads can also be displayed alongside the appropriate set of steps in the driving directions.

Analogous to specifying a set of search phrases in a regular sponsored search auction, advertisers now specify either a set of addresses and a radius around the addresses, which are then matched to the resulting route in order to determine the set of competing advertisers. Advertisers might also specify more complex conditions, such as being a certain number of miles away from the start or destination address. For example, a gas station could choose to advertise only on map routes that start about 300 miles away and pass by the gas station. In addition to specifying addresses, advertisers could also specify search phrases, such as "beach" or "museum", which could then be matched to addresses. In this way, a swimwear store in an inland town could advertise on map searches from that town to any location near a beach.

To sell segments on a map route, we first partition the route into a sequence of *unit segments*. These unit segments are analogous to slots in a list layout. Advertisers then bid on blocks (i.e., intervals) of unit segments, just as in Section 3.2.1.

In addition to allowing advertisers to bid on segments, we can also employ the techniques proposed in Section 3.1.2 to allow advertisers to set different bids based on the number of competing advertisers of the same type displayed on the map. For example, a swimwear store might bid much higher if it is the only swimwear store displayed on the route, since one swimsuit is usually enough for most people.

CHAPTER 4
SHARING WORK BETWEEN AUCTIONS

In this chapter, we look at the problem of sharing work between auctions. We start by examining how to use shared top-k aggregation when solving winner determination problems for multiple auctions. We provide an inapproximability result for this problem. We then turn our attention to sharing other types of aggregates that could appear in advertisers' bidding programs, and we extend our analysis to give a complete characterization of the complexity of shared aggregation for commutative aggregates in terms of the algebraic properties of the aggregation operator.

4.1 Shared Winner Determination

Given the high volume of searches performed each day, several search queries arrive nearly simultaneously at any given time. This presents an opportunity for sharing the work of winner determination among several sponsored search auctions. In order to identify the work that can be shared across auctions, we need to first describe how winner determination is solved for an individual auction.

4.1.1 Separability and Winner Determination

As mentioned in previous chapters, the probability that a user clicks on an advertiser's ad depends on, among other things, the content of the ad and the slot in which the advertisement is displayed. (For example, studies have shown that

Table 4.1: Separable Click-Through Rates

ctr_{ij}	slot 1	slot 2
advertiser A	0.36	0.24
advertiser B	0.33	0.22
advertiser C	0.39	0.26

ads are more likely to be clicked on if they are displayed in slots at the top of a vertical list of slots than if they were placed lower in the list [65].) For the moment, we make the separability assumption, described in Section 2.3.3, in accord with the earlier literature [5, 3, 30, 81]. In Chapter 2, we showed that winner determination could be done efficiently even without assuming separability. In fact, the techniques for sharing computation that we develop here can be applied to the winner determination algorithm proposed in Chapter 2. We return to this issue in Section 4.3.

Assuming separability, winner determination can be solved in time linear in the number of advertisers for any given auction as follows. Recall that the separability assumption states that the probability that a given ad receives a click when displayed in a given slot can be written as the product of two factors, one that depends only on the advertiser and the other that depends only on the slot position. This probability, denoted as ctr_{ij}, is known as the *click-through rate* of advertiser i in slot j. So the separability assumption says that $ctr_{ij} = c_i \times d_j$, where c_i is the advertiser-specific factor and d_j is the slot-specific factor. Table 4.1 shows an example of separable click-through rates that can be decomposed into advertiser-specific factors and slot-specific factors, as depicted in Table 4.2. Since each ctr_{ij} is separable as $c_i \times d_j$, winner determination

Table 4.2: Advertiser-Specific and Slot-Specific Factors

	A	B	C
c_i	1.2	1.1	1.3

	slot 1	slot 2
d_j	0.3	0.2

Table 4.3: Bids and Weighted Bids

	A	B	C
b_i	14	15	10
$b_i c_i$	16.8	16.5	13

is equivalent to finding one-to-one mapping α from slots to advertisers so as to maximize $\sum_{j \in [k]} b_{\alpha(j)} c_{\alpha(j)} d_j$. Then α dictates the allocation of slots: slot j is assigned to advertiser $\alpha(j)$. For example, consider the click-through rates defined by Table 4.1. Now suppose the advertisers bids are as depicted in Table 4.3. Then winner determination assigns slot 1 to advertiser A and slot 2 to advertiser B. Without loss of generality, assume that the slots are ordered such that slot j has the jth highest value of d_j. We can then solve the winner-determination problem by simply finding the advertisers with the top k values of $b_i c_i$ and setting $\alpha(j)$ to the advertiser with the j-highest value of $b_i c_i$. This requires a single scan over the $b_i c_i$s, keeping track of the top k advertisers.

Having described how winner determination works for a single auction, we turn our attention to sharing the work of winner determination between multiple auctions that occur in the same round. The choice of granularity of a round is left to the system designer. While choosing a coarser granularity will lead to higher sharing between auctions (since more searches will occur per round), and thus greater overall efficiency, it will also increase the latency (the time the user has to wait before obtaining her search results). Studies have shown that users tolerate median latencies of up to 2.2 seconds without much adverse perception of search quality, but median latencies of about 3.6 seconds or more are considered too long [70].

Under the separability assumption, winner determination amounts to finding the advertisers with the top k values of $b_i c_i$ where k is the number of slots and b_i is advertiser i's bid and c_i is the advertiser-specific factor of advertiser i's click-through rate. Thus, if the same set of advertisers take part in two auctions in the same round, then slots would be awarded in the same way in both. However, not every bidder takes part in every auction. An advertiser can specify a set of *bid phrases*. If the search query does not match one of the advertiser's bid phrases, then his ad is not entered into the auction.

In determining whether a query matches an advertiser's bid phrase, we assume that the two-stage method proposed in [66] is used, where the search query is first mapped into a lower-dimensional space of bid phrases and is then matched to the advertisers' bid phrases using exact match. Accordingly, if a bid phrase does indeed match some query, then we must find the advertisers with the top k values of $b_i c_i$ whose set of bid phrase contain the bid phrase. This is where we can share work between the different auctions.

For example, suppose that the search queries "hiking boots" and "high-heels" occur in the same round. There might be several general shoe stores that specify both queries as bid phrases. However there might be a few sports stores that specify "hiking boots" but not "high-heels", while a few high-end fashion accessory stores might only be interested in "high-heels" queries. Suppose there are 200 general shoe stores, 40 sports stores, and 30 upscale fashion stores. Finding the top k advertisers for each of the two phrases separately requires us to scan through 240 and 230 advertisers respectively. However, if we find the top k advertisers among the general shoe stores, the top k among sports stores, and the top k among the fashion stores (which requires looking at 200, 30, and 40 advertisers), we can then merge the first and second top k lists to find the top k advertisers interested in "hiking boots", and the first and third top-k lists to find the top k advertisers interested in "high-heels". Merging in this way allows us to scan 40% fewer advertisers.

This suggests the use of merging of two top-k lists as a primitive aggregation operation that we employ to build shared plans that successively aggregate the $b_i c_i$ values of all the advertisers so as to find the aggregates corresponding to the sets of advertiser interested in each bid phrase while minimizing the number of aggregate operations performed. Thus, the plan we build will be a DAG where each leaf node represents an advertiser, and each internal node has in-degree 2 and represents a top-k aggregation operator that aggregates the top k advertisers from the two upstream nodes.

One further issue that complicates sharing is that not all bid phrases occur in a given round. Thus a single shared plan may not be optimal in all rounds. Unfortunately, coming up with a new plan on the fly at every round based is not practical given the latency requirement of winner determination. Instead, we try to find a single plan offline that works well 'on average'. To formalize this, we assume that the event that a bid phrase occurs in a round is an independent Bernoulli trial whose probability is known. We call the probability that bid phrase q occurs its *search rate* and denote it as sr_q. We then try to find the plan involving pairwise top-k aggregation that computes the aggregate for each bid phrase, and minimizes the expected number of nodes *materialized* per round. A node is materialized in a given round if it is used to compute the result for a bid phrase that occurs in that round. In other words, a node is materialized if there is a path in the plan's DAG from that node to some node corresponding to a bid phrase query node. Therefore the probability of node v being materialized is $1 - \prod_{q:v \rightsquigarrow v_q}(1 - sr_q)$ where $v \rightsquigarrow q$ represents the statement that node v is used in the computation of the aggregate query corresponding to bid phrase q in the shared plan. Without loss of generality, we normalize the cost of the aggregation operator to 1. Then, by linearity of expectation, the total expected cost of a plan is

$$\sum_v \left(1 - \prod_{q:v \rightsquigarrow q} (1 - sr_q)\right).$$

4.1.2 Shared Top-k Aggregation

In this section, we examine the problem at the core of sharing winner determination: optimizing shared top-k aggregation plans. To this end, we develop a framework for shared aggregation using an abstract aggregation operator specified by a set of algebraic properties that the operator satisfies. We show that finding an optimal shared plan for our abstraction of the top-k aggregation operator is not only NP-hard, but is in fact inapproximable. The construction used in the proof motivates our heuristic for finding a good shared plan in the next section.

We start out by defining our notion of an abstract aggregation operator and its associated aggregate queries. An abstract aggregate operator is simply a binary function $\oplus$: $Z \times Z \to Z$, for some set of values Z (e.g., $\mathbb{Z}$, $\mathbb{N}$), such that the cost of evaluating $\oplus$ is constant. Given the abstract operator $\oplus$, aggregation queries are represented by *$\oplus$-expressions* which are obtained by starting out with a set of variables X and closing off under the binary $\oplus$ operator. An example of an aggregation query is $(x \oplus y) \oplus z$, where x, y, z are variables that take values in Z. In our setting, each variable represents the bid of some advertiser, and the values of the variables change rapidly since advertisers are constantly updating their bids using external search engine optimizers [51] or automated bidding programs [58] in order to achieve complex advertising goals such as staying in a given slot during specific hours of the day, staying a certain number of slots above a competitor, dividing one's budget across a set of keywords so as to maximize the return-on-investment, etc. [17, 51, 63]. We therefore have to evaluate our aggregate queries at each round since the variables are constantly taking on different values.

Without using information about the algebraic properties of $\oplus$, we can only share work between queries in a rather limited manner by reusing the results of sub-expressions used to compute the queries. For example, we can share work between $x \oplus y$ and $(x \oplus y) \oplus z$ by re-using the value of $x \oplus y$ (which was computed for the first query) during the computation of the second. But if we take advantage of the various algebraic properties that $\oplus$ satisfies, we can increase the amount of shared computation. For example, if $\oplus$ is commutative then we can share work between the queries $x \oplus y$ and $(y \oplus x) \oplus z$ by aggregating the value of z with the value of the first query in order to compute the value of the second.

Let I_q be the set of advertisers interested in bid phrase q. Then an $\oplus$-expression representing the aggregate query for bid phrase q is $\oplus_{i \in I_q} \ b_i$ (we assume left-associativity, allowing us to omit parentheses) where b_i is the variable containing advertiser i's bid. Throughout this subsection, we assume that all bid phrases occur in every round with probability 1 (i.e., $sr_q = 1$ for each bid phrase q). The hardness results presented here therefore extend to the case when the sr_qs are arbitrary. Sharing winner determination then amounts to finding a shared top-k aggregation plan that produces, for each phrase q, the top-k aggregate of the bids of advertisers listed in X_q. Recall that the top-k aggregation operator is the binary function that takes in two k-lists (i.e., lists of size at most k) and outputs a k-list of the top k elements of the union of the two input lists. Notice that this operator is clearly associative, commutative, and idempotent (i.e., aggregating a list L with itself returns L). It also has an identity element, namely, the empty list which, when aggregated with any k-list, returns that list. We therefore abstract the top-k aggregator using an abstract aggregator $\oplus$ satisfying the following algebraic properties.

A1. $\forall a\,.\,\forall b\,.\,\forall c\,.\,a \oplus (b \oplus c) = (a \oplus b) \oplus c$ (associativity)

A2. $\exists e\,.\,\forall a\,.\,a \oplus e = e \oplus a = a$ (identity)

A3. $\forall a\,.\,a \oplus a = a$ (idempotence)

A4. $\forall a\,.\,\forall b\,.\,a \oplus b = b \oplus a$ (commutativity)

For convenience of notation, let $\mathcal{A} = \{A1, A2, A3, A4\}$. $\mathcal{A}$ characterizes the algebraic notion of a semilattice with identity element, and so our results in this subsection apply to any meet or join operator, such as $\min$, $\max$, and Bloom-filter unions and intersections [14], etc.

Two $\oplus$-expressions e and e' are *$\mathcal{A}$-equivalent* iff $e = e'$ is provable in first-order logic plus $\mathcal{A}$. Now we can formally define the notion of a shared plan. Given a set E consisting of $\oplus$-expressions over X, an *$\mathcal{A}$-plan for E* is a DAG satisfying the following properties:

1. each node is labeled with an $\oplus$-expression and has in-degree either 0 or 2,
2. each node with in-degree 0 is labeled with a variable $x \in X$,
3. each node with in-degree 2 is labeled with an $\oplus$-expression $e \oplus e'$,
4. each $e \in E$ is $\mathcal{A}$-equivalent to the label of some node.

The *total cost of an $\mathcal{A}$-plan* is the number of nodes with non-zero in-degree in the graph (i.e., those nodes representing top-k aggregation operators). Now we can formally state the shared aggregation problem as follows. *Given a set E of $\oplus$-expressions over X, find the min-cost plan for computing each $e \in E$.*

We assume, without loss of generality, that no two $\oplus$-expressions in E are $\mathcal{A}$-equivalent and also that no $\oplus$-expression in E is $\mathcal{A}$-equivalent to any variable $x \in X$ since we can identify such expressions upfront and remove such duplicates. We define the *base cost* of an $\mathcal{A}$-plan to be $|E|$. Since every expression E must be the label of a non-leaf node of a plan for E, every plan for E has cost at least $|E|$. What is interesting is the cost of the plan over and above $|E|$. We define the *extra cost* of an $\mathcal{A}$-plan to be the total cost of the plan minus $|E|$. The nodes contributing to extra cost are the partial results that are used to compute the final set of aggregates. Note that minimizing the extra cost is equivalent to minimizing the total cost of an $\mathcal{A}$-plan. Later on, when we discuss inapproximability, we will measure competitive ratio in terms of extra cost instead of total cost, since the base cost for all $\mathcal{A}$-plans for E is the same and is unavoidable.

First we state the following lemma, which is easy to prove.

Lemma 4.1.1 *Two $\oplus$-expressions over a set of variables X are $\mathcal{A}$-equivalent iff the set of variables appearing in the two expressions are equal. In particular, $e_1 \oplus e_2$ and e are $\mathcal{A}$-equivalent iff the set of variables appearing in e is equal to the union of the sets of variables appearing in e_1 and e_2.*

Next, we show that finding an optimal shared plan is NP-hard for our abstraction of the top-k aggregator.

Theorem 4.1.2 *Finding a min-cost $\mathcal{A}$-plan for E is NP-hard, where E is a finite set of $\oplus$-expressions over a finite set of variables X.*

Proof We reduce this to the *set-cover problem*, which is well-known to be NP-complete [48]. Recall that, in the set-cover problem, we are given a finite 'universal' set U, a finite collection $\mathcal{S}$ of subsets of U such that $\cup_{S \in \mathcal{S}} S = U$, and an integer k, and we must determine whether there is some $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \leq k$, and such that $\cup_{S \in \mathcal{S}'} S = U$.

Consider any instance of set cover. We can convert this into an instance of the problem of finding a minimum-cost $\mathcal{A}$-plan using the following construction. We create a variable for each element of the universal set. That is, we set $X = U$. For each $S \subseteq X$, we define a 'canonical' $\oplus$-expression e_S as follows. Let $<_X$ be an arbitrary strict ordering on the variables in X. If $S = \{x_1, \ldots, x_k\}$ is a nonempty set of variables in X, such that $x_1 <_X \cdots <_X x_k$, then e_S is the expression $x_1 \oplus \cdots \oplus x_k$ with implicit left-associativity. (Since $\oplus$ is associative by assumption, we do not need parentheses here.) e_S thus represents the query that aggregates the variables in S in ascending $<_X$-order.

Now let the set of $\oplus$-expressions $E = \{e_U\} \cup \{e_S : S \in \mathcal{S}\}$. That is, we have an

$\oplus$-expression corresponding to each set in $\mathcal{S}$ and one extra $\oplus$-expression corresponding to the universal set. Note that this construction can be done in polynomial time, assuming $<_X$ is represented as a linear list of variables in ascending order.

Now suppose that we have a polynomial-time algorithm for finding the min-cost $\mathcal{A}$-plan for E. Let G be the (DAG) plan returned by the algorithm. Let $\leq$ be the binary relation on nodes in G defined by $u < v$ iff G contains a directed (possibly zero-length) path from u to v. For each $e \in E$, let u_e be the node labeled with the $\oplus$-expression that is $\mathcal{A}$-equivalent to e. Note that checking whether two $\oplus$-expressions are $\mathcal{A}$-equivalent can be done in polynomial time by Lemma 4.1.1. Let $V = \{u : u \leq u_{e_U}\}$ and $W = \{u : \exists S \in \mathcal{S} \,.\, u \leq u_{e_S}\}$. That is, V is the set of all nodes that have a path to the node labeled e_U and W is the set of all nodes that have a path to a node labeled e_S for some $S \in \mathcal{S}$. So V induces an arborescence rooted at u_{e_U} that represents the plan's pairwise aggregation computation of e_U. Similarly, the DAG induced by V represents the plan's computation of the $\oplus$-expressions in $\{e_S : S \in \mathcal{S}\}$. Let Z be the set of nodes in $V \cap W$ that have an edge into $W \setminus V$. The nodes in Z are the ones with paths leading both to u_{e_U} and u_{e_S} for some $S \in \mathcal{S}$. For each node $z \in Z$, let S_z be the set in $\mathcal{S}$ such that $z = u_{e_{S_z}}$. Note that Z forms a cut of the arborescence induced by V, since V and W have the same leaf nodes (namely, the nodes labeled by the variables in X). Therefore, by Lemma 4.1.1, $\{S_z : z \in Z\}$ is a set cover of U, since e_U is formed by aggregating the nodes in Z since Z cuts the arborescence induced by V. Since the plan generated by the algorithm was minimal, this must be a minimal set cover of U, otherwise we could have replaced the nodes in $V \setminus W$ by aggregating the smaller set cover, which would have produced a plan with fewer nodes. $\square$

Finally, we extend the idea behind the construction in the previous proof to show that finding an optimal shared plan for our abstraction of the top-k aggregator is, in fact, hard to approximate to within less than a logarithmic factor of optimal.

Theorem 4.1.3 *There is no polynomial-time algorithm that finds a shared plan whose extra cost is within a* $\log n$ *factor of optimum unless* $P = NP$.

Proof We follow the same construction as the proof of Theorem 4.1.2, except that we close the query expressions off under sub-expressions before adding the universal set query. This ensures that the only extra nodes we add are for computing the universal set query, which as we showed in the proof of Theorem 4.1.2 corresponds to finding a minimal set cover. Then the theorem follows directly from the fact that minimal set cover is not approximable to within a $\log n$ factor of optimal [55]. $\square$

As we mentioned previously, these complexity results apply to the case where the queries are probabilistic as well. We also point out that presence or absence of the identity axiom A2 does not have any affect on our complexity of aggregation. This is mainly due to the fact that we are aggregating variables, not constant elements, and therefore we cannot exploit the properties of the identity element since the variables may or may not contain the identity element at any given round.

Greedy Algorithm

In the previous section we proved that finding an optimal shared plan for top-k aggregation between multiple auctions is inapproximable to within a $\log n$ factor even for the special case where all queries occur with certain probability. We now propose a heuristic for finding a shared aggregation plan for multiple probabilistic queries. Our approach consists of two stages: identifying fragments, and aggregating across fragments.

Identifying fragments. In the first stage, we group together all variables that occur in the same set of query expressions. We associate with each variable a bit string of length m, where the ith bit indicates whether or not the variable occurs in the i query expression. Then we group together all variables that are associated with the same bit string. These groups are equivalence classes of variables and are called *fragments* in [52]. Note that even though there are 2^m possible fragments, only $O(n)$ will be non-empty since there are n variables. We can safely aggregate elements within a fragment since no sharing occurs across fragments, since fragments are equivalence classes. This step itself provides some basic multiquery optimization since no fragment is computed twice. It is not hard to see that this step takes $O(mn \log n)$ time; the $\log n$ factor comes from having to index the fragments by bit string to identify groups of fragments. Alternatively, a hash table of bit strings could be used for grouping.

Aggregating across fragments. In the second stage, we use a greedy heuristic to complete the plan that was started out by aggregating together all the nodes within each fragments. We say that an $\mathcal{A}$-plan is *incomplete* if it does not compute all query expressions, i.e., if there is some query expression that is not equivalent to the label of any node in the plan. We can always complete an incomplete plan by finding a set cover of the missing query nodes from the collection of existing nodes. Note that we are associating nodes with the set of variables mentioned in the $\oplus$-expression labeling the node according to Theorem 4.1.1. Also note that we use the term 'set cover' to mean a cover whose union exactly equals the target set instead of just being a superset of the target set. This usage of the term is made without loss of generality with respect to earlier complexity results.

Suppose, for the moment, that all queries occur with probability 1 at each round. Then the optimal way to complete the plan without any further sharing would be to find a minimum set cover C of each of the missing queries, and to aggregate together all the nodes in C using an arbitrary binary tree, all of which use $|C| - 1$ nodes. Thus, the cost of completing the plan without sharing costs is $\sum_q (|C_q| - 1)$, where C_q is the size of the minimal set cover for query node q. This motivates our greedy heuristic, which works as follows.

At every step, we find two nodes that would aggregate together to form a new node that would lead to the highest decrease in $\sum_q |C_q|$ per unit extra cost of computing the new aggregate node. We call the decrease in $\sum_q |C_q|$ resulting from aggregating a pair of existing nodes the *coverage gain* of the pair. Note that the extra cost of creating a new aggregate node is 1 unless the aggregate is equivalent to a query expression, in which case the extra cost is 0 since the query node would have had to be computed anyway, and would therefore have counted toward the base cost. If there are multiple pairs of nodes that would cover some previously uncovered query, then we pick the pair with the highest coverage gain. The intuition is that the faster we cover all the query nodes, the faster the plan gets completed, and hence the fewer the extra nodes that are used.

Unfortunately, as the reader might have noticed, we run into a problem if we try to carry out the heuristic as stated above. In order to pick the pair of nodes with the highest coverage gain, we need to first calculate the minimum cover for each query node from the existing set of nodes. But as we saw in Section 4.1.2, minimum set cover is an NP-hard problem, and is, in fact, not approximable to within a $\log n + 1$ factor. Therefore, we cannot use the real set cover in measuring coverage gain; instead, we use the cover prescribed by the greedy covering algorithm, which works as follows. Until the target set is covered, repeatedly pick the feasible set that covers the maximum number of as-yet-uncovered elements. It is known that this greedy algorithm produces a cover within a $1 + \log n$ factor of optimal [45]. In fact, the greedy algorithm performs even better during the early decisions: the greedy algorithm achieves a competitive ratio logarithmic in $|S|$, where S is the largest cardinality set in the collection [45] (in our context, this is the size of the largest set associated with an existing node in the incomplete plan). We call the total size of the covers of all the query nodes as prescribed by the greedy covering algorithm the *greedy coverage*. Since we always decrease the greedy coverage at each step, we run for at most $\sum_q |X_q|$ steps, where X_q is the set of variables mentioned in query q. Each step requires finding a greedy cover for each query node. So the total running time is polynomial in $\sum_q |X_q|$.

So far in this subsection, we have assumed that all queries appears at each round. However, as we described earlier, in our application, each query occurs independently with some probability. We therefore extend the algorithm to deal with this probabilistic setting by replacing the notion of coverage gain with *ex-*

pected greedy coverage gain. The expected greedy coverage gain of a pair of nodes is the decrease in expected total greedy coverage of queries (i.e., $\sum_q sr_q|C_q|$, where sr_q is the probability of the aggregate query q occurring) resulting from aggregating that pair of nodes. Thus, the algorithm favors the covering and sharing of queries that are more probable over rare queries.

To summarize, our final algorithm works as follows:

1. First, group variables by the set of query expressions they appear in, and then aggregate the variables within each group.
2. Until the plan covers all query nodes, do the following:
    (a) For each pair of nodes in the incomplete plan, compute the expected greedy coverage.
    (b) If there exist some pairs of nodes that could be aggregated together to form a missing query node, then aggregate one such pair with the maximum expected greedy coverage.
    (c) Otherwise, pick any pair with maximum expected greedy coverage and aggregate them to form a new node.

The running time of the algorithm is polynomial in $\frac{\sum_q sr_q|X_q|}{\min_q sr_q}$, using an analysis similar to that for the deterministic queries case presented above, since the initial expected greedy coverage is $\sum_q sr_q|X_q|$, and this decreases by at least $\min_q sr_q$ at each step. We observe that our algorithm performs within a constant factor of any polynomial-time algorithm (unless $P = NP$) in the inapproximable case described in the proof of Theorem 4.1.3. Initially, our algorithm adds aggregates that compute all the $\oplus$-expressions of E' since these aggregate nodes have zero extra cost. Once these nodes have been created, our algorithm then tries to find a greedy covering for the query node labeled e_U, and therefore essentially runs greedy set cover, which is a $(1 + \log n)$-approximation to optimal.

4.1.3 Shared Sorting and Winner Determination

In the previous section, we provided techniques for sharing between auctions with where the advertiser-specific factor of the click-through rate was identical across all the bid phrases. In reality, it seems quite reasonable that the same advertiser in the same slot position might have different click-through rates for substantially different bid phrases. For example, a book store that mainly sells books but also dabbles in selling movies and music might have a higher advertiser-specific factor when the bid phrase is “books” than when the bid phrase is “DVDs”. If this is the case, we cannot directly share the top-k aggregation of the $b_i c_i$ values across bid phrases as we did in Section 4.1.1, since the value of c_i can be different for each phrase. Instead, we devote this section to examining how we can share work by exploiting the fact that the b_i values are shared across bid phrases.[1]

Threshold Algorithm

In order to share work between auctions, we use the well-known *threshold algorithm* [34] to find the top q advertisers for each bid phrase that occurs in a given round. In our context, the threshold algorithm works as follows. For a

[1]If, in addition, we allow the values of b_i to vary across bid phrases, then there is no opportunity for sharing work between phrases at all, since no data is shared between the phrases.

given bid phrase q, let c_i^q be the advertiser-specific factor of the click-through rate of advertiser i for bid phrase q. Let $j_1^q, j_2^q, \ldots$ be an ordering of advertisers who are interested in bid phrase q, ordered by descending values of c_i^q.[2] Let $i_1^q, i_2^q, \ldots$ be an ordering of advertisers who are interested in bid phrase q, ordered by descending values of b_i. The algorithm proceeds in stages. At each stage $s = 1, 2, \ldots$, the threshold algorithm incrementally maintains k indices in $\{i_{s'}^q : 1 \leq s' \leq s\} \cup \{j_{s'}^q : 1 \leq s' \leq s\}$ with the highest values of $b_i c_i^q$. It terminates early at the first stage s where all top k values are no less than the threshold defined by $b_{i_s^q} c_{j_s^q}$. It is well-known that the threshold algorithm is *instance optimal* for the class of algorithms that find the advertisers with the top k values of $b_i c_i^q$ without making "wild guesses" (i.e., the algorithms must not access an advertiser until that advertiser is encountered via a sequential scan of one of the lists). Instance optimality means that, for any input, the threshold algorithm finds the top k values within a constant factor of the time it takes the fastest algorithm that avoids wild guess on that input. Thus, we could solve the top-k problem for each bid phrase in an instance-optimal manner if we had a way of listing, on demand, the advertisers interested in phrase q, starting with the advertiser i with the highest b_i and proceeding in decreasing order of b_i values. This motivates the following problem, which we call *shared merge-sort*.

Shared Sorting

Consider a sponsored search auction that matches some bid phrase q, and let I_q be the set of advertisers who are interested in k. For ease of exposition, we assume that each $|I_q|$ is a power of two; the discussion generalizes to arbitrary

[2]We assume that the click-through rates are recalculated only occasionally and for the most part remain fixed. Therefore the ordering $j_1^q, j_2^q, \ldots$ can be treated as fixed and can be precomputed.

cardinalities in a straightforward way. The threshold algorithm as described above initially asks for the advertiser $i \in I_q$ with the highest value of b_i. It then asks for the advertiser from I_q with the next highest value of b_i, and then the next, and so on, until the threshold condition described above has been met. To supply the threshold algorithm with an advertiser at each stage, we construct a plan whose DAG is a balanced binary tree as used in a merge-sort of the set $\{b_i : i \in I_q\}$. Each leaf node is associated with a distinct b_i from this set. Rather than running the entire merge-sort upfront, we treat each non-leaf node as an on-demand operator that stores a left register and a right register, and sends the contents of the larger of the two registers upstream and then clears that register; if a register to be read from is empty, its value is pulled from its corresponding downstream node. This way, we don't do any extra work beyond the stage where the threshold condition is met. Each operator stores the sequence of values it has sent upstream. This will be used for caching results when operators are shared between multiple sort plans.

Now suppose that there is some other auction for another phrase q', and let $I_{q'}$ be the set of advertisers who are interested in q'. If $I_q \cap I_{q'} \neq \emptyset$, then we have already done some work in ordering advertisers who are interested in k. We would like to re-use some of this work when feeding advertisers to the threshold algorithm for phrase q'. Clearly, we can re-use the cached results of any operators below which all leaves correspond to advertiser in $I_q \cap I_{q'}$. This amounts to the problem of optimally sharing our on-demand merge operators in the merge-sort trees for multiple bid phrases. With each each merge operator v, we associate the set of advertisers I_v corresponding to the leaves below the operator. Then, according to the usual merge-sort tree restrictions, two operators u and v can be merged into a new merge operator w only if $I_u \cap I_v = \emptyset$ and

$|I_u| = |I_v|$. The total number of times an operator v is invoked in the worst case is $|I_v|$. This happens when the threshold condition is never met and the entire set I_v is sorted. Since we do not model the distribution of values that the b_is take, we conservatively use this *full-sort cost* when evaluating the cost of shared plans. Thus, the expected full-sort cost of a merge-sort operator in a shared plan is $|I_v| \left(1 - \prod_{q:v \rightsquigarrow q}(1 - sr_q)\right)$, where sr_q is the probability that bid phrase q appears in some auction, and $q \rightsquigarrow v$ denotes the property that operator v is used in bid phrase q's merge-sort tree in the shared plan. By linearity of expectation, the total expected full-sort cost of a shared merge-sort plan is

$$\sum_v |I_v| \left(1 - \prod_{q:v \rightsquigarrow q} (1 - sr_q)\right).$$

Greedy Algorithm

We propose the following simple bottom-up greedy heuristic for building a shared merge-sort plan that starts out with the leaf nodes, each corresponding to a distinct advertiser, and successively merges the two nodes that would lead to the largest savings in expected cost. When creating a new node w, we annotate it with the set of bid phrases Q_w whose merge-sort tree it contributes to. Initially, each leaf nodes v is annotated with $Q_v = \{q : v \in I_q\}$. At any point, we can merge nodes u and v into a new node w only if $Q_u \cap Q_v \neq \emptyset$, $I_u \cap I_v = \emptyset$, and $|I_u| = |I_v|$. We then set $Q_w = Q_u \cap Q_v$ and $I_w = I_u \cup I_v$. We pick the u and v such that the expected savings of merging them to create new node w is maximized. The expected savings from creating node w is given by

$$|I_w| * \sum_{i=1}^{n} \left[\left(\prod_{1 \le j < i} (1 - sr_{q_j}) \right) sr_{q_i} \left(\sum_{j=i+1}^{n} sr_{q_j} \right) \right]$$

where $Q_w = \{q_1, \ldots, q_n\}$. Note that $\sum_{i=1}^{n} \left[\left(\prod_{1 \leq j < i} (1 - sr_{q_j}) \right) sr_{q_i} \left(\sum_{j=i+1}^{n} sr_{q_j} \right) \right]$ is simply the expected number of queries in Q_w that occur beyond the first.

4.1.4 Non-Separable Click-Through Rates

Our work in Chapters 2 and 3 went beyond the traditional assumption of separable click-through rates [58]. Advertisers were allowed to bid on clicks, impressions, and purchases resulting from displaying their ad, and click-through and purchase rates were allowed to be non-separable. The techniques we proposed in Chapter 2 take advantage of the fact that the number of slots is usually very small in comparison with the number of bidders. Recall that a complete bipartite graph is constructed with advertisers on one side and slots on the other. The edge between an advertiser and a slot is weighted by the expected realized bid that would obtained by assigning that advertiser to that slot. The graph is then pruned to a much smaller graph by considering only the advertisers with the k highest edges incident to each slot, where k is the number of slots. Then the maximum weight bipartite matching is found between these $O(k^2)$ advertiser and k slots using the well-known Hungarian algorithm [53]. Our work in this chapter fits very well into that framework – we can use the shared top-k algorithms presented in Sections 4.1.2 and 4.1.3 to find the top k advertisers for each slot in the graph-pruning step required by our algorithm from Chapter 2.

4.2 Shared Aggregation

In this section, we look at the problem of sharing aggregation between multiple bidding programs. Recall that in Chapter 2, we proposed the idea of having each advertiser run a bidding program on their behalf. In order to make informed decisions about how to bid, it would be useful for these programs to be able to compute quantities such as average (or maximum) bid placed on a given set of bid phrases (e.g., those bid phrases containing the word 'music'), or the total number of users who have searched for one of a set of bid phrases. These quantities can be computed using sum, average, and count aggregates over bid phrases. Often multiple advertisers will want to perform similar aggregates over similar sets of bid phrases, giving us the opportunity to share such aggregation. We therefore consider aggregates other than the top-k aggregate that we considered in Section 4.1. However, rather than considering the shared aggregation problem for each particular aggregate, we take a more general approach and employ the abstract algebraic framework that we introduced earlier. In this section, we present the relationship between algebraic properties of the aggregation operation in question and the complexity of finding the optimal shared aggregation plan. To this end, we consider the following algebraic properties of binary aggregation operator $\oplus$.

A1. $\forall a \,.\, \forall b \,.\, \forall c \,.\, a \oplus (b \oplus c) = (a \oplus b) \oplus c$ (associativity)

A2. $\exists e \,.\, \forall a \,.\, a \oplus e = e \oplus a = a$ (identity)

A3. $\forall a \,.\, a \oplus a = a$ (idempotence)

A4. $\forall a \,.\, \forall b \,.\, a \oplus b = b \oplus a$ (commutativity)

A5. $\forall a \,.\, \forall b \,.\, \exists! c \,.\, \exists! d \,.\, a \oplus c = d \oplus a = b$ (divisibility)

These axioms can been used to characterize various algebraic structures of interest, including semigroups ($\{A1\}$), monoids ($\{A1, A2\}$), groups ($\{A1, A2, A5\}$), Abelian groups ($\{A1, A2, A4, A5\}$), bands ($\{A1, A3\}$), semilattices ($\{A1, A3, A4\}$), quasigroups ($\{A5\}$), and loops ($\{A2, A5\}$). In this work, we focus on those aggregates satisfying $A4$, since the most common and important aggregation operators that come up in our setting, and in database and stream settings in general are commutative. Such aggregates include sum, count, product, max, min, top-k, and Bloom-filter unions and intersections. Moreover, these aggregates can be combined with each other to compute other useful aggregates such as mean and variance.

We have already seen axioms A1, A2, A3, and A4 from Section 4.1. For aggregates satisfying the divisibility axiom A5, we can define a binary division operator $\ominus$ that such that $a \ominus b$ returns the unique c such that $a = b \oplus c$. (Note that $a \ominus c = b$ since we are restricting our attention to commutative operators and since divisors are unique by A5.) We assume that the cost of evaluating the $\ominus$ operator is constant, and is the same as the cost of evaluating the $\oplus$ operator. Then the $\ominus$ operator allows us to leverage divisibility to further reduce the cost of plans in certain cases. For example, consider an aggregation operator that is commutative, associative, and divisible (e.g., sum). Suppose we have to evaluate the following five queries: $u \oplus v \oplus w \oplus x$, $u \oplus v \oplus w \oplus y$, $u \oplus v \oplus x \oplus y$, $u \oplus w \oplus x \oplus y$, and $v \oplus w \oplus x \oplus y$. (Since we are assuming associativity, we can omit parentheses.) Without the $\ominus$ operator, the best possible $\mathcal{A}$-plan uses nine $\oplus$ operations. However, if we allow the use of $\ominus$, we can construct a plan that uses four $\oplus$ operations and four $\ominus$ operations, for a total of eight operations, by constructing a sequence of nodes with the following labels: $u \oplus v$, $u \oplus v \oplus w$, $a \oplus v \oplus w \oplus x$, $u \oplus v \oplus w \oplus x \oplus y$,

Table 4.4: Complexity Results for Optimally Sharing Aggregation

A1	A2	A3	A4	A5	Complexity
N	*	*	*	N	PTIME
N	N	N	*	Y	PTIME
N	Y	N	*	Y	PTIME
N	N	Y	*	Y	PTIME
N	Y	Y	*	Y	$O(1)$
Y	*	N	Y	N	NP-complete
Y	*	N	Y	Y	NP-complete
Y	*	Y	Y	N	NP-complete
Y	*	Y	*	Y	$O(1)$

$(u \oplus v \oplus w \oplus x \oplus y) \ominus x$, $(u \oplus v \oplus w \oplus x \oplus y) \ominus w$, $(u \oplus v \oplus w \oplus x \oplus y) \ominus v$, and $(u \oplus v \oplus w \oplus x \oplus y) \ominus u$. Thus, if $\oplus$ satisfies A5, we extend the $\oplus$-expressions over a set of variables X by starting with the variables in X and closing off under both the $\oplus$ and $\ominus$ operators, and we extend the definition of $\mathcal{A}$-plans to allow the use of the $\ominus$ operator in addition to the $\oplus$ operator. Although $\oplus$-expressions are extended to include the user of the $\ominus$ operator, we restrict query expressions to contain only the $\oplus$ operator.

Let $\mathcal{A}$ be the set of the algebraic axioms above that the abstract $\oplus$ operator under consideration satisfies. We provide a nearly complete characterization of the complexity of shared aggregation for the various possibilities for $\mathcal{A}$. Table 4.4 summarizes our complexity results. Note that this includes a complete characterization for commutative aggregates. The first line of Table 4.4 is handled in Proposition 4.2.1. Lines 2 through 4 are handled in Theorem 4.2.2. Line

5 and 9 are handled in Theorem 4.2.3. Theorems 4.2.4, and 4.2.5 provide the complexity results for lines 7 and 8, respectively. In Section 4.1, we already saw that the problem is NP-complete when $\{A1, A3, A4\} \subseteq \mathcal{A}$ and $A5 \notin \mathcal{A}$, giving us line 6 of Table 4.4. We do not yet have complexity bounds for the cases corresponding to lines 6 through 9, when $A4 = N$.

Proposition 4.2.1 *If $A1, A5 \notin \mathcal{A}$ and E is a finite set of $\oplus$-expressions over a finite set of variables X, then we can find a min-cost $\mathcal{A}$-plan in time quadratic in the total size of the expressions in E.*

Proof If $A1, A5 \notin \mathcal{A}$, then there is a straightforward polynomial-time algorithm for finding the optimal shared plan for all combinations of the remaining axioms. The algorithm turns the subtrees of the parse trees of the expressions in E into a canonical form and then merges identical canonicalized subtrees in a bottom-up fashion. This can be done in time quadratic in the total size of the parse trees. $\square$

Theorem 4.2.2 *If $A1 \notin \mathcal{A}$, $A5 \in \mathcal{A}$, $\{A2, A3\} \cap \mathcal{A}^c \neq \emptyset$, and E is a finite set of $\oplus$-expressions over a finite set of variables X, then we can find a min-cost $\mathcal{A}$-plan in time quadratic in the total size of the expressions in E.*

Proof Let ε denote a "null" node with no label. If T and T' are trees, let $T \oplus T'$ denote the tree with a root labeled $\oplus$ connected to two subtrees, T and T'. If $A4 \notin \mathcal{A}$, then the edge to subtree T is colored red, and the edge to subtree T' is colored black; otherwise both edges are colored black. Let $T \ominus T'$ denote the tree with a root labeled $\ominus$ connected to two subtrees, T and T', by a red edge and a black edge, respectively. We write $T = T'$ iff T and T' are isomorphic up to graph structure, edge coloring, and node labels. For any expression e involving possibly both $\oplus$ and $\ominus$ operators, we define a tree $T(e)$ inductively on the structure of e as follows. If e is a variable x, then $T(e)$ consists of a single node labeled x. If e is of the form $e_1 \oplus e_2$, then, $T(e) = T(e_1)$ if $T(e_2) = \varepsilon$; else, $T(e) = T(e_2)$ if $T(e_1) = \varepsilon$; else, $T(e) = T(e_1)$ if $T(e_1) = T(e_2)$ and $A3 \in \mathcal{A}$; else, $T(e) = T$ if $T(e_1) = T \ominus T(e_2)$; else, $T(e) = T$ if $T(e_2) = T \ominus T(e_1)$ and $A4 \in \mathcal{A}$; otherwise, $T(e) = T(e_1) \oplus T(e_2)$. If e is of the form $e_1 \ominus e_2$, then, $T(e) = T(e_1)$ if $T(e_2) = \varepsilon$; else, $T(e) = \varepsilon$ if $T(e_1) = T(e_2)$ and $A2 \in \mathcal{A}$; else, $T(e) = T(e_2)$ if $T(e_1) = T(e_2)$ and $A3 \in \mathcal{A}$; else, $T(e) = T$ if $T(e_1) = T \oplus T(e_2)$; otherwise, $T(e) = T(e_1) \ominus T(e_2)$. Then it is not hard to see that two expressions e and e' are $\mathcal{A}$-equivalent iff $T(e) = T(e')$. By the inductive nature of the tree definition, every non-empty subtree of $T(e)$ is isomorphic to $T(\hat{e})$ for some subexpression $\hat{e}$ of e. Furthermore, if e is an expression involving only the $\oplus$ operator, then for every subexpression $\hat{e}$, $T(\hat{e})$ is a subtree of $T(e)$. Putting these two facts together, we see that if e and e' are two $\mathcal{A}$-equivalent expressions such that e involves only the $\oplus$ operator, then every subexpression of e is $\mathcal{A}$-equivalent to some subexpression of e'.

Consider any optimal $\mathcal{A}$-plan for E and any lowest (with respect to height in the $\mathcal{A}$-plan) node v with label e' such that e' contains the $\ominus$ operator and is $\mathcal{A}$-equivalent to some $e \in E$. Without loss of generality, we can assume that $e = e_1 \oplus e_2$, since the expressions in E involve only the $\oplus$ operator, and since we can ignore trivial expressions that are just variables. Then the set of labels of the nodes below v is the set of subexpressions of e', by definition of an $\mathcal{A}$-plan. So every subexpression of e must be $\mathcal{A}$-equivalent to the label of some node below v, since e involves only the $\oplus$ operator. In particular, e_1 and e_2 are $\mathcal{A}$-equivalent to some nodes u and w below v. Note that e_1 and e_2 both involve only the $\oplus$ operator since they are subexpressions of e. Add e_1 and e_2 to E, and observe that the plan is still $\mathcal{A}$-optimal for E, since e_1 and e_2 are computed by the nodes u and w anyway. Replace the two edges incoming to v by edges from u and w, relabeling the nodes above (and including) v appropriately. Note that this does not increase the height of any node since u and w were already below v. Repeat this process until there are no more nodes with a label that contains the $\ominus$ operator and is equivalent to some expression in E. This loop terminates because, at every step, we either decrease the height of the lowest such node (if the labels of either u or w contain the $\ominus$ operator) or else we eliminate at least one $\ominus$ operator from the labels of the query nodes (if the labels of neither u nor w contain the $\ominus$ operator, in which case all $\ominus$ operators in v's label are eliminated). Therefore, an optimal plan can be found by considering plans involving only the $\oplus$ operator, and so our previous algorithm works in this case as well. Therefore, the algorithm outlined in the proof of Proposition 4.2.1 produces an optimal $\mathcal{A}$-plan here as well. $\square$

Theorem 4.2.3 *If $\{A1, A2\} \cap \mathcal{A} \neq \emptyset$ and $\{A3, A5\} \subseteq \mathcal{A}$, then we can find a min-cost plan in $O(1)$ time.*

Proof Suppose that $\{A1, A3, A5\} \subseteq \mathcal{A}$. Consider any a and b. By A5, there is a unique c such that $c \oplus a = b$. Then we have the following.

$$\begin{aligned} c \oplus (a \oplus b) &= (c \oplus a) \oplus b && \text{(by A1)} \\ &= (c \oplus a) \oplus (c \oplus a) && \text{(by choice of } c\text{)} \\ &= c \oplus a && \text{(by A3)} \end{aligned}$$

Thus, since right-divisors are unique by A5, $a = a \oplus b$. By A3, $a = a \oplus a$, and so, by uniqueness of right-divisors, $a = b$. Thus, $a = b$ for any a and b. So aggregation is trivial in this case, since $\oplus$ is a constant function and hence all $\oplus$-expressions are equivalent.

Now suppose that $\{A2, A3, A5\} \subseteq \mathcal{A}$. By A2, there is some e such that $a \oplus e = a$ for all a. However, $a \oplus a = a$ for all a by A3. But, by A5, divisors are unique, and so $a = e$ for all a. So aggregation is trivial in this case, since $\oplus$ is a constant function and hence all $\oplus$-expressions are equivalent. $\square$

Theorem 4.2.4 *Finding a min-cost $\mathcal{A}$-plan for E is NP-hard when $\{A1, A4\} \subseteq \mathcal{A}$, $\{A3, A5\} \subseteq \mathcal{A}^c$, and E is a finite set of $\oplus$-expressions over a finite set of variables X.*

Proof We reduce this to the *exact-cover problem*, which is well-known to be NP-complete [48]. Recall that, in the exact-cover problem, we are given a finite 'universal' set U and a finite collection $\mathcal{S}$ of subsets of U, and we must determine whether there is some $\mathcal{S}' \subseteq \mathcal{S}$ such that $S \cap S' = \emptyset$ for all distinct $S, S' \in \mathcal{S}'$, and such that $\cup_{S \in \mathcal{S}'} S = U$.

Consider any instance of exact cover. Let U be the universal set. Let $\mathcal{S}$ be the finite collection of subsets of U that cover U. We can convert this into an instance of the problem of finding a minimum-cost $\mathcal{A}$-plan using the following construction.

Let the set of variables $X = U \cup \{x_S : S \in \mathcal{S}\}$ so that we have one variable for each element of the universal set, and one distinct new variables x_S for each set $S \in \mathcal{S}$. Let $<_X$ be an arbitrary ordering on the variables in X. For each non-empty set $S \subseteq X$, we define a 'canonical' $\oplus$-expression e_S as in the proof of Theorem 4.1.2. Let $E'' = \{e_S \oplus x_S : S \in S\}$. Let E' be the closure of E'' under sub-expressions. Let $E = E' \cup \{e_X\}$. Note that this construction can be done in polynomial time.

We now argue that $\mathcal{S}$ contains an exact cover of U iff the extra cost of all minimum-cost $\mathcal{A}$-plans for E is at most $|\mathcal{S}| - 2$. Then we can tell whether or not $\mathcal{S}$ contains an exact cover of U simply by examining the $\mathcal{A}$-plan generated by any algorithm that finds a min-cost $\mathcal{A}$-plan.

For the "if" direction, suppose $\mathcal{S}$ contains an exact cover of U, say $\mathcal{S}' \subseteq \mathcal{S}$. Then we can construct a plan with extra cost $|S| - 2$ as follows. Start with the parse trees of the expressions in E'', merging those nodes with $\mathcal{A}$-equivalent labels if necessary to form a single plan. This plan clearly computes all the expressions in E'' and uses no extra nodes since E'' is closed under subexpressions. The only expression left to complete is e_X. This can be done by aggregating together the $|\mathcal{S}'|$ nodes labeled e_S, for each $S \in \mathcal{S}'$, and the $|\mathcal{S}| - |\mathcal{S}'|$ nodes labeled $e_{\{x_S\}}$ for each $S \in \mathcal{S} \setminus \mathcal{S}'$. This uses $|\mathcal{S}| - 1$ aggregation nodes, out of which $|\mathcal{S}| - 2$ are extra nodes (the node labeled e_X does not count as an extra node since $e_X \in E$).

For the "only if" direction, suppose that there is some $\mathcal{A}$-plan for E that uses at most $|\mathcal{S}| - 2$ extra nodes. For a contradiction, suppose that $\mathcal{S}$ contains no exact cover of U. Consider all the nodes with paths leading to the node labeled by e_X. These nodes induce an arborescence in the DAG rooted at the e_X node. This arborescence must contain all the extra nodes since all the other query nodes are closed under subexpressions and can hence be aggregated using no extra nodes. Consider the set V of nodes in the arborescence that are labeled by $\oplus$-expressions in E' and have an edge into an extra node. Then there must be $|V| - 1$ nodes descending from these $|V|$ nodes in the arborescence. Out of these, $|V| - 2$ will be extra nodes, and one will be the query node labeled e_X. Now for each $S \in \mathcal{S}$, either the node labeled x_S or else the node labeled $e_S \oplus x_S$ must be in

V since these are the only two nodes labeled with an $\oplus$-expression in E' that contain the variable x_S by construction of E' and since x_S was a distinct new variable. Thus $|V| \geq |\mathcal{S}|$. Suppose for a moment that $|V| = |\mathcal{S}|$. Then by the pigeon-hole principle, all the nodes in V are labeled either with x_S or $e_S \oplus x_S$ for some $S \in \mathcal{S}$. Let $\mathcal{S}'$ be the collection of sets $S \in \mathcal{S}$ for which some nodes in V is labeled $e_S \oplus x_S$. Now it is easy to see that, since $\{A1, A4\} \subseteq \mathcal{A}$ and $\{A5, A3\} \subseteq \mathcal{A}^c$, two $\oplus$-expressions over a set of variables X are $\mathcal{A}$-equivalent iff the multiset of variables appearing in the two expressions are equal. Therefore, $\mathcal{S}'$ must form an exact cover of U, contradicting our initial assumption that $\mathcal{S}$ contained no exact cover of U. There it is not possible that $|V| = |\mathcal{S}|$. So it must be the case that $|V| > |\mathcal{S}|$, and hence there are more than $|\mathcal{S}| - 2$ extra nodes in the $\mathcal{A}$-plan since there are $|V| - 2$ extra nodes. This completes the second direction. $\square$

Theorem 4.2.5 *Finding a min-cost $\mathcal{A}$-plan for E is NP-hard when $\{A1, A4, A5\} \subseteq \mathcal{A}$, $\{A3\} \subseteq \mathcal{A}^c$, and E is a finite set of $\oplus$-expressions over a finite set of variables X.*

Proof We use the same reduction as the proof of Theorem 4.2.4 and observe that the addition of the $\ominus$ operator does not change the proof; in particular, it is still the case that $|V| > |\mathcal{S}|$ iff $\mathcal{S}$ contains no exact cover for U. $\square$

4.3 Related Work

Early work on multiquery optimization includes work done by Sellis [71] to provide shared plans for select, project, and joint queries. However, this work did not consider shared aggregation. Cocke looked at sharing work for computing expressions where the operators were commutative non-associative operators in the context of compiler optimization by finding global common subexpressions [22]. In contrast, our work focuses on operators that are associative as well as commutative.

There has recently been a lot of work done in the context of data streaming and sensor networks that is closely related to ours. For example, Dobra, Garofalakis, et al. introduce a technique for computing approximate aggregates by sharing work across multiple queries [29] transmitting 'sketches' of the data rather than the entire data. In our setting, it is important to find the exact aggregate values in order to ensure the desired economic properties of the auction, such as truthfulness and envy-freeness. Trigoni, Yao, et al. look at optimizing aggregates in sensor networks [79]. They focus on communication cost and use more coarse cost-model than ours. They consider a unit cost of sending a vector of aggregates whose length depend on the problem size. In contrast, we consider the cost of computing each individual aggregate, which is a more accurate computation cost model. Zhang, Koudas, et al. consider the problem of sharing aggregation over data streams in the Gigascope system where the queries are aggregates of group-bys of several attributes [86]. They use 'phantom' group-by aggregates that contain partial results for multiple queries and propose a greedy heuristic finding the optimal set of phantoms for count and sum aggregates. Krishnamurthy, Wu, and Franklin suggest the use of fragments [52] (i.e., grouping inputs by the set of selection predicates that they satisfy) that we use in the first stage of our algorithm. However, they did not take advantage of further algebraic properties of the aggregation operator as we do. Huebsch, et al. consider the problem of sharing aggregate computation for distributed queries and classify aggregates based on whether or not they are linear and whether or not they are duplicate-insensitive [43]. Like [79], this work uses a coarser cost model than ours. Other work on multiquery aggregation includes that of Silberstein and Yang where they look at aggregation using a set of multicast trees in a network that satisfies certain assumptions on the relationship between the trees [73]. Our setting is different in that, rather than being given a network, we have to design an optimal network between sources (inputs) and sinks (queries).

4.4 Conclusions

In this chapter, we highlight the opportunity for sharing work when there is a high search volume by sharing the winner-determination computation across multiple sponsored search auctions that occur simultaneously. We provided techniques for both separable and non-separable click-through rates even if the advertiser-specific factor is different across bid phrases in the case of separable click-through rates. As future work, we would like to determine whether these algorithms have provable approximation bounds. We studied the problem of sharing aggregation operators between bidding programs as a function of the algebraic axioms that they satisfy. We provide complexity results for most combinations of the axioms, including a complete characterization for all commutative cases. Future work includes determining the complexity of the few remaining non-commutative cases.

CHAPTER 5

REVEALING AUCTION DATA WHILE LIMITING DISCLOSURE

5.1 Introduction

In order to enable advertisers to improve their bidding strategies, it would be helpful if search providers release historical search and auction data. However, they must do so in such a way as to limit the disclosure about any individual advertiser while still providing as much useful information as possible. For example, suppose the search provider wants to release a table containing the username, age, location, gender, and search topic that the user has been most interested in (e.g., celebrity gossip, online gambling, etc.), as depicted in Table 5.1. This table cannot be released as-is, since it would reveal the search habits of individual users. Instead, the search provider could release an anonymized version of this data by partially or completely suppressing identifying attributes (i.e., username, age, location, and gender) from the table, as depicted in Table 5.2. Then, each time a user submits a search, the search provider could simply inform the advertisers of which bucket the user falls into.

Finding an appropriately anonymized version of the data is an instance of the privacy-preserving data publishing problem, which is applicable to a wide variety of settings beyond search auctions, including publishing census data for policy makers, and hospital patient data for medical researchers. We consider the following general situation. A data publisher (such as a search provider, or hospital) has collected useful information about a group of individuals (such as user profiles, or patient records) and would like to publish this data while preserving the privacy of the individuals involved. The information is stored

Table 5.1: Original Table

	non-sensitive			sensitive
Name	**Zip**	**Age**	**Sex**	**Interest**
Bob	14850	23	M	Anarchism
Charlie	14850	24	M	Anarchism
Dave	14850	25	M	Online Dating
Ed	14850	27	M	Online Dating
Frank	14853	29	M	Hacking
Gloria	14850	21	F	Anarchism
Hannah	14850	22	F	Anarchism
Irma	14853	24	F	Online Gambling
Jessica	14853	26	F	Adult Sites
Karen	14853	28	F	Celebrity Gossip

as a table (as in Table 5.1) where each record corresponds to a unique individual and contains a sensitive attribute (e.g., top interest, or disease) and some non-sensitive attributes (e.g., address, gender, age) that might be learned using externally available data (e.g., phone books, birth records). The data publisher would like to limit the disclosure of the sensitive values of the individuals in order to defend against an attacker who possibly already knows some facts about the table. Our goal in this chapter is to quantify the precise effect of background knowledge possessed by an attacker on the amount of disclosure and to provide algorithms to check and ensure that the amount of disclosure is less than a specified threshold.

The problem we solve is of real and practical importance; an egregious example of a privacy breach was the reidentification of a user from anonymized

Table 5.2: 5-anonymous Table

Name	Zip	Age	Sex	Interest
*	1485*	2*	M	Anarchism Online Dating Hacking Anarchism Online Dating
*	1485*	2*	F	Anarchism Online Gambling Anarchism Celebrity Gossip Adult Sites

search logs published by AOL [11]. An even more famous incident was the discovery of the medical records of the Governor of Massachusetts from an easily accessible and supposedly anonymized dataset. All that was needed was to link it to voter registration records [77]. To defend against such attacks, Samarati and Sweeney [69] introduced a privacy criterion called k-anonymity which requires that each individual be indistinguishable (with respect to the non-sensitive attributes) from at least $k - 1$ others. This is done by grouping individuals into *buckets* of size at least k, and then permuting the sensitive values in each bucket and sufficiently masking their externally observable non-sensitive attributes. Table 5.2 depicts a 5-anonymous version of Table 5.1. Table 5.3 depicts the permutation of sensitive values that was used to construct this table.

However, k-anonymity does not adequately protect the privacy of an indi-

Table 5.3: Bucketized Table

Name	Zip	Age	Sex	Interest
Bob	14850	23	M	Anarchism
Charlie	14850	24	M	Online Dating
Dave	14850	25	M	Hacking
Ed	14850	27	M	Anarchism
Frank	14853	29	M	Online Dating
Gloria	14850	21	F	Anarchism
Hannah	14850	22	F	Online Gambling
Irma	14853	24	F	Anarchism
Jessica	14853	26	F	Celebrity Gossip
Karen	14853	28	F	Adult Sites

vidual;[1] for example, when all individuals in a bucket have the same sensitive value, the sensitive values of the individuals in that bucket is disclosed regardless of the bucket size. Even when there are multiple sensitive values in the same bucket, the frequencies of the sensitive values in the bucket still matter when an attacker has some background knowledge about the particular individuals in the table. Suppose the data publisher has published the 5-anonymous table as depicted in Table 5.2. Consider an attacker Alice who would like to learn the interests of her neighbors. One of her neighbors is Ed, a 27 year-old male living in Ithaca (zip code 14850). Alice knows that Ed is a user of the search provider that published the anonymized dataset in Table 5.3, and she wants to find out Ed's interest. Using her knowledge of Ed's age, gender, and zip-code, Alice can identify the bucket in the anonymized table that Ed belongs to (namely, the first bucket). Alice does not know which interest listed within that bucket is Ed's

[1]Indeed, the definition of k-anonymity does not even mention the sensitive attribute!

since the sensitive values were permuted. Therefore, without additional knowledge, Alice's estimate of the probability that Ed is interested in anarchism is $2/5$. But suppose that Alice knows that Ed doesn't know much about computers and is therefore extremely unlikely to be interested in hacking. After ruling this possibility out, the probability that Ed is interested in anarchism increases to $1/2$. Now, if Alice also somehow discovers that Ed is not interested in gambling, then the fact that he is interested in anarchism becomes certain. Here, two pieces of knowledge of the form "Ed does not have X" were enough to fully disclose Ed's sensitive attribute. To guard against this, Machanavajjhala et al. [57] proposed a privacy criterion called ℓ-diversity that ensures that it takes at least $\ell - 1$ such pieces of information to sufficiently disclose the sensitive value of any individual. The main idea is to require that, for each bucket, the ℓ most frequent sensitive values are roughly equi-probable.

ℓ-diversity focuses on one type of background knowledge: knowledge of the form "individual X does not have sensitive value Y". But an attacker might well have other types of background knowledge. For example, suppose Alice lives across the street from a married couple, Charlie and Hannah. Once again, using her knowledge of their genders, ages and zip-codes, Alice can identify the buckets Charlie and Hannah belong to. Without additional background knowledge, Alice thinks that Charlie is interested in anarchism with probability $2/5$. But suppose that Alice knows that Charlie is very impressionable when it comes to political outlook. Then Alice can deduce that if Hannah is highly interested in anarchism, then it is extremely likely that Charlie is as well. This knowledge allows her to update her probability that Charlie is interested in anarchism to $10/19$. We show how these probabilities are computed in Section 5.3. ℓ-diversity does not guard against the type of background knowledge in this example.

It is thus clear that we need a more general-purpose framework that can capture knowledge of *any* property of the underlying table that an attacker might know. Moreover, unlike in the two examples above where we knew Alice's background knowledge, we will not assume that we know exactly what the attacker knows. We therefore take the following approach. In Section 5.2, we propose a language that is expressive enough to capture any property of the sensitive values in a table. This language enables us to decompose background knowledge into basic units of information. Then, given an anonymized version of the table, we can quantify the worst-case disclosure risk posed by an attacker with k such units of information; k can be thought of as a bound on the power of an attacker. In Section 5.3, we show how to efficiently preserve privacy by ensuring that the *worst-case* (i.e., maximum) disclosure for *any* k pieces of information is less than a specified threshold. Furthermore, we show to integrate our techniques into existing frameworks to find a "minimally sanitized" table for which the maximum disclosure is less than a specified threshold. In Section 5.4, we show that our techniques can be applied to other types of background knowledge. In particular, we frame the existing notion of ℓ-diversity within our framework and show that it does indeed bound the maximum disclosure with respect to the type of background knowledge that it considers. Such formal analysis and proof of correctness was missing in the original paper [57]. We present experiments in Section 5.5, related work in Section 5.6, and we conclude in Section 5.7.

To the best of our knowledge, this is the first such formal analysis of the effect of unknown background knowledge on the disclosure of sensitive information.

5.2 Framework

We begin by modeling the data publishing situation formally. Let P be a (finite) set of individuals. For each $p \in P$, we associate a tuple t_p which has one sensitive attribute S (e.g., top interest, or disease) with finite domain and one or more non-sensitive attributes. We overload notation and use S to represent both the sensitive attribute and its domain. The data publisher has a table T, which is a set of tuples corresponding to a subset of P. The publisher would like publish T in a form that protects the sensitive information of any individual from an attacker with background knowledge that can be expressed in a language $\mathcal{L}$. (We propose such a language to express background knowledge in Section 5.2.2.)

5.2.1 Bucketization

We first need to carefully describe how the published data is constructed from the underlying table if we are to correctly interpret this published data. That is, we need to specify a sanitization method. We briefly describe two popular sanitization methods.

- The first, which we term *bucketization* [84], is to partition the tuples in T into *buckets*, and then to separate the sensitive attribute from the non-sensitive ones by randomly permuting the sensitive attribute values within each bucket. The sanitized data then consists of the buckets with permuted sensitive values.
- The second sanitization technique is *full-domain generalization* [77], where we coarsen the non-sensitive attribute domains. The sanitized data con-

sists of the coarsened table along with generalization used. Note that, unlike bucketization, the exact values of the non-sensitive attributes are not released; only the coarsened values are released.

Note that if the attacker knows the set of individuals in the table and their non-sensitive values, then full-domain generalization and bucketization are equivalent. In this work, we use bucketization as the method of constructing the published data from the original table T, although all our results hold for full-domain generalization as well. We plan to extend our algorithms to work for other sanitization techniques, such as data swapping [24] (which, like bucketization, also permutes the sensitive values, but in more complex ways) and suppression [69], in the future.

We now specify our notion of bucketization more formally. Given a table T, we partition the tuples into buckets (i.e., horizontally partition the table T according to some scheme), and within each bucket, we apply an independent random permutation to the column containing S-values. The resulting set of buckets, denoted by $\mathcal{B}$, is then published. For example, if the underlying table T is as depicted in Table 5.1, then the publisher might publish bucketization $\mathcal{B}$ as depicted in Table 5.3. Of course, for added privacy, the publisher can completely mask the identifying attribute (Name) and may partially mask some of the other non-sensitive attributes (Age, Sex, Zip).

For a bucket $b \in \mathcal{B}$, we use the following notation.

- P_b denotes the set of individuals $p \in P$ with tuples $t_p \in b$,
- n_b denotes the number of tuples in bucket b,
- $n_b(s)$ denotes the frequency of the sensitive value $s \in S$ in bucket b, and

- $s_b^0, s_b^1, \ldots$ denote the sensitive values appearing in bucket b in descending order of frequency.

5.2.2 Background Knowledge

We pessimistically assume that the attacker has managed to obtain complete information about which individuals have records in the table, what their non-sensitive data is, and which buckets in the bucketization these records fall into. That is, we assume that the attacker knows P_b, the set of individuals in bucket b, for each $b \in \mathcal{B}$, and knows $t_p[X]$ for every individual p in the table and every non-sensitive attribute X. We call this *full identification information*. One way of obtaining identification information in practice is to link quasi-identifying non-sensitive attributes published in the bucketization (e.g., address, gender, age) with publicly available data (e.g., phone directories, birth records) [77].

We make the standard random-worlds assumption [10]: in the absence of any further knowledge, we consider all tables consistent with this bucketization to be equally likely. That is, the probability of $t_p \in b$ having s for its sensitive attribute is $n_b(s)/n_b$ since each assignment of sensitive attributes to tuples within a bucket is equally likely.

We now need to consider knowledge beyond the identification information that an attacker might possess. We assume that this further knowledge is the knowledge that the underlying table satisfies a given *predicate* on tables. That is, the attacker knows that the underlying table is among the set of tables satisfying the given predicate. This is a rather general assumption. For example, "the average age of people interested in online gambling in the table is 48 years" could be one such predicate. In order to quantify the power of such knowledge, we use the notion of a *basic unit* of knowledge, and we propose a language which consists of finite conjunctions of such basic units. Given full identification information, we desire that any predicate on tables be expressible using a conjunction of the basic units that we propose. We employ a very simple propositional syntax.

Definition 5.2.1 (Atoms) *An* atom *is a formula of the form $t_p[S] = s$, for some value $s \in S$ and individual $p \in P$ with tuple $t_p \in T$. We say that atom $t_p[S] = s$* involves *individual p and value s.*

The interpretation of atoms is obvious: $t_{\text{Jack}}[\text{Interest}] = \text{online dating}$ says that the Jack's tuple has the value online dating for the sensitive attribute Interest.

The basic units of knowledge in our language are *basic implications*, defined below.

Definition 5.2.2 (Basic implications) *A* basic implication *is a formula of the form*

$$(\wedge_{i \in [m]} A_i) \rightarrow (\vee_{j \in [n]} B_j)$$

for some $m \geq 1, n \geq 0$ and atoms A_i, B_j, $i \in [m], j \in [n]$ (note that we use the standard notation $[n]$ to denote the set $\{0, \ldots, n-1\}$).

The fact that basic implications are a sufficiently expressive "basic unit" of knowledge is made precise by the following theorem.

Theorem 5.2.3 (Completeness) *Given full identification information and any predicate on tables, one can express the knowledge that the underlying table satisfies the* *identification information and the given predicate using a finite conjunction of basic implications.*

Proof Since the attacker is assumed to have full identification information, the values of $t_p[X]$ and which bucket t_p falls into are assumed to be common knowledge. The only remaining information that it takes to completely define a particular table is the mapping between individuals and sensitive values within each bucket. Thus we need to show that a finite conjunction of basic implications can express any set of mappings between individuals in the table and sensitive values. Note that, since the domain of S and the table size are finite, there are only finitely many mappings between individuals in the table and sensitive values. Any particular such mapping between individuals and sensitive values can clearly be represented by a finite conjunctions of atoms of the form $t_p = s$. Thus any set of mappings between individuals and sensitive values can be represented by a finite disjunction of finite conjunctions of atoms. We show that, in fact, a finite conjunction of basic implications can represent *any* finite Boolean combination of atoms.

Consider any finite Boolean combination of atoms. Without loss of generality, assume that the formula is in conjunction normal form. It thus remains to show that any disjunction of literals (i.e., atoms or their negations) can be represented by a finite conjunction of basic implications. We break this into two cases depending on whether or not the given disjunction contains at least one negative literal. In the first case, if the disjunction contains at least one negative literal, then the disjunction is equivalent to a single a basic implication $\varphi \to \theta$ where φ is the conjunction of the atoms appearing in the negative literals and ψ is the disjunction of the atoms appearing in the positive literals. In the second case, if the disjunction contains no negative literals, the disjunction is equivalent to the following conjunction of basic implications: $\wedge_{s \in S}(t_p = s \rightarrow \varphi)$, where φ is the given disjunction itself and p is any individual in the table. $\square$

Hence we can model arbitrarily powerful attackers.[2] Consider an attacker who knows the sensitive value of every individual in the table except for Bob. Then publishing any bucketization will reveal Bob's sensitive value. To avoid pathological and unrealistic cases like this, we need to assume a bound on the power of an attacker. We model attackers with bounded power by limiting the number of basic implications that the attacker knows. That is, the attacker knows a single formula from language $\mathcal{L}^k_{\text{basic}}$ defined below.

Definition 5.2.4 $\mathcal{L}^k_{\text{basic}}$ *is the language consisting of conjunctions of* k *basic implications. That is,* $\mathcal{L}^k_{\text{basic}}$ *consists of formulas of the form* $\wedge_{i \in [k]} \varphi_i$ *where each* φ_i *is a basic implication.*

k can thus be viewed as a bound on the attacker's power and can be increased to provide more conservative privacy guarantees.

Note that our choice of basic implications for the "basic unit" of our language has important consequences on our assumptions about the attacker's power. In particular, some properties of the underlying table might require a large number of basic implications to express. Since basic implications are essentially CNF clauses with at least one negative atom, our language suffers from an exponential blowup in the number of basic units required to express arbitrary DNF formulas. It may be that other choices of basic units may lead to equally expressive

[2]A major shortcoming of the ℓ-diversity definition was that its choice of "basic unit" of knowledge was essentially negated atoms (i.e., $\neg t_p[S] = s$) which cannot capture all properties of the underlying table. For example, negations cannot express basic implications in general.

languages while at the same time requiring fewer basic units to express certain natural properties, and we consider this an important direction for future research. Nevertheless, many natural types of background knowledge have succinct representations using basic implications. For example, Alice's knowledge that "if Hannah is interested in anarchism, then so is Charlie" is simply the basic implication

$$t_{\text{Hannah}}[\text{Interest}] = \text{anarchism} \rightarrow t_{\text{Charlie}}[\text{Interest}] = \text{anarchism}.$$

And the knowledge that "Ed is not interested in hacking" is

$$t_{\text{Ed}}[\text{Interest}] = \text{hacking} \rightarrow t_{\text{Ed}}[\text{Interest}] = \text{online gambling}.$$

In general, we can represent $\neg t[S] = s$ by $(t[S] = s) \rightarrow (t[S] = s')$ for any choice of $s' \neq s$ since each tuple has exactly one sensitive attribute value.

Note that maintaining privacy when there is dependence between sensitive values, especially *across buckets*, is a problem that has not been previously addressed in the privacy literature. The assignments of individuals to sensitive values in different buckets are not necessarily independent. As we saw in the example with Hannah and Charlie, fixing a particular assignment in one bucket could affect what assignments are possible in another. One of the contributions of this work is that we provide a polynomial time algorithm for computing the maximum disclosure even when the attacker has knowledge of such dependencies.

5.2.3 Disclosure

Having specified how the bucketization $\mathcal{B}$ is constructed from the underlying table T and how an attacker's knowledge about sensitive information can be expressed in language $\mathcal{L}^k_{\text{basic}}$, we are now in a position to define our notion of disclosure precisely.

Definition 5.2.5 (Disclosure risk) *The* disclosure risk *of bucketization $\mathcal{B}$ with respect to background knowledge represented by some formula φ in language $\mathcal{L}^k_{\text{basic}}$ is*

$$\max_{t_p \in T, s \in S} \Pr(t_p[S] = s \mid \mathcal{B} \wedge \varphi).$$

That is, disclosure risk is the likelihood of the most highly predicted sensitive attribute assignment.

Definition 5.2.6 (Maximum disclosure) *The* maximum disclosure *of bucketization $\mathcal{B}$ with respect to language $\mathcal{L}^k_{\text{basic}}$ that expresses background knowledge is*

$$\max_{t_p \in T, s \in S, \varphi \in \mathcal{L}^k_{\text{basic}}} \Pr(t_p[S] = s \mid \mathcal{B} \wedge \varphi).$$

By our assumptions in 5.2.2, we compute $\Pr(t_p[S] = s \mid \mathcal{B} \wedge \varphi)$ by considering the set of all tables consistent with bucketization $\mathcal{B}$ and with background knowledge φ and then taking the fraction of those tables that satisfy $t_p[S] = s$. Using this, the maximum disclosure of the bucketization in Table 5.3 with respect to $\mathcal{L}^1_{\text{basic}}$ turns out to be $\frac{10}{19}$, and occurs when φ is $t_{p'} = s' \rightarrow t_p = s$ where p is an individual in the first bucket, p' is an individual in the second bucket, and s and s' are both anarchism. Our goal is to develop general techniques to:

1. efficiently calculate the maximum disclosure for any given bucketization, and
2. efficiently find a "minimally sanitized" bucketization[3] (or the set of all minimally sanitized bucketizations) for which the maximum disclosure is below a specified threshold (if any exist).

[3]We will make precise the notion of "minimally sanitized" in Section 5.3.4; we want "minimal sanitization" in order to preserve the utility of the data.

5.3 Checking And Enforcing Privacy

In Section 5.2.2, we defined basic implications as the "unit of knowledge" and showed that this was a fully expressive (in the presence of full identification information) and reasonable choice. We now show how to efficiently calculate and limit maximum disclosure against an attacker who has full identification information and has up to k additional pieces of background knowledge (i.e., up to k basic implications). In order to do this, we will show in Theorem 5.3.3 that there is a set of k basic implications that maximizes disclosure with respect to $\mathcal{L}^k_{\text{basic}}$. Furthermore, each such implication has *only one atom in the antecedent and one atom in the consequent*. This motivates the following definition.

Definition 5.3.1 (Simple implications) *A* simple implication *is a formula of the form $A \rightarrow B$ for some atoms A, B.*

5.3.1 Hardness of Computing Disclosure Risk

Unfortunately, naive methods for computing the maximum disclosure will not work – in fact, we can show that computing the disclosure risk of a given bucketization with respect to a given set of k simple implications is #P-hard. Note that k simple implications can be written in 2-CNF, for which satisfiability is easily checkable. Complexity is introduced in trying to *simultaneously* satisfy the k implications *and* the given bucketization. In fact, deciding whether a given bucketization is consistent with a set of k simple implications is NP-complete.

Theorem 5.3.2 *Given as input bucketization $\mathcal{B}$ and a conjunction of simple implications φ, the problem of deciding if $\mathcal{B}$ and φ are both satisfiable by some table T is*

NP-*complete. Moreover, given an atom C as further input, the problem of computing* $\Pr(C \mid \mathcal{B} \wedge \wedge_{i\in[k]}\varphi_i)$ *is* #P-*complete.*

Proof Consider the problem of deciding if $\mathcal{B}$ and φ are both satisfiable by some table T, given as input a bucketization and a conjunction of simple implications. It is clear that the problem is in NP, because given a mapping of tuples to sensitive values (which has a description that is linear in bucketization size), we can verify that it is indeed consistent with the bucketization and that it satisfies $\wedge_{i\in[k]}(A_i \to B_i)$ in polynomial time.

To show that the problem is NP-hard, we reduce the problem of deciding 3-CNF satisfiability, which is NP-complete, to this problem as follows. Consider any 3-CNF formula. We construct a bucketization and set of basic implications from this formula as follows. For each variable x mentioned in the 3-CNF formula, we construct a bucket containing two tuples, named t_x and $t_{\neg x}$, and two sensitive values, T and F. For each clause C of the form $X \vee Y \vee Z$ in the 3-CNF formula (where X, Y, Z are either variables or their negations), we construct a bucket containing five tuples, named $t^C_X, t^C_Y, t^C_Z, t^C_{dummy1}, t^C_{dummy2}$, and five sensitive values, T, T, T, F, F. The background knowledge then consists of the following set of statements:

- $t_x[S] = T \to t^C_X[S] = T$, for every variable x and every clause C containing literal $X \equiv x$,
- $t^C_X[S] = T \to t_x[S] = T$, for every variable x and every clause C containing literal $X \equiv x$,
- $t_{\neg x}[S] = T \to t^C_X[S] = T$, for every variable x and every clause C containing literal $X \equiv \neg x$,

- $t_X^C[S] = T \to t_{\neg x}[S] = T$, for every variable x and every clause C containing literal $X \equiv \neg x$, and
- $t_{dummy1}^C[S] = T \to t_{dummy2}^C[S] = T$, for every clause C.

Let k be the number of implications that we added above. Note that k is linear in the size of the 3-CNF formula. It is fairly clear that if there is a mapping of tuples to values that is consistent with the bucketization and background knowledge, then assigning each variable x to the value $t_x[S]$ satisfies the 3-CNF formula (since, in each bucket corresponding to a clause, at least one tuple representing a literal must have sensitive value T). So we can decide if the 3-CNF formula is satisfiable, given an oracle for our problem. Thus the decision problem is NP-complete.

It should therefore not come as a surprise that computing the probability of $\Pr(C \mid \mathcal{B} \wedge (\wedge_{i \in [k]}(A_i \to B_i)))$ is #P-complete since computing the probability and counting satisfying assignments are intimately related. We reduce the problem of counting the satisfying assignments of a 2-CNF formula, which is #P-complete [80], to an instance of computing $\Pr(A \mid \mathcal{B} \wedge (\wedge_{i \in [k]}(A_i \to A'_i)))$. Consider a 2-CNF formula φ, with variables $x_0, \ldots, x_{n-1}$. We can find a satisfying assignment of φ in polynomial time since φ is 2-CNF. Let $\wedge_{i \in [n]} X_i$ represent the satisfying assignment, where X_i is either x_i or $\neg x_i$, depending on the value of x_i in the satisfying assignment. Consider a complete binary tree with n leaf nodes, where the i^{th} leaf is associated with the literal X_i. For every non-leaf node, we introduce a new variable y and a constant number of 3-CNF clauses that are equivalent to $y \leftrightarrow (U \wedge V)$, where U and V are the literals at the left and right children of the non-leaf node. Let φ' be the conjunction of all the newly-introduced 3-CNF clauses. Then the conjunction of all the newly-introduced

3-CNF clauses implies that $z \leftrightarrow \wedge_{i\in[n]} X_i$ where z is variable associated with the root of the tree. Note that φ' is polynomial in the size of φ since the complete binary tree with n leaves has at most $O(n)$ nodes, and we introduced a constant number of clauses for each internal node. $\varphi \wedge \varphi'$ is 3-CNF formula. And $\varphi \wedge \varphi' \wedge z$ is a 3-CNF formula with exactly one satisfying assignment (namely, setting each variable in φ according to $\wedge_{i\in[n]} X_i$, and each newly-introduced variable to true). So, applying the construction from the proof of Theorem 5.3.2 to get A and A_i from $\varphi \wedge \varphi'$, it is easy to check that $\frac{1}{\Pr(t_z[S]=T|\varphi_{\mathcal{B}} \wedge \varphi \wedge \varphi')}$ is exactly the number of satisfying assignments of φ. $\square$

5.3.2 A Special Form for Maximum Disclosure

It turns out that, despite the hardness results above, computing the *maximum* disclosure with respect to language $\mathcal{L}^k_{\text{basic}}$ can be done in polynomial time. The key insight is summarized in Theorem 5.3.3.

Theorem 5.3.3 *For any bucketization, there is a set of* k simple implications, all sharing the same consequent, *such that the conjunction of these k simple implications maximizes disclosure with respect to $\mathcal{L}^k_{\text{basic}}$.*

This insight is tremendously useful in devising a polynomial-time dynamic programming algorithm for computing the maximum disclosure with respect to $\mathcal{L}^k_{\text{basic}}$ as it allows us to restrict our attention to sets of k simple implications of the form $(t_{p_i}[S] = s_i) \rightarrow (t_p[S] = s)$ for individuals $p, p_i \in P$, and values $s, s_i \in S$, $i \in [k]$. The proof of Theorem 5.3.3 follows from the following two lemmas.

Lemma 5.3.4 *For any formulas $\psi, \varphi, \theta_i, \varphi_i$,*

$$\Pr(\varphi \mid \psi \wedge (\wedge_{i\in[k]}(\theta_i \rightarrow \varphi_i))) \leq \Pr(\varphi \mid \psi \wedge (\wedge_{i\in[k]}(\theta_i \rightarrow \varphi)))$$

Proof For convenience of notation,

- let θ be $\neg(\wedge_{i\in[k]}\neg\theta_i)$,
- let χ be $(\wedge_{i\in[k]}(\theta_i \rightarrow \varphi_i))$,
- let $u = \Pr(\theta \wedge \varphi \wedge \psi)$,
- let $v = \Pr(\neg\theta \wedge \psi \wedge \varphi)$,
- let $w = \Pr(\neg\theta \wedge \psi)$,
- let $x = \Pr(\theta \wedge \chi \wedge \psi \wedge \varphi)$, and
- let $y = \Pr(\theta \wedge \chi \wedge \psi)$.

Then, for all $\psi' \in \mathcal{L}$, $\wedge_{i\in[k]}(\theta_i \rightarrow \varphi) \wedge \psi'$ is logically equivalent to $(\theta \wedge \varphi \wedge \psi') \vee (\neg\theta \wedge \psi')$. Hence,

$$\begin{aligned}
&\Pr(\varphi \mid (\wedge_{i\in[k]}(\theta_i \rightarrow \varphi)) \wedge \psi) \\
&\quad = \tfrac{\Pr((\wedge_{i\in[k]}(\theta_i\rightarrow\varphi))\wedge\psi\wedge\varphi)}{\Pr((\wedge_{i\in[k]}(\theta_i\rightarrow\varphi))\wedge\psi)} \\
&\quad = \tfrac{\Pr((\theta\wedge\varphi\wedge\varphi\wedge\psi)\vee(\neg\theta\wedge\psi\wedge\varphi))}{\Pr((\theta\wedge\varphi\wedge\psi)\vee(\neg\theta\wedge\psi))} \\
&\quad = \tfrac{\Pr(\theta\wedge\varphi\wedge\psi)+\Pr(\neg\theta\wedge\psi\wedge\varphi)}{\Pr(\theta\wedge\varphi\wedge\psi)+\Pr(\neg\theta\wedge\psi)} \\
&\quad = \tfrac{u+v}{u+w}.
\end{aligned}$$

Similarly, using that fact that, for all $\psi' \in \mathcal{L}$, $\wedge_{i\in[k]}(\theta_i \rightarrow \varphi_i) \wedge \psi'$ is logically equivalent to $(\theta \wedge \chi \wedge \psi') \vee (\neg\theta \wedge \psi')$, we get that

$$\begin{aligned}
&\Pr(\varphi \mid (\wedge_{i\in[k]}(\theta_i \rightarrow \varphi_i)) \wedge \psi) \\
&\quad = \tfrac{\Pr(\theta\wedge\chi\wedge\psi\wedge\varphi)+\Pr(\neg\theta\wedge\psi\wedge\varphi)}{\Pr(\theta\wedge\chi\wedge\psi)+\Pr(\neg\theta\wedge\psi)} \\
&\quad = \tfrac{x+v}{y+w}.
\end{aligned}$$

However, since $\theta \wedge \chi \wedge \psi \wedge \varphi$ logically implies both $\theta \wedge \varphi \wedge \psi$ and $\theta \wedge \chi \wedge \psi$, we have $u \geq x$ and $y \geq x$. Similarly, since $\neg\theta \wedge \psi \wedge \varphi$ logically implies $\neg\theta \wedge \psi$, we have $v \leq w$. So, since $u, v, w, x, y \geq 0$, we get $\frac{u+v}{u+w} \geq \frac{x+v}{x+w} \geq \frac{x+v}{y+w}$, thus proving the required result. □

Starting with any set of k basic implications that maximize disclosure, Lemma 5.3.4 enables us to replace the consequent in all the basic implications by a single common atom (namely the atom corresponding to the highest predicted assignment of sensitive value to an individual), while still maintaining maximum disclosure. Note that there always exists *some* set of k basic implications that maximize disclosure since there are only finitely many atoms and therefore $\mathcal{L}^k_{\text{basic}}$ is finite.

Lemma 5.3.5 *For any formulas ψ, B, θ_i, where B is an atom and θ_i is a conjunction of atoms, there exist atoms A_i such that*

$$\Pr(B \mid \psi \wedge (\wedge_{i\in[k]}(\theta_i \rightarrow B))) \leq \Pr(B \mid \psi \wedge (\wedge_{i\in[k]}(A_i \rightarrow B))).$$

Proof Since each of the implications $(\theta_i \rightarrow B)$ is basic, θ_i is a conjunction of positive atoms. Hence, from each of the θ_i pick one of the atoms A_i (the atoms need not be distinct). Clearly, $\theta_i \rightarrow A_i$. Hence, the required result follows from Lemma 5.3.6. □

Lemma 5.3.6 *For all $\theta_0, \ldots, \theta_{k-1}, \theta'_0, \ldots, \theta'_{k-1}, \psi, \varphi$, such that $\theta_i \rightarrow \theta'_i$, for all $i \in [k]$, we have*

$$\Pr(\varphi \mid \psi \wedge (\wedge_{i\in[k]}(\theta_i \rightarrow \varphi))) \leq \Pr(\varphi \mid \psi \wedge (\wedge_{i\in[k]}(\theta'_i \rightarrow \varphi))).$$

Proof Note that

$$\begin{aligned}\Pr(\varphi \mid (\wedge_{i\in[k]}(\theta_i \rightarrow \varphi)) \wedge \psi) &= \tfrac{\Pr((\wedge_{i\in[k]}(\theta_i\rightarrow\varphi))\wedge\psi\wedge\varphi)}{\Pr((\wedge_{i\in[k]}(\theta_i\rightarrow\varphi))\wedge\psi)} \\ &= \tfrac{\Pr(\varphi\wedge\psi)}{1-\Pr(\vee_{i\in[k]}(\theta_i\wedge\neg\varphi)\vee\neg\psi)}.\end{aligned}$$

So it is enough to show that

$$\Pr(\vee_{i\in[k]}(\theta_i \wedge \neg\varphi) \vee \neg\psi) \quad \leq \quad \Pr(\vee_{i\in[k]}(\theta'_i \wedge \neg\varphi) \vee \neg\psi).$$

We know that $\theta_i \rightarrow \theta'_i$. Hence, any model that satisfies θ_i also satisfies θ'_i. This implies that any model that satisfies $(\vee_{i\in[k]}(\theta_i \wedge \neg\varphi) \vee \neg\psi)$ also satisfies $(\vee_{i\in[k]}(\theta'_i \wedge \neg\varphi) \vee \neg\psi)$. Hence, the required result. $\square$

Next, Lemma 5.3.5 allows us to replace the antecedent of each of the resulting implications by an atom (possibly with a different atom for each implication), while still maintaining maximum disclosure.

In both Lemmas 5.3.4 and 5.3.5, we use ψ to represent the attacker's knowledge about the bucketization $\mathcal{B}$. However, it is worthwhile pointing out that neither lemma places any restriction on ψ or on the underlying probability distribution. This makes the results presented here extremely general and powerful because *they characterize the form of background knowledge that maximizes disclosure risk for any form of anonymization and for any additional background knowledge*.

The main idea behind the proof of Lemma 5.3.4 (and also Lemma 5.3.5) can be illustrated as follows. Consider a bucketization $\mathcal{B}$. Let $(t_{p_i}[S] = s_i) \rightarrow (t_{p'_i}[S] = s'_i)$, for $i \in \{0, 1\}$, be two simple implications which maximize the disclosure of $\mathcal{B}$ with respect to $\mathcal{L}^2_{\text{basic}}$. For convenience, we let A_i denote the atom $t_{p_i}[S] = s_i$ and B_i the atom $t_{p'_i}[S] = s'_i$. Let C be the atom $t_p[S] = s$ such that $\Pr(C \mid \mathcal{B} \wedge (\wedge_{i\in[2]}(A_i \rightarrow B_i)))$ is the maximum disclosure.

Table 5.4: Truth Tables

	$\wedge_{i\in[2]}(A_i \to B_i)$						$\wedge_{i\in[2]}(A_i \to C)$					
	A_0	A_1	B_0	B_1	C		A_0	A_1	B_0	B_1	C	
a	0	0	*	*	0	$=$	0	0	*	*	0	a
b	0	0	*	*	1	$=$	0	0	*	*	1	b
c	0	1	*	1	0							
d	0	1	*	1	1	$\subseteq$	0	1	*	*	1	d'
e	1	0	1	*	0							
f	1	0	1	*	1	$\subseteq$	1	0	*	*	1	f'
g	1	1	1	1	0							
h	1	1	1	1	1	$\subseteq$	1	1	*	*	1	h'

Now let us restrict our attention to the set of tables consistent with $\mathcal{B}$. Let $\mathcal{T}_1$ be the set of tables satisfying the simple implications $A_0 \to B_0$ and $A_1 \to B_1$, and let $\mathcal{T}_2$ be the set of tables satisfying $A_0 \to C$ and $A_1 \to C$. Table 5.4 is a diagrammatic representation of $\mathcal{T}_1$ and $\mathcal{T}_2$. Each row in the the truth table on the left (resp., right) in Table 5.4 represents a subset of $\mathcal{T}_1$ (resp., $\mathcal{T}_2$). The variables a, b, c, d, e, f, g, h in the left-most (resp., a, b, d', f', h' in the right-most) column represents the size of the corresponding set. For example, the set of tables represented by the second row is the set of tables that satisfy the atom C but do not satisfy A_0 and A_1, and the number of such of tables is b.

It is now clear from Figure 5.4 that the implications $A_0 \to C$ and $A_1 \to C$ also produce the maximum disclosure as follows. $\Pr(C \mid \wedge_{i\in[2]} A_i \to B_i) = \frac{b+d+f+h}{a+b+c+d+e+f+g+h}$ and $\Pr(C \mid \wedge_{i\in[2]} A_i \to C) = \frac{b+d'+f'+h'}{a+b+d'+f'+h'}$. Also $\frac{b+d+f+h}{a+b+c+d+e+f+h} \leq \frac{b+d+f+h}{a+b+d+f+h} \leq \frac{b+d'+f'+h'}{a+b+d'+f'+h'}$ since $d \leq d'$, $f \leq f'$, and $h \leq h'$. Thus $\Pr(C \mid \wedge_{i\in[2]} A_i \to B_i) \leq \Pr(C \mid \wedge_{i\in[2]} A_i \to C)$.

5.3.3 Computing Maximum Disclosure Efficiently

Having reduced our search space from sets of basic implications that could lead to maximum disclosure to sets of simple implications with the same consequent, we are now in a position to create an efficient algorithm to compute the maximum disclosure. We want to *maximize* $\Pr(A \mid \mathcal{B} \wedge \wedge_{i\in[k]}(A_i \rightarrow A))$ over all atoms A, A_i, $i \in [k]$. Notice that for any atoms A, A_i, $i \in [k]$ such that A and $\wedge_{i\in[k]}A_i \rightarrow A$ are consistent with bucketization $\mathcal{B}$ we have

$$\begin{aligned}
&\Pr(A \mid \mathcal{B} \wedge (\wedge_{i\in[k]}A_i \rightarrow A)) \\
&\quad = \frac{\Pr(A \wedge (\wedge_{i\in[k]}(A_i \rightarrow A)) \mid \mathcal{B})}{\Pr((\wedge_{i\in[k]}(A_i \rightarrow A)) \mid \mathcal{B})} \\
&\quad = \frac{\Pr(A \mid \mathcal{B})}{\Pr((\neg A \wedge (\wedge_{i\in[k]}\neg A_i)) \vee A \mid \mathcal{B})} \\
&\quad = \frac{\Pr(A \mid \mathcal{B})}{\Pr(\neg A \wedge (\wedge_{i\in[k]}\neg A_i) \mid \mathcal{B}) + \Pr(A \mid \mathcal{B})}.
\end{aligned}$$

So it suffices to construct an efficient algorithm to *minimize*, over all atoms A, A_i, $i \in [k]$,

$$\frac{\Pr(\neg A \wedge (\wedge_{i\in[k]}\neg A_i) \mid \mathcal{B})}{\Pr(A \mid \mathcal{B})}. \tag{5.1}$$

In Section 5.3.3, we show how to minimize $\Pr(\wedge_{i\in[k]}\neg A_i \mid \mathcal{B})$ over atoms A_i involving individuals in the same bucket. We use this in Section 5.3.3 to provide a dynamic programming algorithm MINIMIZE1 that minimizes Formula (5.1) over atoms A, A_i, $i \in [k]$ involving individuals in the same bucket. Finally, in Section 5.3.3, we use MINIMIZE1 to construct another dynamic programming algorithm MINIMIZE2 to minimize Formula (5.1) jointly over the entire bucketization.

Minimizing $\Pr(\wedge_{i\in[k]}\neg A_i \mid \mathcal{B})$ for One Bucket

Consider all sets of k atoms involving individuals whose tuples are in a single $b \in \mathcal{B}$. Each set of k atoms is associated with a tuple $(l, k_0, \ldots, k_{l-1})$, where l is the number of individuals involved in the k atoms, and k_i is the number of atoms involving the i-th individual. We label the k atoms $A_{i,j}$ for $i \in [l]$ and $j \in [k_i]$ such that atom $A_{i,j}$ is the j-th atom (out of k_i atoms) involving the i-th individual. Lemma 5.3.7 provides a closed form for the minimum value of $\Pr(\wedge_{i\in[k]}\neg A_i \mid \mathcal{B})$ over all sets of k atoms associated with a particular $(l, k_0, \ldots, k_{l-1})$.

Lemma 5.3.7 *Let $b \in \mathcal{B}$ be any bucket. Let k, l, and $k_0, k_1, \ldots, k_{l-1}$ be such that $k = \Sigma_{i\in[l]} k_i$ and $k_i \geq k_{i+1}$ for all $i \in [l-1]$. Let $s_b^0, s_b^1, s_b^2, \ldots$ be the sensitive values arranged in descending order of frequency in b. Then $\Pr(\wedge_{i\in[l],j\in[k_i]}\neg A_{i,j} \mid \mathcal{B})$ is minimized over all atoms $A_{i,j}$ when, $A_{i,j}$ is $t_{p_i}[S] = s_b^j$, for all $i \in [l]$ and all $j \in [k_i]$, where $p_0, p_1, \ldots, p_{l-1} \in P_b$ are distinct. Consequently, the minimum probability is given by*

$$\prod\nolimits_{i\in[l]} \tfrac{n_b - i - \sum_{j\in[k_i]} n_b(s_b^j)}{n_b - i}. \tag{5.2}$$

Proof When $A_{i,j}$ is $t_{p_i} = s_b^j$ for all $i \in [l]$ and all $j \in [k_i]$, then it is easy to see that

$$\Pr(\wedge_{i\in[l],j\in[k_i]}\neg A_{i,j} \mid \mathcal{B}) = \prod_{i\in[l]} \frac{n_b - i - \sum_{j\in[k_i]} n_b(s_b^j)}{n_b - i}.$$

We now show, by induction on l (i.e., the number of individuals involved in the k atoms), that for all $b \in \mathcal{B}$ and all atoms $A_{i,j}$ (not necessarily $t_i[S] = s_i$) such that $A_{i,j}$ and $A_{i,j'}$ mention the same tuple in b iff $i = i'$, we have

$$\Pr(\wedge_{i\in[l],j\in[k_i]}\neg A_{i,j} \mid \mathcal{B}) \geq \prod_{i\in[l]} \frac{n_b - i - \sum_{j\in[k_i]} n_b(s_b^j)}{n_b - i}.$$

In the base case (i.e., $l = 1$, $k_0 = k$), all the atoms mention the same individual, say p_0. So taking $A_{0,j}$ to be $t_{p_0}[S] = s_b^j$ for $j \in [k]$ clearly minimizes $\Pr(\wedge_{j\in[k]}\neg A_{0,j} \mid \mathcal{B})$, and actually achieves a probability of $\frac{n_b - \Sigma_{j\in[k]} n_b(s_b^j)}{n_b}$.

Suppose that the induction hypothesis holds for all l. Consider the case for $l+1$. Consider p_0, the individual involved in the most (i.e., k_0) atoms (namely, $A_{0,j}$, for $j \in [k_0]$). Let S' be the set of values in S *not* involved in these atoms. That is, $S' = \{s \in S : \forall j \in [k_0] \,.\, A_{0,j} \not\equiv t_{p_0}[S] = s\}$. For each $s \in S'$,

- let $k_i^s = k_{i+1}$, for each $i \in [l]$,
- let $A_{i,j}^s$ be $A_{i+1,j}$ for each $i \in [l]$ and each $j \in [k_i^s]$,
- let b^s and $\mathcal{B}^s$ be the bucket and bucketization, respectively, obtained from b and $\mathcal{B}$ by removing $t_{p_0}[X]$ and one occurrence of s from b.

Then it is easy to see that

- $n_{b^s} = n_b - 1$, and
- $\sum_{j \in [k_i^s]} n_{b^s}(s_{b^s}^j) \leq \sum_{j \in [k_{i+1}]} n_b(s_b^j)$.

So, using these facts and the induction hypothesis, we have

$$\begin{array}{ll}
\multicolumn{2}{l}{\Pr(\wedge_{i \in [l+1], j \in [k_i]} \neg A_{i,j} \mid \mathcal{B})} \\
= & \sum_{s \in S'} \Pr(\wedge_{i \in [l+1], j \in [k_i]} \neg A_{i,j} \mid \mathcal{B} \wedge t_{p_0}[S] = s) \Pr(t_{p_0}[S] = s \mid \mathcal{B}) \\
= & \sum_{s \in S'} \Pr(\wedge_{i \in [l], j \in [k_i^s]} \neg A_{i,j}^s \mid \mathcal{B}^s) \frac{n_b(s)}{n_b} \\
\geq & \sum_{s \in S'} (\prod_{i \in [l]} \frac{n_{b^s} - i - \sum_{j \in [k_i^s]} n_{b^s}(s_{b^s}^j)}{n_{b^s} - i}) \frac{n_b(s)}{n_b} \\
\geq & \sum_{s \in S'} (\prod_{i \in [l]} \frac{n_b - i - 1 - \sum_{j \in [k_{i+1}]} n_b(s_b^j)}{n_b - i - 1}) \frac{n_b(s)}{n_b} \\
= & (\prod_{i \in [l]} \frac{n_b - i - 1 - \sum_{j \in [k_{i+1}]} n_b(s_b^j)}{n_b - i - 1}) \sum_{s \in S'} \frac{n_b(s)}{n_b} \\
\geq & (\prod_{i \in [l]} \frac{n_b - i - 1 - \sum_{j \in [k_{i+1}]} n_b(s_b^j)}{n_b - i - 1}) \frac{n_b - \sum_{j \in [k_0]} n_b(s_b^j)}{n_b} \\
= & \prod_{i \in [l+1]} \frac{n_b - i - \sum_{j \in [k_i]} n_b(s_b^j)}{n_b - i}.
\end{array}$$

This completes the induction. □

Note that $l \leq k$ and $k = \sum_{i \in [l]} k_i$ since each atom involves at exactly one individual. So the question of minimizing $\Pr(\wedge_{i \in [k]} \neg A_i \mid \mathcal{B})$ over all atoms A_i that mention only tuples in b becomes one of minimizing $\prod_{i \in [l]} \frac{n_b - i - \sum_{j \in [k_i]} n_b(s_b^j)}{n_b - i}$ over all $l \leq k$ and all $k_0, \ldots, k_{l-1}$ such that $\sum_{i \in [l]} k_i = k$.

Algorithm 1: Minimize Within Bucket

```
1: procedure MINIMIZE1($b, i, \hat{k}_i, \hat{k}$))
2:     Input: b is the bucket under consideration
3:     Input: $i$ is the index of the next individual $p_i$ for which $k_i$ (i.e., the number of
   atoms involving individual $p_i$) is to be determined (initially $0$)
4:     Input: $\hat{k}_i$ is the the upper bound for $k_i$ (initially $k$)
5:     Input: $\hat{k}$ is the number of atoms for which the individuals involved have yet to
   be been determined (initially $k$)
6:     $p_{\min} \leftarrow 1$
7:     for $k_i = 1, 2, \ldots, \min(\hat{k}_i, \hat{k})$ do
8:         $p \leftarrow$ MINIMIZE1$(b, i+1, k_i, \hat{k} - k_i)$
9:         $p \leftarrow \frac{n_b - i - \sum_{j \in [k_i]} n_b(s_b^j)}{n_b - i} \times p$
10:        $p_{\min} \leftarrow \min(p_{\min}, p)$
11:    end for
12:    return $p_{\min}$
13: end procedure
```

This can easily be done using Algorithm 1. Thus, calling MINIMIZE1$(b, 0, k, k)$ minimizes $\Pr(\wedge_{i \in [k]} \neg A_i \mid \varphi_{\mathcal{B}})$ over all atoms A_i that involve individuals with tuples in bucket b. It is easy to modify the algorithm to remember the minimizing values of $k_0, \ldots, k_{l-1}$, and thus we can even reconstruct the set of minimizing atoms according to Lemma 5.3.7.

Algorithm complexity. Note that the parameters of MINIMIZE1 are bounded. That is, for every recursive call MINIMIZE1$(b, i, k_i, \hat{k})$ that occurs inside the initial call to MINIMIZE1$(b, 0, k, k)$, parameter b does not change, and parameters $i, \hat{k}_i, \hat{k}$ are all bounded by k (i.e., the number of implications we allow the attacker to know). So we can easily turn this into an $O(k^3)$ time and space algorithm using dynamic programming.

Minimizing Formula (5.1) within One Bucket

Let us now minimize $\frac{\Pr(\neg A \wedge (\wedge_{i \in [k]} \neg A_i) | \mathcal{B})}{\Pr(A|\mathcal{B})}$ over all $k+1$ atoms A and A_i, for $i \in [k]$, that only mention tuples in bucket b. Clearly any A, A_i that simultaneously minimize the numerator and maximize the denominator will work. We know that MINIMIZE1$(b, 0, k+1, k+1)$ will minimize the numerator. According to Lemma 5.3.7, at least one of these minimal $k+1$ atoms mention the most frequent sensitive value. So, taking this atom to be A, we maximize the denominator as well. Thus, the minimum value is given by

$$\text{MINIMIZE1}(b, 0, k+1, k+1) \times \frac{n_b}{n_b(s_b^0)}.$$

Minimizing Formula (5.1) over All Buckets

We look again at minimizing $\frac{\Pr(\neg A \wedge (\wedge_{i \in [k]} \neg A_i) | \mathcal{B})}{\Pr(A|\mathcal{B})}$, except this time, we allow A and A_i for $i \in [k]$ to mention tuples in possibly different buckets. To do this, we make use of the independence between buckets. Suppose that the $k+1$ minimizing atoms (including A) are such that k_i of them mention tuples in bucket b_i, for each $i \in [l]$ for some $l \leq k+1$. Let b_j be the bucket containing the tuple mentioned by A. Then, since the permutation of sensitive values for each bucket was picked independently, we can compute the minimum as

$$\frac{n_{b_j}}{n_{b_j}(s^0_{b_j})} \times \prod_{i \in [l]} \textsc{Minimize1}(b_i, 0, k_i, k_i).$$

So we need to minimize the above for all choices of $l \leq k+1$, j, and $k_0, k_1, \ldots, k_{l-1}$ (which we can assume without loss of generality to be in descending order). Assuming buckets in $\mathcal{B}$ are labeled as $b_0, b_1, b_2, \ldots$, this is done by the MINIMIZE2.

So MINIMIZE2$(0, k, \mathit{true})$ minimizes $\frac{\Pr(\neg A \wedge (\wedge_{i\in[k]} \neg A_i)|\mathcal{B})}{\Pr(A|\mathcal{B})}$ over all atoms A, A_i, $i \in [k]$. It is easy to modify the algorithm to remember the i's and h_i's, and hence reconstruct the minimizing atoms.

Algorithm complexity. Note that the parameters of MINIMIZE2 are bounded. That is, for every recursive call to MINIMIZE2(i, h_i, a) that occurs inside the initial call to MINIMIZE2$(0, k, \mathit{true})$, parameter i is bounded by the number of buckets, parameter k_i is bounded by the total number of implications k, and a is either true or false. Thus, assuming that we first memoize (i.e., precompute all possible calls to) MINIMIZE1 (which we can do in time $O(|\mathcal{B}| \times k^3)$), we can modify the MINIMIZE2 algorithm using dynamic programming to take an additional $O(|\mathcal{B}| \times k)$time and space. So the whole algorithm can be made to run in $O(|\mathcal{B}| \times k^3)$time and space.

Incidentally, if two bucketizations $\mathcal{B}$ and $\mathcal{B}^*$ differ only in that $\mathcal{B}^*$ is the result of removing some buckets from $\mathcal{B}$ and adding x new buckets to $\mathcal{B}$, then, after we run the algorithm for $\mathcal{B}$, we memoize MINIMIZE1 for the x new buckets; so the incremental cost of running the algorithm for $\mathcal{B}^*$ is $O(|\mathcal{B}^*| \times k + x \times k^3)$-time. Moreover, if we knew in advance which buckets will be removed, we can order the buckets $b_0, b_1, \ldots$ to reuse the memoization of MINIMIZE2 as well.

Algorithm 2: Minimize Across Buckets

```
1: procedure MINIMIZE2($i, h_i, a$)
2:     Input: $i$ is the current bucket $b_i$ (initially $0$)
3:     Input: $h_i$ is number of atoms $A_j, j \in [k]$ that yet to be determined (initially $k$)
4:     Input: $a$ is a Boolean flag that indicates if atom $A$ involves someone in an earlier
   bucket $b_j$, $j < i$ (initially false)
5:     $r_{min} \leftarrow \infty$
6:     if $i = |\mathcal{B}|$ then
7:         return $r_{min}$
8:     end if
9:     for $h_{i+1} = 0, 1, 2, \ldots, h_i$ do
10:        $u \leftarrow$ MINIMIZE1($b_i, 0, h_{i+1}, h_{i+1}$)
11:        $x \leftarrow$ MINIMIZE2($i + 1, h_i - h_{i+1}, true$)
12:        if $a = false$ then
13:            // Atom $A$ does not involve an earlier bucket $b_j$, $j < i$
14:            $v \leftarrow$ MINIMIZE1($b_i, 0, h_{i+1} + 1, h_{i+1} + 1$)
15:            $r_{\min} \leftarrow \min(r_{\min}, v \times x \times \frac{n_{b_i}}{n_{b_i}(s^0_{b_i})})$
16:            $r_{\min} \leftarrow \min(r_{\min}, u \times$ MINIMIZE2($i + 1, h_i - h_{i+1}, false$))
17:        else
18:            // Atom $A$ involves an earlier bucket $b_j$, $j < i$
19:            $r_{\min} \leftarrow \min(r_{\min}, u \times x)$
20:        end if
21:    end for
22:    return $r_{min}$
23: end procedure
```

5.3.4 Finding a Safe Bucketization

Armed with a method to compute the maximum disclosure, we now show how to efficiently find a "minimally sanitized" bucketization for which maximum disclosure is below a given threshold. Intuitively, we would like a minimal sanitization in order to preserve the utility of the published data. Let us be more concrete about the notion of minimal sanitization. Given a table, consider the set of bucketizations of this table. We impose a partial ordering $\preceq$ on this set of bucketizations where $\mathcal{B} \preceq \mathcal{B}'$ if and only if every bucket in $\mathcal{B}'$ is the union of one of more buckets in $\mathcal{B}$. Thus the bucketization $\mathcal{B}_\top$ that has all the tuples in one bucket is the unique top element of this partial order, and the bucketization $\mathcal{B}_\bot$ that has one tuple per bucket is the unique bottom element of this partial order. Our notion of a "minimally sanitized" bucketization is one that is as low as possible in the partial order (i.e., as close to $\mathcal{B}_\bot$) while still having maximum disclosure lower than a specified threshold.

Definition 5.3.8 ((c, k)-safety) *Given a threshold $c \in [0, 1]$, we say that $\mathcal{B}$ is* a (c, k)-safe bucketization *if the maximum disclosure of $\mathcal{B}$ with respect to $\mathcal{L}^k_{\text{basic}}$ is less than c.*

If the maximum disclosure is *monotonic* with respect to the partial ordering $\preceq$, then finding a $\preceq$-minimal (c, k)-safe bucketization can be done in time logarithmic in the height of the bucketization lattice (which is at most the number of tuples in the table) by doing a binary search. The following theorem says that we do indeed have monotonicity.

Theorem 5.3.9 (Monotonicity) *Let $\mathcal{B}$ and $\mathcal{B}'$ be bucketizations such that $\mathcal{B} \preceq \mathcal{B}'$.*

Then the maximum disclosure of $\mathcal{B}$ is at least as high as the maximum disclosure of $\mathcal{B}'$ with respect to $\mathcal{L}^k_{\text{basic}}$.

Proof Let b_1 and b_2 be two buckets of sizes m and n respectively in bucketization $\mathcal{B}$. Let b be the bucket formed by merging b_1 and b_2 and let $\mathcal{B}'$ be the new bucketization.

To show monotonicity, it is enough to show that the minimum $\Pr(\wedge_{i\in[k]}\neg A_i \mid \mathcal{B}')$ is at least as high as the minimum $\Pr(\wedge_{i\in[k]}\neg A_i \mid \mathcal{B})$, where A_i range over atoms that involve only individuals in b in both cases.

According to Lemma 5.3.7, let $t_{p_i}[S] = s^j_b$ be the atoms that minimize the second probability (for $\mathcal{B}'$), for $i \in [l]$ and $j \in [k_i]$ (where $p_0, \ldots, p_l$ are the individuals involved in $k_0, \ldots, k_l$ atoms, respectively). Then, as in Lemma 5.3.7, the minimum probability is given by

$$\prod_{i\in[l]} \frac{a_i + b_i - i}{m + n - i},$$

where $a_i = n_{b_1} - \sum_{j\in[k_i]} n_{b_1}(s^j_b)$ and $b_i = n_{b_2} - \sum_{j\in[k_i]} n_{b_2}(s^j_b)$.

For each i, we define P_i , c_i, and d_i inductively:

1. $P_0 = 1$, $c_0 = 0$, $d_0 = 0$.
2. If $\frac{a_i-c_i}{m-c_i} \leq \frac{b_i-d_i}{n-d_i}$ then $P_{i+1} = P_i\frac{a_i-c_i}{m-c_i}$ and $c_{i+1} = c_i + 1$ and $d_{i+1} = d_i$.
3. If $\frac{a_i-c_i}{m-c_i} > \frac{b_i-d_i}{n-d_i}$ then $P_{i+1} = P_i\frac{b_i-d_i}{n-d_i}$ and $c_{i+1} = c_i$ and $d_{i+1} = d_i + 1$.

Think of this as choosing atoms $t_{p'_i} = s^j_b$ for $i \in [l], j \in [k_i]$ where p'_i is a new individual in bucket b_1 or b_2 depending on whether $\frac{a_i-c_i}{m-c_i} \leq \frac{b_i-d_i}{n-d_i}$ or not. It is easy to see that $P_l \geq P(\wedge_{i\in[l],j\in[k_i]}\neg t_{p'_i} = s^j_b \mid \mathcal{B})$. Note that, by definition, $c_i + d_i = i$

for all i. So we have $\frac{a_i+b_i-i}{m+n-i} = \frac{a_i-c_i+b_i-d_i}{m-c_i+n-d_i} = \frac{m-c_i}{m-c_i+n-d_i}\frac{a_i-c_i}{m-c_i} + \frac{n_i-d_i}{m-c_i+n-d_i}\frac{b_i-d_i}{n-d_i} \geq \min(\frac{a_i-c_i}{m-c_i}, \frac{b_i-d_i}{n-d_i})$. So at each step i, we get P_{i+1} by multiplying P_i by a factor that is no more than $\frac{a_i+b_i-i}{m+n-i}$. So $\prod_{i\in[l]} \frac{a_i+b_i-i}{m+n-i} \geq P_l \geq P(\wedge_{i\in[l],j\in[k_i]} \neg t_{p'_i} = s_b^j \mid \mathcal{B})$ and so we are done. □

Another approach is to find *all* $\preceq$-minimal (c, k)-safe bucketizations, and return the one that maximizes a specified utility function. The monotonicity property allows us to make use of existing algorithms for efficient itemset mining [7], k-anonymity [13, 54] and ℓ-diversity [57].[4] For example, we can modify the Incognito [54] algorithm, which finds all the $\preceq$-minimal k-anonymous bucketizations, by simply replacing the check for k-anonymity with the check for (c, k)-safety from Section 5.3.3. We can thus find the bucketization that maximizes a given utility function subject to the constraint that the bucketization be (c, k)-safe.

5.4 Maximum Disclosure and ℓ-diversity

We now use our framework to analyze a different restriction on background knowledge and relate this with a privacy condition recently proposed in [57], called ℓ-diversity. This exercise provides further insight into our techniques, while at the same time contributing an essential piece of formal analysis that was missing in [57], namely, proving that recursive (c, ℓ)-diversity is equivalent to $(\frac{c}{c+1}, \ell - 2)$-safety with respect to a simple language expressing sensitive value elimination. This is an important contribution to our understanding of

[4]While these algorithms typically have worst-case exponential running time in the height of the bucketization lattice, they have been shown to run fast in practice.

(c, ℓ)-diversity because it shows that (c, ℓ) diversity protects against $\ell - 2$ pieces of information involving *possibly several different individuals*, rather than the earlier belief that it protects against $\ell - 2$ pieces of information involving *only one individual*.

Before we begin, however, let us quickly recall the definition of recursive (c, ℓ)-diversity.

Definition 5.4.1 (Recursive (c, ℓ)-diversity) *A bucketization $\mathcal{B}$ is said to be* recursive (c, ℓ)-diverse *if for all buckets $b \in \mathcal{B}$,*

$$n_b(s_b^0) \leq c \times (n_b - n_b(s_b^0) - \sum_{i=1}^{\ell-2} n_b(s_b^i)).$$

Intuitively, this definition states that a bucketization is $(c, 2)$-diverse if, for every bucket, the most frequent attribute value of the sensitive attribute appears at most c times as frequently as all the remaining attribute values of the sensitive attribute combined. As argued in [57], it then follows that if an adversary is able to eliminate $k - 2$ values of the sensitive attribute of *one particular individual* in some bucket, the disclosure risk *for that individual* is at most $\frac{c}{c+1}$.

We now show that eliminating the sensitive values for *one particular individual* maximizes disclosure over background knowledge from language $\mathcal{L}_{\text{neg}}$ (defined below), which allows for sensitive value elimination for *possibly several different individuals*. Once again the disclosure maximizing background knowledge has a special structure, namely, that all statements mentioned the same tuple. Our proof uses the techniques from Section 5.3.

Definition 5.4.2 *Let $\mathcal{L}_{\text{neg}}$ be the set of the formulas of the form $\neg A$ where A is an atom.*

Recall that an atom is a formula of the form $t_p[S] = s$. $\mathcal{L}_{\text{neg}}$ thus captures knowledge of the form "Ed does not have the flu".

Theorem 5.4.3 *For any bucketization $\mathcal{B}$, we have*

$$\max_{\text{atoms } A, A_i} \Pr(A \mid \mathcal{B} \wedge (\wedge_{i \in [k]} \neg A_i)) = \max_{b \in \mathcal{B}} \frac{n_b(s_b^0)}{n_b - \sum_{i \in [k]} n_b(s_b^{i+1})}.$$

Proof This follows immediately from independence between the permutations in separate buckets and Lemma 5.4.4 below. □

Lemma 5.4.4 *Consider a bucket $b \in \mathcal{B}$, and let p be any individual with a tuple t_p in b. Then $\Pr(A \mid \mathcal{B} \wedge (\wedge_{i \in [k]} \neg A_i))$ is maximized over all atoms $A, A_0, \ldots, A_{k-1}$ that involve only individuals from b when*

1. *A is $t_p[S] = s_b^0$, and*
2. *A_i is $t_p[S] = s_b^{i+1}$, for $i \in [k]$.*

Moreover, the maximum probability is given by

$$\frac{n_b(s_b^0)}{n_b - \sum_{i \in [k]} n_b(s_b^{i+1})}.$$

Proof First note that when A is the statement $t_p[S] = s_b^0$, and each A_i is the statement $t_{p_i}[S] = s_b^{i+1}$, then it is easy to see that

$$\Pr(A \mid \mathcal{B} \wedge (\wedge_{i \in [k]} \neg A_i)) = \frac{n_b(s_b^0)}{n_b - \sum_{i \in [k]} n_b(s_b^{i+1})}, \tag{5.3}$$

since this is the relative frequency of s_b^0 after $s_b^1, \ldots, s_b^k$ have been eliminated. We now show that no other choice of atoms A, A_i (involving only individuals with tuples in b) gives a higher probability. We proceed by induction on the number of individuals involved in the atoms $A_0, \ldots, A_{k-1}$ to show that

$$\Pr(A \mid \mathcal{B} \wedge (\wedge_{i\in[k]} \neg A_i)) \leq \frac{n_b(s_b^0)}{n_b - \sum_{i\in[k]} n_b(s_b^{i+1})}.$$

In the base case, where all the atoms $A, A_0, \ldots, A_{k-1}$ involve exactly one individual, it is easy to see that the worst case is given by Equation 5.3. Now, using the induction hypothesis, assume that the lemma is true when the atoms involve at most $m-1$ distinct individuals. We will consider the case where $A, A_0, \ldots, A_{k-1}$ involve $m \geq 2$ individuals. Let p be the individual involved in A. Now $A_0, \ldots, A_{k-1}$ involve some other individual $p' \neq p$, since $m \geq 2$. Without loss of generality, $A_0, \ldots, A_{k'-1}$ be the atoms not involving p' and let $A_{k'}, \ldots, A_{k-1}$ be the atoms involving p', for some $k' < k$. For ease of notation, we abbreviate $\wedge_{i\in[k]}\neg A_i$ by κ and $\wedge_{i\in[k']}\neg A_i$ by κ'. Thus our original background knowledge κ is split into two parts. The first part, κ', is the part of our background knowledge not involving p'; the second part, $\wedge_{i\in[k]\setminus[k']}\neg A_i$, is the part of our background knowledge involving only p'. Since $k' < k$, we can apply our induction hypothesis to the statement κ'.

Let S_b be the set of sensitive values that appear in bucket b (i.e., $S_b = \{s \in S : n_b(s) > 0\}$). For each $s \in S_b$, let b^s and $\mathcal{B}^s$ be the bucket and bucketization, respectively, that are obtained by removing the non-sensitive attributes of p' and an occurrence of s from bucket b. Then it is not hard to show that:

1. $n_{b^s} = n_b - 1$,
2. $n_{b^s}(s_{b^s}^0) \leq n_b(s_b^0)$, and
3. $1 + \sum_{i\in[k']} n_{b^s}(s_{b^s}^{i+1}) \leq \sum_{i\in[k]} n_b(s_b^{i+1})$.

So, using the induction hypothesis (in the first inequality below) and the above facts (in the second inequality), we get:

$$\begin{aligned}
\Pr(A \mid \mathcal{B} \wedge \kappa) &= \textstyle\sum_{s \in S_b} \Pr(A \wedge t_{p'}[S] = s \mid \mathcal{B} \wedge \kappa) \\
&= \textstyle\sum_{s \in S_b} \Pr(A \mid \mathcal{B} \wedge \kappa \wedge t_{p'}[S] = s) \\
&\qquad\qquad \times \Pr(t_{p'}[S] = s \mid \mathcal{B} \wedge \kappa) \\
&= \textstyle\sum_{s \in S_b} \Pr(A \mid \mathcal{B}^s \wedge \kappa') \Pr(t_{p'}[S] = s \mid \mathcal{B} \wedge \kappa) \\
&\leq \textstyle\sum_{s \in S_b} \frac{n_{b^s}(s^0_{b^s})}{n_{b^s} - \sum_{i \in [k']} n_{b^s}(s^{i+1}_{b^s})} \Pr(t_{p'}[S] = s \mid \mathcal{B} \wedge \kappa) \\
&\leq \textstyle\sum_{s \in S_b} \frac{n_b(s^0_b)}{n_b - \sum_{i \in [k]} n_b(s^{i+1}_b)} \Pr(t_{p'}[S] = s \mid \mathcal{B} \wedge \kappa) \\
&= \textstyle\frac{n_b(s^0_b)}{n_b - \sum_{i \in [k]} n_b(s^{i+1}_b)} \sum_{s \in S_b} \Pr(t_{p'}[S] = s \mid \mathcal{B} \wedge \kappa) \\
&\leq \textstyle\frac{n_b(s^0_b)}{n_b - \sum_{i \in [k]} n_b(s^{i+1}_b)}.
\end{aligned}$$

This completes the induction. □

5.5 Experiments

In this section, we present a case-study of our framework for worst-case disclosure using the Adult Database from the UCI Machine Learning Repository [9]. We only consider the projection of the Adult Database onto five attributes: Age, Marital Status, Race, Gender and Occupation. The dataset has 45,222 tuples after removing tuples with missing values. We treat Occupation as the sensitive attribute; its domain consists of fourteen values. We use pre-defined generalization hierarchies for the attributes similar to the ones used by LeFevre et al. [54]. Age can be coarsened to six levels (using intervals of size 1, 5, 10, 20, 40, and 100 years), Marital Status can be coarsened to three levels, and Race and Gender can each either be left as is or be completely suppressed. We consider all the possible anonymized tables using those generalizations.

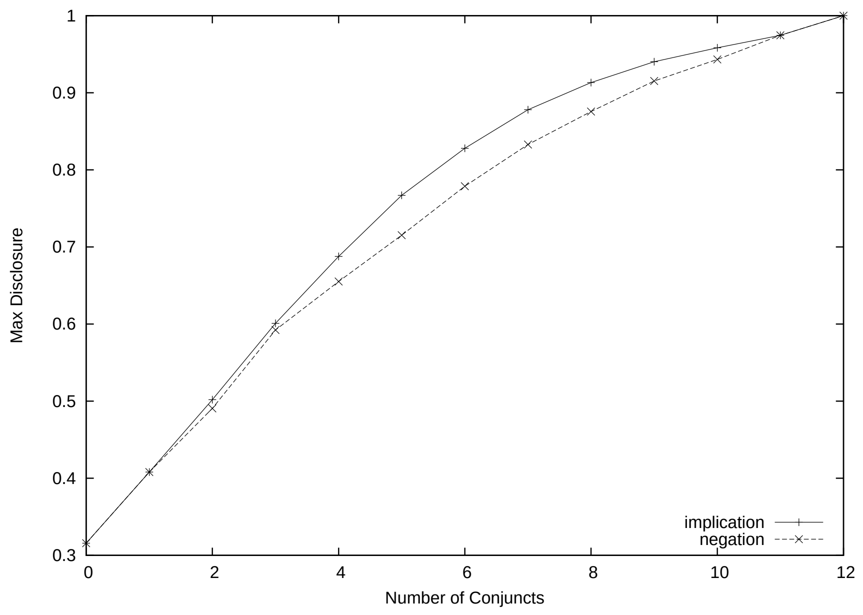

Figure 5.1: Disclosure vs Number of Pieces of Background Knowledge

We computed the maximum disclosure for k pieces of background knowledge, for k ranging from 0 (i.e., no background knowledge) to 12 (since we know that maximum disclosure certainly reaches 1 at $k = 13$ because there are only fourteen possible sensitive values). Figure 5.1 plots, for one anonymized table, the number of pieces of knowledge available to an adversary against the maximum disclosure for both negated atoms (ℓ-diversity) and basic implications. In the anonymized table used, all the attributes other than Age were suppressed and the Age attribute was generalized to intervals of size 20. The solid line corresponds to implication statements and the dotted line corresponds to negated atoms. This graph agrees with our earlier observation that implication-type background knowledge subsumes negation; the maximum disclosure for k negated atoms is always smaller than the maximum disclosure for k implica-

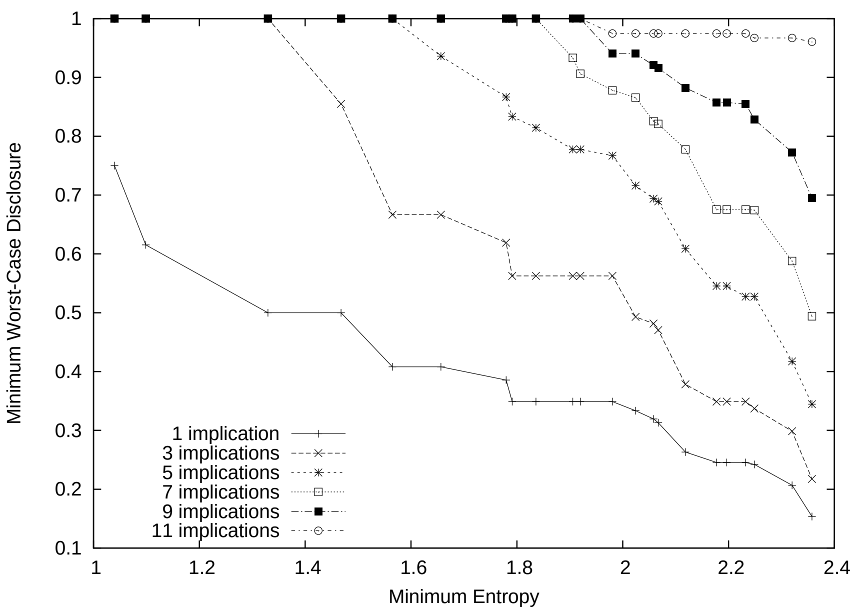

Figure 5.2: Entropy vs Maximum Disclosure Risk

tions. However, note that, for a given k, the difference between the maximum disclosure for negated atoms and for basic implications is not too large. This means that an anonymized table which tolerates maximum disclosure due to k negated atoms need not be anonymized much further to defend against k implications.

Intuitively, if all the buckets in a table have a nearly uniform distribution, then the maximum disclosure should be lower, but the exact relationship is not obvious. To get a better picture, we performed the following experiment. We fixed a value k for the number of pieces of information. For every entropy value h, we looked at the set $\mathcal{T}(h)$ of tables for which the minimum entropy of the sensitive attribute over all buckets was equal to h. Among the tables in $\mathcal{T}(h)$, we found the table $T(h)$ with the least maximum disclosure for k implications.

Let the worst case disclosure for $T(h)$ given k pieces of knowledge be denoted by $w(T(h), k)$. We plotted h versus $w(T(h), k)$ for $k = 1, 3, 5, 7, 9, 11$ in Figure 5.2. We see a behavior which matches our intuition. For a given k, the disclosure risk monotonically decreases with increase in h. This is because increasing h means that we are looking at tables with more and more entropy in their buckets (and, consequently, less skew).

5.6 Related Work

Many metrics have been proposed to quantify privacy guarantees in publishing publishing anonymized data-sets. 'Perfect privacy' [26, 61] guarantees that published data does not disclose any information about the sensitive data. However, checking whether a conjunctive query discloses any information about the answer to another conjunctive query is shown to be very hard (Π_2^p-complete [61]). Subsequent work showed that checking for perfect privacy can be done efficiently for many subclasses of conjunctive queries [56]. Perfect privacy places very strong restrictions on the types of queries that can be answered [61] (in particular, aggregate statistics cannot be published). Less restrictive privacy definitions based on asymptotic conditional probabilities [25] and certain answers [75] have been proposed. Statistical databases allow answering aggregates over sensitive values without disclosing the exact value [1]. De-identification, like k-anonymity [68, 77] and "blending in a crowd" [18], ensures that an individual cannot be associated with a unique tuple in an anonymized table. However, under both of those definitions, sensitive information can be disclosed if groups are homogeneous.

Background knowledge can lead to disclosure of sensitive information. Su et al. [76] and Yang et al. [85] limit disclosure when functional dependencies in the data are known to the data publisher upfront. The notion of ℓ-diversity [57] guards against limited amounts of background knowledge unknown to the data publisher. Farkas et al. [35] provide a survey of indirect data disclosure via inference channels.

There are several approaches to anonymizing a dataset to ensure privacy. These include generalizations [13, 54, 69], cell and tuple suppression [23, 69], adding noise [1, 8, 18, 33], publishing marginals that satisfy a safety range [28], and data swapping [24], where attributes are swapped between tuples so that certain marginal totals are preserved. Queries can be posed online and the answers audited [49] or perturbed [27]. Not all approaches guarantee privacy. For example, spectral techniques can separate much of the noise from the data if the noise is uncorrelated with the data [42, 47]. *Anatomy* [84] is a recently proposed anonymization technique that corresponds exactly to the notion of bucketization that we use in this chapter. When the attacker knows full identification information, then generalization provides no more privacy than bucketization. However, we recommend generalizing the attributes before publishing the data since this will prevent attackers that do not already have full identification information from re-identifying individuals via linking attacks [77]. In many cases, the fact that a particular individual is in the table is considered sensitive information [18].

The utility of data that has been altered to preserve privacy has often been studied for specific future uses of the data. Work has been done on preserving association rules while adding noise [33]; reconstructing distributions of continuous variables after adding noise with a known distribution [6, 8]; reconstructing data clusters after perturbing numeric attributes [18]; and maximizing decision tree accuracy while anonymizing data [44, 83]. There have also been some negative results for utility. Publishing a single k-anonymous table can suffer from the curse of dimensionality [2] - large portions of the data need to be suppressed to ensure privacy. Subsequent work [50] shows how to publish several tables instead of a single one to combat this.

5.7 Conclusions

In this chapter, we initiate a formal study of the worst-case disclosure with background knowledge. Our analysis does not assume that we are aware of the exact background knowledge possessed by the attacker. We assume bounds on the the attacker's background knowledge given in terms of the number of basic units of knowledge that the attacker possesses. We propose basic implications as an expressive choice for these units of knowledge. Although computing the probability of a specific disclosure from a given set of k basic implications is intractable, we show how to efficiently determine the worst-case over all sets of k basic implications. In addition, we show how to search for a bucketization that is robust (to a desired threshold c) against any k basic implications by combining our check for (c, k)-safety with existing lattice-search algorithms. Finally, we demonstrate that, in practice, ℓ-diversity has similar maximum disclosure to our notion of (c, k)-safety, which guards against a richer class of background knowledge.

In the advertising auctions setting, our work has two important limitations.

First, our framework assumes a single categorical sensitive attribute. Therefore, we cannot handle multiple sensitive attributes or set-valued sensitive attributes (e.g., search log data containing the set of phrases that each user has searched for). Second, our framework assumes a one-time data publishing model. In order to handle re-publishing updated data (e.g., when users' interests change from time to time), we would need to take into account the data that has already been published when computing disclosure.

Another issue to consider is our choice of basic units of knowledge. Since we chose basic implications as our units of knowledge, our algorithms will clearly yield very conservative bucketizations if we try to protect against an attacker who knows information that can only be expressed using a large number of basic implications. One way to reduce the number of basic units required is to add more powerful atoms to our existing language. For example, an interesting class of formulas that require a large number of basic implications are those of the form $\vee_{s \in S}(t_p[S] = s \wedge t_{p'}[S] = s)$. Such formulas express equality between the sensitive attributes of two tuples and can be expressed using $|S|$ basic implications. We could therefore try to update our framework to include atoms of the form $t_p[S] = t_{p'}[S]$ in our language and consider basic implications that contain these new types of atoms as well. Finding the right language for basic units of knowledge is an important direction of future work. Other directions for future work include extending our framework for probabilistic background knowledge, studying cost-based disclosure (since it was observed in [57] that not all disclosures are equally bad), and extending our results to other forms of anonymization, such as data-swapping and collections of anonymized marginals [50].

CHAPTER 6

CONCLUSIONS

In this dissertation, we highlight the need for more expressive auctions, and provide a means for advertisers to express dynamic strategies for bidding on clicks, purchases, and slot positions in the form of bidding programs. Given the output of these programs, we provide algorithms to solve the winner-determination problem in time linear in the number of advertisers, reducing the problem size to one that depends on only the number of slots. We extend our techniques to accommodate more complicated scenarios, such as when advertisers can win multiple slots, and when the probability of an advertiser receiving a click depends on the slots assigned to other advertisers. We also identify important cases where we can reduce the work required to evaluate the bidding programs using logical updates. We demonstrate the efficacy of our techniques experimentally. We study the complexity of saving work by optimally sharing aggregation between bidding programs during a single auction, and between the winner-determination computation of multiple auctions. Finally, we provide a framework for revealing useful user information to advertisers while limiting disclosure against any attacker who possesses a specified amount of background knowledge.

There are many interesting opportunities for future work. Identifying other features of bidding programs that could lead to finding the top k programs more efficiently would be very useful. For example, if we are able to statically determine the frequency with which some programs update their bids, then we can avoid running those programs whose bids we know will not be be changing for a while. For the problem of optimally sharing aggregation between bidding programs, we provide complexity results for most of the abstract aggregators that we considered; it would be nice to resolve the complexity of the few remaining non-commutative cases. Furthermore, in the cases that we show to be NP-complete, it is of practical importance to determine whether approximate solutions can be found efficiently. Another practical problem is how to find good initial estimates for the click and purchase probabilities of new ads. When a new ad enters the system, the probability that it will get a click is unknown, and, in order to learn this probability, we need to display it even though it may not lead to an optimal winner assignment. We need a way to occasionally inject new ads into the winner assignment without affecting the long-term optimality of winner determination too much.

Although we have assumed a fairly centralized system in this dissertation, it is important to consider the issues that arise when running bidding programs in a fault-tolerant distributed system. For example, it may be useful to have several copies of a bidding program running on different servers in order to protect against the failure of a single server. This raises questions about what kind of consistency we need to guarantee across these replicas. At the very least, the copies should see a consistent view of certain key variables, such as the amount of daily budget that the advertiser has remaining. Moreover, in a large distributed system, communication cost becomes an important factor to consider. One interesting problem is that of assigning bidding programs to servers spread across the world in such a way that the machines running the bidding programs are located near the geographic area that the advertiser wants to target, in order to minimize communication delay.

Our work on privacy-preserving data publishing can be extended in useful ways as well. For example, consider extending our techniques to handle non-categorical sensitive attributes so as to deal with set-valued attributes (e.g., users' keyword search history), and numerical attributes (e.g., users' click frequencies). This would allow search engines to publish data that is of great utility not just to advertisers, but to researchers and website designers as well. With these new kinds of attributes, we must consider new types of background knowledge, such as statements involving the subset relation (for set-valued sensitive attributes) and comparison relations (for numerical sensitive attributes). Another issue that arises in practice is that user data changes over time, but our framework assumes a one-time data publishing model. Allowing updated data to be re-published in a privacy-preserving manner is a crucial open problem, not just in our setting, but in the publishing of medical and census data as well.

Our work is a first step toward applying database principles to the exciting and important problems arising in advertising auctions. We believe that the database community has much to offer this area given its vast experience with the trade-offs between expressiveness and scalability; providing advertisers with more expressive bidding while retaining the scalability of these auctions is crucial to the continued growth of this multi-billion dollar industry.

BIBLIOGRAPHY

[1] Nabil R. Adam and John C. Wortmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys (CSUR)*, 21(4):515–556, 1989.

[2] Charu C. Aggarwal. On k-anonymity and the curse of dimensionality. In *VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases*, pages 901–909. ACM Press, New York, NY, USA, 2005.

[3] Gagan Aggarwal, Ashish Goel, and Rajeev Motwani. Truthful auctions for pricing search keywords. In *EC '06: Proceedings of the 7th ACM Conference on Electronic Commerce*, pages 1–7. ACM Press, New York, NY, USA, 2006.

[4] Gagan Aggarwal and Jason D. Hartline. Knapsack auctions. In *SODA '06: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm*, pages 1083–1092. ACM Press, New York, NY, USA, 2006.

[5] Gagan Aggarwal, S. Muthukrishnan, and Jon Feldman. Bidding to the top: VCG and equilibria of position-based auctions. In *WAOA '06: Proceedings of the 4th Workshop on Approximation and Online Algorithms*, pages 15–28. Springer, Berlin, Germany; Heidelberg, Germany, September 2006.

[6] Dakshi Agrawal and Charu C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *PODS '01: Proceedings of the 20th ACM Symposium on Principles of Database Systems*, pages 247–255. ACM Press, New York, NY, USA, 2001.

[7] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[8] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 439–450. ACM Press, New York, NY, USA, 2000.

[9] Arthur Asuncion and David J. Newman. UCI machine learning repository, 2007.

[10] Fahiem Bacchus, Adam J. Grove, Joseph Y. Halpern, and Daphne Koller. From statistical knowledge bases to degrees of belief. *Artificial Intelligence*, 87(1-2):75–143, 1996.

[11] Michael Barbaro and Tom Zeller. A face is exposed for AOL searcher no. 4417749. The New York Times, August 2006. Available at `http://select.nytimes.com/gst/abstract.html?res=F10612FC345B0C7A8CDDA10894DE404482`.

[12] Richard A. Bartle. Hearts, clubs, diamonds, spades: Players who suit MUDs. Journal of Virtual Environments, 1996. Available at `http://www.brandeis.edu/pubs/jove/HTML/v1/bartle.html`.

[13] Roberto J. Bayardo and Rakesh Agrawal. Data privacy through optimal k-anonymization. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 217–228. IEEE Computer Society, Washington, DC, USA, 2005.

[14] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[15] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[16] Christian Borgs, Jennifer Chayes, Nicole Immorlica, Kamal Jain, Omid Etesami, and Mohammad Mahdian. Dynamics of bid optimization in online advertisement auctions. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 531–540. ACM Press, New York, NY, USA, 2007.

[17] Christian Borgs, Jennifer Chayes, Nicole Immorlica, Mohammad Mahdian, and Amin Saberi. Multi-unit auctions with budget-constrained bidders. In *EC '05: Proceedings of the 6th ACM Conference on Electronic Commerce*, pages 44–51. ACM Press, New York, NY, USA, 2005.

[18] Shuchi Chawla, Cynthia Dwork, Frank McSherry, Adam Smith, and Hoeteck Wee. Toward privacy in public databases. In *Proceedings of the 2005 Theory of Cryptography Conference*, pages 363–385. Springer, Berlin, Germany; Heidelberg, Germany, 2005.

[19] Miroslav Chlebík and Janka Chlebíková. Approximation hardness of optimization problems in intersection graphs of d-dimensional boxes. In *SODA*

'05: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 267–276. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005.

[20] Vašek Chvátal. On certain polytopes associated with graphs. *Journal on Combinatorial Theory Series B*, 13:138–154, 1975.

[21] Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[22] John Cocke. Global common subexpression elimination. *SIGPLAN Notices*, 5(7):20–24, 1970.

[23] Lawrence H. Cox. Suppression methodology and statistical disclosure control. *Journal of the American Statistical Association*, 75(370):377–385, 1980.

[24] Tore Dalenius and Steven P. Reiss. Data swapping: a technique for disclosure control. *Journal of Statistical Planning and Inference*, 6:191–194, 1982.

[25] Nilesh N. Dalvi, Gerome Miklau, and Dan Suciu. Asymptotic conditional probabilities for conjunctive queries. In *ICDT '05: Proceedings of the 10th International Conference on Database Theory*, pages 289–305. Springer, Berlin, Germany; Heidelberg, Germany, 2005.

[26] Alin Deutsch and Yannis Papakonstantinou. Privacy in database publishing. In *ICDT '05: Proceedings of the 10th International Conference on Database Theory*, pages 230–245. Springer, Berlin, Germany; Heidelberg, Germany, 2005.

[27] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *PODS '03: Proceedings of the 22nd ACM Symposium on Principles of Database Systems*, pages 202–210. ACM Press, New York, NY, USA, 2003.

[28] Adrian Dobra. *Statistical tools for disclosure limitation in multiway contingency tables*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2002.

[29] Alin Dobra, Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Sketch-based multi-query processing over data streams. In *EDBT '04: Proceedings of the 9th International Conference on Extending Database Technology*, pages 551–568. Springer, Berlin, Germany; Heidelberg, Germany, 2004.

[30] Benjamin G. Edelman, Michael Ostrovsky, and Michael Schwarz. Internet

advertising and the generalized second price auction: Selling billions of dollars worth of keywords. NBER Working Paper No. W11765, November 2005.

[31] eMarketer. The unstoppable surge of search advertising. `http://www.emarketer.com/Article.aspx?1004811`, April 2007.

[32] S. Even, Amir Pnueli, and A. Lempel. Permutation graphs and transitive graphs. *Journal of the ACM*, 19(3):400–410, 1972.

[33] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS '03: Proceedings of the 22nd ACM Symposium on Principles of Database Systems*, pages 211–222. ACM Press, New York, NY, USA, 2003.

[34] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS '01: Proceedings of the 20th ACM Symposium on Principles of Database Systems*, pages 102–113. ACM Press, New York, NY, USA, 2001.

[35] Csilla Farkas and Sushil Jajodia. The inference problem: a survey. *ACM SIGKDD Explorations Newsletter*, 4(2):6–11, 2002.

[36] Morteza Fayyazi, David Kaeli, and Waleed Meleis. An adjustable linear time parallel algorithm for maximum weight bipartite matching. *Information Processing Letters*, 97(5):186–190, 2006.

[37] Andras Frank. Some polynomial algorithms for certain graphs and hypergraphs. In *Proceedings of the 5th British Combinatorial Conference*, pages 211–226. Utilitas Mathematica Publishing, Inc., Winnipeg, Manitoba, 1976.

[38] Kris Graft. Study: In-game ads actually work. Next Generation, July 2007. Available at `http://www.next-gen.biz/index.php?option=com_content&task=view&id=6533&Itemid=2`.

[39] Theodore Groves. Incentive in teams. *Econometrica*, 41:617–631, 1973.

[40] Udaiprakash I. Gupta, Der-Tsai Lee, and Joseph Y.-T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12(4):459–467, 1982.

[41] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[42] Zhengli Huang, Wenliang Du, and Biao Chen. Deriving private information from randomized data. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 37–48. ACM Press, New York, NY, USA, 2005.

[43] Ryan Huebsch, Minos Garofalakis, Joseph M. Hellerstein, and Ion Stoica. Sharing aggregate computation for distributed queries. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 485–496. ACM Press, New York, NY, USA, 2007.

[44] Vijay S. Iyengar. Transforming data to satisfy privacy constraints. In *KDD '02: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 279–288. ACM Press, New York, NY, USA, 2002.

[45] David S. Johnson. Approximation algorithms for combinatorial problems. In *STOC '73: Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 38–49. ACM Press, New York, NY, USA, 1973.

[46] Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 1992.

[47] Hillol Kargupta, Souptik Datta, Qi Wang, and Krishnamoorthy Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *ICDM '03: Proceedings of the 3rd IEEE International Conference on Data Mining*, page 99. IEEE Computer Society, Washington, DC, USA, 2003.

[48] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, NY, USA, 1972.

[49] Krishnaram Kenthapadi, Nina Mishra, and Kobbi Nissim. Simulatable auditing. In *PODS '05: Proceedings of the 24th ACM Symposium on Principles of Database Systems*, pages 118–127. ACM Press, New York, NY, USA, 2005.

[50] Daniel Kifer and Johannes Gehrke. Injecting utility into anonymized datasets. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 217–228. ACM Press, New York, NY, USA, 2006.

[51] Brendan Kitts and Benjamin LeBlanc. Optimal bidding on keyword auctions. *Electronic Markets*, 14(3):186–201, 2004.

[52] Sailesh Krishnamurthy, Chung Wu, and Michael Franklin. On-the-fly sharing for streamed aggregation. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 623–634. ACM Press, New York, NY, USA, 2006.

[53] Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics*, 2:83–97, 1955.

[54] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: efficient full-domain k-anonymity. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 49–60. ACM Press, New York, NY, USA, 2005.

[55] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994.

[56] Ashwin Machanavajjhala and Johannes Gehrke. On the efficiency of checking perfect privacy. In *PODS '06: Proceedings of the 25th ACM Symposium on Principles of Database Systems*, pages 163–172. ACM Press, New York, NY, USA, 2006.

[57] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, USA, 2006.

[58] David J. Martin, Johannes Gehrke, and Joseph Y. Halpern. Toward expressive and scalable sponsored search auctions. In *ICDE '08: Proceedings of the 24th IEEE International Conference on Data Engineering*, pages 237–246. IEEE Computer Society, Washington, DC, USA, 2008.

[59] David J. Martin, Daniel Kifer, Ashwin Machanavajjhala, Johannes Gehrke, and Joseph Y. Halpern. Worst-case background knowledge in privacy. Technical report, Cornell University, 2006.

[60] David J. Martin, Daniel Kifer, Ashwin Machanavajjhala, Johannes Gehrke, and Joseph Y. Halpern. Worst-case background knowledge for privacy-preserving data publishing. In *ICDE '07: Proceedings of the 23rd IEEE International Conference on Data Engineering*, pages 126–135. IEEE Computer Society, Washington, DC, USA, April 2007.

[61] Gerome Miklau and Dan Suciu. A formal analysis of information disclosure in data exchange. *Journal of Computer and System Sciences*, 73(3):507–534, 2007.

[62] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society of Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[63] S. Muthukrishnan, Martin Pál, and Zoya Svitkina. Stochastic models for budget optimization in search-based advertising. In *Internet and Network Economics*, volume 4858 of *Lecture Notes in Computer Science*, pages 131–142. Springer, Berlin, Germany; Heidelberg, Germany, 2007.

[64] Alantha Newman. The maximum acyclic subgraph problem and degree-3 graphs. In *APPROX '01/RANDOM '01: Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 5th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 147–158. Springer-Verlag, Berlin, Germany; Heidelberg, Germany; London, UK; etc., 2001.

[65] Nielsen/NetRatings. Interactive advertising bureau (IAB) search branding study. Commissioned by the IAB Search Engine Committee, August 2004. Available at `http://www.iab.net/resources/iab_searchbrand.asp`.

[66] Filip Radlinski, Andrei Broder, Peter Ciccolo, Evgeniy Gabrilovich, Vanja Josifovski, and Lance Riedel. Optimizing relevance and revenue in ad search: A query substitution approach. In *SIGIR '08: Proceedings of the Conference on Research and Development in Information Retrieval*, page (to appear). ACM Press, New York, NY, USA, 2008.

[67] Christopher Ré, Nilesh Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE '07: Proceedings of the 23rd IEEE International Conference on Data Engineering*, pages 886–895. IEEE Computer Society, Washington, DC, USA, 2007.

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

[69] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generaliza-

tion and suppression. Technical report, Carnegie Mellon University, SRI, 1998.

[70] Andrew Sears, Julie A. Jacko, and Michael S. Borella. Internet delay effects: how users perceive quality, organization, and ease of use of information. In *CHI '97: CHI '97 extended abstracts on Human Factors in Computing Systems*, pages 353–354. ACM Press, New York, NY, USA, 1997.

[71] Timos K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13(1):23–52, 1988.

[72] Mike Shields. In-game ads could reach $2 bil. Adweek, April 2006. Available at `http://www.adweek.com/aw/national/article_display.jsp?vnu_content_id=1002343563`.

[73] Adam Silberstein and Jun Yang. Many-to-many aggregation for sensor networks. In *ICDE '07: Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering*, pages 986–995. IEEE Computer Society, Washington, DC, USA, 2007.

[74] Jeremy P. Spinrad. *Efficient Graph Representations*, volume 19 of *Field Institute Monographs*. American Mathematical Society, Providence, RI, USA, 2003.

[75] Kilian Stoffel and Thomas Studer. Provable data privacy. In Kim Viborg Andersen, John K. Debenham, and Roland Wagner, editors, *DEXA '05: Proceedings of 16th International Conference on Database and Expert Systems Applications*, volume 3588 of *Lecture Notes in Computer Science*, pages 324–332. Springer, Berlin, Germany; Heidelberg, Germany, 2005.

[76] Tzong-An Su and Gultekin Ozsoyoglu. Controlling FD and MVD inferences in multilevel relational database systems. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):474–485, 1991.

[77] Latanya Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[78] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. In *SIGGRAPH '91: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, pages 61–70. ACM Press, New York, NY, USA, 1991.

[79] Niki Trigoni, Yong Yao, Alan J. Demers, Johannes Gehrke, and Rajmohan Rajaraman. Multi-query optimization for sensor networks. In *DCOSS '05: Proceedings of the 2005 International Conference on Distributed Computing in Sensor Systems*, pages 307–321. Springer, Berlin, Germany; Heidelberg, Germany, 2005.

[80] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[81] Hal R. Varian. Position auctions. UC Berkeley Working Paper, 2006.

[82] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[83] Ke Wang, Benjamin C. M. Fung, and Philip S. Yu. Template-based privacy preservation in classification problems. In *ICDM '05: Proceedings of the 5th IEEE International Conference on Data Mining*, pages 466–473. IEEE Computer Society, Washington, DC, USA, 2005.

[84] Xiaokui Xiao and Yufei Tao. Anatomy: simple and effective privacy preservation. In *VLDB '06: Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 139–150. ACM Press, New York, NY, USA, 2006.

[85] Xiaochun Yang and Chen Li. Secure XML publishing without information leakage in the presence of data inference. In *VLDB '04: Proceedings of the 30th International Conference on Very Large Data Bases*, pages 96–107. ACM Press, New York, NY, USA, 2004.

[86] Rui Zhang, Nick Koudas, Beng Chin Ooi, and Divesh Srivastava. Multiple aggregations over data streams. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 299–310. ACM Press, New York, NY, USA, 2005.