OPTIMIZATION PROBLEMS IN DATA MINING

BY

CINDA HEEREN

B.S., Stanford University, 1990
M.S., Stanford University, 1990

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

Abstract

One natural, yet unusual, source of data is the set of queries that are performed on a database. We consider such queries to be reflective of data access patterns and we use them to create indices on the data that are likely to be useful in minimizing the cost of answering future queries. We formalize the problem of finding these optimal indices under a constraint on the total amount of space available for storing them, we give strong negative and positive performance bounds, and we quantify the error in performance introduced by running the algorithm on a sample drawn from an unknown query distribution.

We investigate the problem of finding optimized support association rules for a single numerical attribute, where the optimized region is a union of k disjoint intervals from the range of the attribute. We give the first polynomial time algorithm for the problem of finding such a region maximizing support and meeting a cumulative confidence threshold. Experiments demonstrate that the best algorithm for a more constrained version of the problem has performance degradation on both synthetic and real world data. We prove theoretical bounds on sufficient sample size to achieve a given performance level, and we validate convergence on synthetic and real-world data experimentally. We propose a natural greedy algorithm, and analyze its performance.

We introduce a novel type of rule, wherein claims of the form "our object ranked r or better in x of the last t time units," are formalized, and where maximal claims of this form are defined under two natural partial orders. For the first, we give an efficient and optimal algorithm for finding all such claims. For the second, we give an algorithm whose running time is significantly more efficient than that of a naïve one. Finally, we connect this boasting

problem to that of finding a sequence of optimized confidence association rules, and give an efficient algorithm for solving a simplification of the problem.

To Mike.

Acknowledgments

Thank you to H.V. Jagadish, for posing the index selection problem, and for his patience and generosity.

Thank you to Jaiwei Han, for his friendliness, encouragement, interest, and patience.

Heartfelt thanks to Sariel Har-Peled, for suggesting that the dynamic convex hull data structure would apply to the solution of the wine problem, but more importantly, for challenging me to do better, and then for kindly supporting me in my effort to do so.

Deepest thanks to my advisor, Lenny Pitt, for keeping me organized and focussed, for being an unparalleled teacher and an even better friend, and for making work a joy.

Thank you to Mom, for teaching me to love books, thank you to Dad, for teaching me to love thinking about math, and thank you to both, for modeling a life of curiosity and optimism and affection.

Thank you to Isaak and Cory, for all the laughter in our house.

Finally, thanks to Mike, for the lovely life he has created for us.

This research was supported in part by NSF grant IIS-9907483.

Table of Contents

Chapter 1

Introduction

The explosion of ubiquitous computing, electronic communication, embedded chips, and sensors, together with low cost storage, make data collection on a massive scale both alluring and compulsory. The magnitude and variety of data provide opportunity for more informed decision making, more adaptable software, and more effective modeling tools, but they also force the formidable challenge of doing so in a computationally efficient manner. Innovative techniques for gleaning information from such vast amounts of data require careful thought about the types of patterns that might be useful, how they can be used, and whether or not the helpful patterns can be discovered efficiently. Our work has two main foci within this arena: the first on partial index construction/selection for large databases, and the second on the search for patterns within numerical attributes of a database.

One natural and yet somehow unusual source of data is the set of queries that are performed on a database. We consider such queries to be reflective of data access patterns and we thus use them to create indices on the data that are likely to be useful in minimizing the cost of answering future queries. We formalize the problem of finding these optimal indices under a constraint on the total amount of space available for storing them, we give strong negative and positive performance bounds, and we quantify the error in performance introduced by running the algorithm on a sample drawn from an unknown query distribution.

The search for association rules is the search for sets of attributes in a database that are predictive of some Boolean characteristic of the data. A useful association rule is one that is

supported by a significant portion of the database, and that is sufficiently predictive of the Boolean characteristic. Optimized association rules, on the other hand, assume the attributes of interest are given, and it is the specific instantiations of these given attributes that must be predictive of the Boolean characteristic. Our results extend the body of knowledge on optimized association rules in two ways.

We investigate the problem of finding optimized support association rules for a single numerical attribute, where the optimized region is a union of k disjoint intervals from the range of the attribute. We give the first polynomial time algorithm for the problem of finding such a region maximizing support and meeting a minimum *cumulative* confidence threshold. Because the algorithm is not practical, we consider an ostensibly easier, more constrained version of the problem. Our experiments demonstrate that the best extant algorithm for the constrained version has significant performance degradation on both a synthetic model of patterned data and on real world data sets. We propose running the algorithm on a small random sample as a means of obtaining near optimal results with high probability. We prove theoretical bounds on sufficient sample size to achieve a given performance level, and we validate rapid convergence on synthetic and real-world data experimentally. Finally, we propose a natural greedy algorithm, and we prove tight performance of the approach.

We introduce a novel type of rule, a "boast," wherein claims of the form "our object ranked r or better in x of the last t time units," are formalized, and where maximal claims of this form are defined under two natural and different partial orders. For the first partial order we give an efficient and optimal algorithm for finding all such claims. For the second partial order we give an algorithm whose running time is significantly more efficient than that of a naïve one. Finally, we connect this boasting problem to that of finding a sequence of optimized confidence association rules, and give an efficient algorithm for solving a simplification of the sequence of optimized rules problem.

Many problems in data mining, such as association rule discovery, for example, involve the enumeration of useful patterns found in the data, and often the goal is to select all the

sets that satisfy some collection of constraints. In other problems, the goal is to make an optimal choice among candidate patterns. The problems we solve are all of this second type.

Our work is a veritable survey of optimization and analysis techniques from theoretical computer science.

- We use randomization and linear programming relaxation techniques to select a set of indices to build on a database. [Chapter 2]
- We use NP-hardness and inapproximability reductions to demonstrate the difficulty of the general index selection problem. [Chapter 2]
- We use approaches from learning theory, Chernoff bounds and the VC-dimension, to determine sample sizes sufficient to guarantee the performance of our algorithms on an unknown probability distribution. [Chapters 2 and 3]
- We use a greedy algorithm and analysis to obtain approximation guarantees for the k-interval optimized support association rule problem, where the confidence level of each interval must independently exceed some minimum value. [Chapter 3]
- We use dynamic programming to give a polynomial time algorithm for finding the k-interval optimized support association rule, whose the k-interval must cumulatively meet the minimum confidence bound. [Chapter 3]
- We use methods for the dynamic maintenance of convex hulls from computational geometry to find optimal boasting rules. [Chapter 4]
- We define partial orders and efficiently search them to find a maximal set of boasting rules. [Chapter 4]

The remainder of this thesis is arranged as follows: In Chapter 2 we formalize, give performance bounds on, and give sampling results for the index selection problem. In Chapter 3

we address the k-interval, optimized support association rule problem, and provide experimental results, sampling bounds, and a greedy algorithm. Chapter 4 defines and solves in two different ways, the maximal boasting problem. Finally, Chapter 5 gives a concise statement of all significant results in the thesis.

Chapter 2

Optimal Index Selection

Indices are frequently used to speed up query evaluation in data management systems. Any given index can be of value only with respect to certain data accesses, and not with respect to others. With limited space resources, the question arises which indices to build? What is the best use of space for indices to minimize the average execution time of a query?

We address these problems assuming a general data model where data is composed of indexable units (records) without any implied structure. Indices can be constructed on arbitrary predicates, not just predicates of the form $attr = val$, and return identifiers of the indexable units that satisfy the index predicate. Our query model is (slightly more general, as described in the next section, than) a conjunctive selection query. A typical evaluation plan for selection queries involving conjunctions is to use multiple indices to retrieve identifiers for items satisfying one or more of the predicates, and intersect the resulting sets to obtain a smaller set of items that can then be tested for satisfaction of the remaining predicates. An upper bound on the cost of evaluating a query is thus proportional to the cardinality of the smallest indexed predicate (i.e., the most selective index that matches the query). We take this upper bound as a simplified cost model.

Let opt_B be the least average cost of evaluating a query chosen from the workload, using space B to construct indices for M indexable units in the database. We establish the following:

- Provided that the workload of m distinct queries is known (or accurately characterized), we give an algorithm that selects indices using total space $O(B \ln m)$, with expected per-query cost at most opt_B. In other words, a log factor relaxation in the space used for indexing permits the optimum cost to be achieved. The solution uses a novel randomized rounding technique applied to an integer programming formulation.

- Even when the query workload is not known, by employing an additional constraint N on the number of distinct indices that may be built, we can obtain an expected query cost at most ϵM above optimal, finding $O(N \ln m)$ indices using space $O(B \ln m)$, when we are allowed to observe $m = \text{ poly}(1/\epsilon, B, N)$ random queries drawn from the unknown query distribution.

- The logarithmic space relaxation is necessary: If M is the size of the dataset, no polynomial time algorithm can guarantee average query cost less than $M^{1-\epsilon} opt_B$ using space αB, for any constants $\alpha, \epsilon > 0$, unless $NP \subseteq n^{O(\log \log n)}$, which would give nearly polynomial-time algorithms for NP-hard problems - a breakthrough in complexity theory. Since an average query cost of M is trivially obtained, this shows that there is no practical way to obtain a cost anywhere close to optimum even with a constant factor relaxation in space.

The problem formulation we have used is sufficiently general that our results have applicability in a wide variety of settings. We can cast in our framework index selection in many different systems, including multi-attribute indices on tables in a data warehouse, path indices in XML and OO databases, sub-string indices in (biological) sequence databases, and inverted term indices in document information retrieval. Our results also apply to materialized views and predicate caching. See Section 2.4.

Some of the results reported in this chapter are based on joint work with H.V. Jagadish and L. Pitt, and appear in preliminary form in [18].

2.1 Problem Definition

Assume a data set, D, comprising M distinct data items. Each data item may be a record, a document, an XML element or an object, among other things. Let P be a possibly infinite universe of predicates on D. A query is just an element of $p \in P$, and the answer is the set of all data $d \in D$ for which $p(d) = 1$. Note that there is no requirement that the predicate be evaluated against a single table in a relational database (or even, for that matter, that there is a table structure to the data at all). A join query is expressed in our model as a predicate that will be satisfied by a subset of an appropriate cartesian product of tables in the database. (Real queries against a database system typically will do something with the data retrieved from this selection, such as compute an aggregate function. We will argue below that the simplified model above is sufficient to obtain reasonable estimates of query evaluation time.)

Given a set of predicates P on dataset D, there is an induced partial order $(P, \leq_{P,D})$ henceforth denoted without the subscripts, and defined by: $p_1 \leq p_2$ if and only if for all $d \in D$, $p_1(d) \rightarrow p_2(d)$. Thus, any datum satisfying query p_1 would also satisfy query p_2. Note that there is no requirement this partial order be derived from the semantics of the predicate specifications – it could just as well arise due to (known) characteristics of the data. For concreteness, the reader may find it useful to think of each predicate as a conjunction of atomic predicates, even though we do not impose any structure on the predicate specification. The partial order is then obtained by considering subsets of these atomic predicates. For example, if each atomic predicate is an attribute-value pair, we can say: $((x.A = a_1) \wedge (x.B = b_2)) \leq (x.B = b_2)$.

Query response can be facilitated by considering the use of any predicate $p \in P$ as a possible (partial) index, referencing all and only data in D that satisfy p. Below, when we refer to some element in P, we may mean either the corresponding query, or the partial

index built to reference all data that satisfies the predicate. The meaning will be clear from context, though we may use the terms query and index for emphasis.

Often, when the term "index" is used in the context of a database, by it is meant a complete index, with respect to some class of predicates. For instance a B-tree index on a specified attribute A, identifies the database records that satisfy the predicate (value of attribute A equals v) *for each value v of A*. In our terminology, such a B-tree index is a collection of multiple indices, one for each value v of the attribute A. While this notion of choosing only some values of an attribute to index may appear strange in a relational context, such choices are readily made in other contexts: In information retrieval, a set of "index terms" is explicitly chosen, and is frequently not the same as the set of all "words" appearing in the document set. Even in a relational setting, such partial indices may be useful, for example, to reference (perhaps a subset of) records returned by a commonly occuring join. Section 2.4 contains additional discussion.

In an ideal world, we would have an index prebuilt that references exactly the data requested for every possible query. However, each index p built will require space proportional to the amount of data that it references. Let $s_D(p) \in [0, M]$ denote the space required to store the necessary pointers to the data satisfying query p. In other words, the space cost for building index p. This space is proportional to the number of records in D satisfying p. If R is a set of indices, define the total space used by R as $s_D(R) = \sum_{p \in R} s_D(p)$.

Typically we do not have the space to store the answers to all possible queries. Nonetheless, we can still facilitate evaluation of query q if we have an index p such that $q \leq p$ (in which case we'll say that p *covers* q). Given a collection of indices $\{p_1, p_2, \ldots p_k\}$, if none of them covers a given query q, then the only evaluation plan for the query requires a scan of the entire data set at a cost proportional to $M = |D|$, the cardinality of the data set. When one or more indices p_j covers q, multiple evaluation plans are possible: we may choose to consult zero or more of these indices, intersect the sets of identifiers returned, access the data items in this intersection, and then check for satisfaction of the remainder of predicate q. To

make matters concrete for our analysis, we restrict ourselves to the consideration of a single standard evaluation plan that consults only one index of minimum cardinality, the smallest p_j covering q. All data items identified by this index are retrieved, checked for satisfaction of predicate q, and then returned after additional processing, such as grouping and aggregate computation, if any. The time required to process query q is then proportional to the size of the set of data items identified by the index, and hence retrieved and processed. (There will be some time required to find the index with smallest cardinality that covers q from among the indices available, but this is not a function of the size of the data set, and hence can be ignored as comparatively small. Similarly, the cost of any post-processing steps beyond retrieving the data satisfying q is ignored as likely to be small.) Ignoring constant factors, define:

$$c_D(q,p) = \begin{cases} s_D(p) & \text{if } q \leq p, \\ M & \text{otherwise,} \end{cases}$$

The cost of answering a query q relative to index set R is (proportional to)

$$c_D(q,R) = \min_{p \in R} c_D(q,p),$$

the minimum cost among all indices in R that cover q. Note that if there are no covering indices, then the cost will be M by the definition of $c_D(q,p)$.

With limited available space, there arises a natural tradeoff between finding indices for small subcollections of the data (using little space and providing quick query response) and finding indices that are useful for (i.e., cover) many queries. In this paper we consider the extent to which one can make optimal choices of indices under just such constraints:

Definition 1. Let D be a (finite) set of data of cardinality M. Let P be a (possibly infinite) set of polynomially-computable predicates. Given a finite set $Q \subseteq P$ of queries of cardinality m, a finite candidate set $C \subseteq P$ of indices and a total index storage bound B, find a set

$R \subseteq C$ with $s_D(R) \leq B$ minimizing the average per-query cost

$$\frac{1}{m} \sum_{i \in Q} c_D(i, R).$$

There is nothing that prevents C from being the same as P, if the latter is finite. However, it is often possible to define a set C substantially smaller than P to advantage. In many settings, queries are sufficiently structured so as to admit an efficient derivation of a polynomially-sized collection of *all* possible covering "subqueries" $\{q_1, q_2, \ldots, q_{p(n)}\}$ as possible indices for the query. Taking the union of these candidate sets over all workload queries q provides a rich and still relatively small candidate set C guaranteed to contain an optimal subcollection.

For example, in the case of a database of DNA sequences where queries correspond to substrings, since any query string of length n contains $O(n^2)$ subtrings, there are only polynomially many indices that might be built that could cover the query string. Similarly, when queries are constant sized conjunctions of attributes, there are only (an exponentially larger) constant number of possible subsets of attributes that could be useful in covering the query. When the data consists of XML documents and queries are relatively small constant-sized subtree patterns, we can efficiently enumerate all subtrees of each query. Our results apply when queries are arbitrary paths, since these are efficiently enumerable for each document. For the problem of materializing views on the datacube, the set of views considered for materialization is just the set of all vertices of the cube, and can be chosen without even reference to Q.

In other settings where the number of possible covering "subqueries" is too large to find a candidate set C containing some optimal subcollection, heuristics can be used to find the most promising covering subqueries. (For example, for databases of genetic sequences, the search for strings with small edit distance from each string in a large subcollection of the data have proved useful.) Of course, in this case our optimality guarantees are only as good

as the heuristics used. The general problem of finding a complete candidate set C, or a "best" set, is related to difficult problems in both learning theory (find conjunctions that cover many positive examples), datamining (enumerate maximal frequent itemsets), and information retrieval (search for useful words or phrases on which to build inverted indices).

2.2 The problem is hard

Our problem is closely related to the view selection problem for the datacube [16, 14, 20, 37]. Hence, it is not surprising that a strong non-approximability result similar to the one given by Karloff and Mihail [20] holds:

Theorem 2. Let opt_B be the smallest average cost per query attainable by any set of indices $R \subseteq C$ using space at most B on data set D of cardinality M. Then:

- If for some $\epsilon > 0$ there exists a polynomial-time algorithm that can find a set R of space at most B and with average query cost at most $M^{1-\epsilon} opt_B$, then $P = NP$.
- If for some $\epsilon, \alpha > 0$ there exists a polynomial time algorithm that can find a set R of space at most αB and with average query cost at most $M^{1-\epsilon} opt_B$, then $NP \subseteq n^{O(\log \log n)}$.

The theorem can be proved by various modifications to the proof in [20], though a simpler direct proof is possible.

The theorem is particularly ominous-sounding: The first part says that not only is optimal not achievable unless $P = NP$, but it is unlikely that any approximation algorithm can find a set R whose performance is slightly better than that given by the trivial solution of scanning the entire dataset of size M for each query. The second part says, under a slightly stronger assumption (that NP languages are not in "almost" polynomial time), that even if we relax the constraints to allow the index set found to use a constant factor more space, then as in the first part, the performance when compared to the optimal using only space B remains

very poor. As we now show, an only slightly more generous space bound allows for optimal performance when compared to opt_B.

2.3 An optimal solution using extra space

We formulate the problem as a 0-1 integer program, consider its relaxation to a linear program, and then use a novel randomized rounding technique to find a collection of indices R using space $O(B \ln m)$, and with expected query cost opt_B. A similar result will hold if we want to bound the cardinality of R, or some linear combination of the cardinality and the space used. We realize the LP approach is expensive in practice, but we have preliminary results suggesting a deterministic non-LP algorithm with comparable bounds and performance.

Lin and Vitter [25] use similar techniques (which could be applied in this setting) to solve clustering and related problems, achieving a performance guarantee of $(1+\epsilon)opt_B$ with a relaxed resource factor of $(1+\frac{1}{\epsilon})\ln m$.

Given query set Q of size m, candidate index set C, storage bound B, index costs $s_D(j)$ for each $j \in C$, and query processing costs $c_D(i,j)$ for each $i \in Q$ and $j \in C$, consider the 0-1 integer program with decision variables $x_{ij}, j \in C, i \in Q$, and $y_j, j \in C$ with the following intended meanings:

$$x_{ij} = \begin{cases} 1 & \text{if query } i \text{ should be answered by} \\ & \text{selecting index } j, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if index } j \text{ should be selected for} \\ & \text{use in answering some query } i, \\ 0 & \text{otherwise.} \end{cases}$$

The integer program is:

$$\min Z = \sum_{i \in Q} \sum_{j \in C} x_{ij} c_D(i,j) \tag{2.3.1}$$

$$\text{subject to} \tag{2.3.2}$$

$$\sum_{j \in C} x_{ij} = 1, \forall i \in Q \tag{2.3.3}$$

$$y_j \geq x_{ij}, \forall j \in C, i \in Q \tag{2.3.4}$$

$$\sum_{j \in C} y_j s_D(j) \leq B \tag{2.3.5}$$

$$x_{ij}, y_j \in \{0,1\}. \tag{2.3.6}$$

Note that the objective function Z (line (2.3.1)) is the *total* cost of answering all queries, hence if Z is the solution to the integer program then $Z = m \cdot opt_B$. Of course, the average query cost opt_B is minimized by minimizing total cost Z, and this will be our focus for the remainder of the section.

The variables x_{ij} indicate that index j is used to cover query i, and as such, exactly one x_{ij} is nonzero for each query, corresponding to line (2.3.3), above. In contrast, the y_j indicate whether or not an index is used by some query. This is assured by constraints (2.3.4), which require that y_j be set to 1 if x_{ij} is 1 for any i. The reuse of a single index for multiple queries is captured in the x_{ij} variables, so the total query cost (2.3.1) is in terms of the x_{ij}, whereas the constraint on storage is independent of the queries and expressed (2.3.5) in terms of the index variables y_j. That each query using an index incurs the cost differentiates the problem from the (weighted) set cover problem. Since the IP (line (2.3.6)) cannot be solved efficiently we find an approximate solution by applying an adaptation of the technique of randomized rounding to the optimal fractional solution of the relaxed linear program.

1. Solve the LP relaxation of the IP. Let $\hat{x}_{ij}$, and $\hat{y}_j$ denote the values of the decision variables in the optimal solution.

2. Initialize R to the empty set, and $x^*_{ij} = 0$, $y^*_j = 0$ for all $i \in Q$, $j \in C$.

3. Repeat t times:
   $\forall j \in C - R$
   let $r_j \leftarrow$ random uniform[0,1]
   $\forall i \in Q$
   if $\hat{x}_{ij} \geq r_j$ then $x^*_{ij} \leftarrow 1$
   if $\hat{y}_j \geq r_j$ then $y^*_j \leftarrow 1$ and $R \leftarrow R \cup \{j\}$

4. Output solution set R.

Figure 2.1: Algorithm Integer Programming with Randomized Rounding (IPRR)

Adapted Randomized Rounding.

Consider the LP relaxation of the query IP, obtained by replacing (2.3.6) with the real-valued constraints $x_{ij}, y_j \in [0, 1]$. Given query set Q, storage bound B, index candidate set C, and database D, let $\mathcal{Z}(Q, B, C, D) = (\hat{Z}, \hat{R}, \hat{X}, \hat{Y})$ denote the optimal fractional solution to this LP, where $\hat{Z}$ is the value of the objective function, $\hat{R} = \{j : y_j > 0\}$ is the set of fractional indices chosen by the LP, and $\hat{X} = [\hat{x}_{ij}]$, $i \in Q$,$j \in C$, and $\hat{Y} = [\hat{y}_j]$, $j \in C$ are the resulting values of the decision variables in the solution. Then algorithm IPRR (Figure 2.1) produces a set of indices $R \subseteq \hat{R}$ of storage cost $s_D(R) \approx B \ln m$ covering query set Q with high probability, and such that total query cost $\sum_{i \in Q} c_D(i, R) \approx \hat{Z}$. Denote by x^*_{ij}, and y^*_j the integer assignment to variables x_{ij} and y_j resulting from algorithm IPRR.

The traditional method of randomized rounding of a linear programming relaxation of an integer program works as follows. Each decision variable x_i of the LP is rounded to 0 or 1, based on flipping a coin with bias exactly that of the fractional value of $\hat{x}_i$ in the LP

solution. By linearity of expectation, the expected value of the resource bounds are met, and the expected value of the solution is just the optimal Z from the LP. Some additional probabilistic analysis is typically sufficient to show that with constant probability (which may then be amplified by repetition), the algorithm achieves the desired performance.

This approach fails here, and a novel (to our knowledge) adaptation of randomized rounding is introduced. To understand the issue, notice that if we applied traditional randomized rounding on all the decision variables in our problem, choosing integer values for the variables depending on the outcome of Bernoulli trials, we would be unable to ensure that the $x_{ij} \leq y_j$ constraints were met, since x_{ij} could "flip to 1" while y_j "flipped to 0". What would result would be a nonsensical "solution" dictating that some query i should be covered by using index j, without actually selecting index j into the set R.

We circumvent this problem by allowing some dependence on the flipping procedures. Actually, we do not "flip" coins at all, instead we select a random number r_j uniformly from the interval $[0, 1]$ for each index $j \in C$. Then, for each $i \in Q$, we compare $\hat{x}_{ij}$ to r_j, and set x^*_{ij} to 1 iff $\hat{x}_{ij} \geq r_j$. Similarly, we set y^*_j to 1 iff $\hat{y}_j \geq r_j$. Note that from the solution to the LP, for each $j \in C$, $\hat{y}_j \geq \hat{x}_{ij}$, hence if any x^*_{ij} is set to 1, then so is y^*_j, maintaining the desired inequality constraints after rounding.

Analysis of the algorithm's performance requires consideration of the total space $s_D(R)$ used by the set of indices R returned by the algorithm, and the expected query cost. We show that with high probability, $s_D(R) \leq 3B \ln 3m$, and that the expected total query cost is at most $\hat{Z}$. In addition we show that coverage and the cost bound can be achieved simultaneously.

We make the following observations about the values of the variables x^*_{ij} and y^*_j after one iteration of step 3 of IPRR. These all follow from the uniform choice of r_j in the flipping procedure, the LP constraints, and the linearity of expectation.

1. $Pr(x^*_{ij} = 1) = Pr(r_j \leq \hat{x}_{ij}) = \hat{x}_{ij} = E[x^*_{ij}]$.

2. $Pr(y_j^* = 1) = Pr(r_j \leq \hat{y}_j) = \hat{y}_j = E[y_j^*]$.

3. $\forall i \in Q$ and $\forall j \in C$ the randomized variable assignment yields $x_{ij}^* \leq y_j^*$.

4. $\forall i \in Q$, $E[\sum_{j \in C} x_{ij}^*] = 1$.

5. $E[\sum_{j \in C} y_j^* s_D(j)] \leq B$.

Say that a query i has been *covered* by the algorithm if for some j, x_{ij}^* is set to 1.

Lemma 3. Given a set Q of m queries, each $i \in Q$ of which has $\sum_{j \in C} \hat{x}_{ij} = 1$, after $t \geq \ln(1+\epsilon)m$ repetitions of step 3, with probability greater than or equal to $1 - \frac{1}{1+\epsilon}$, all m queries are covered.

Proof. The probability that a single query i remains uncovered in one iteration of step 3 is $\prod_{j \in C}(1 - \hat{x}_{ij}) \leq (1 - \frac{1}{|C|})^{|C|} \leq \frac{1}{e}$, and the likelihood that it remains uncovered after t rounds is at most $\frac{1}{e^t}$. The probability of *any* of m queries remaining uncovered is thus at most $m(\frac{1}{e^t}) \leq \frac{1}{1+\epsilon}$ when $t \geq \ln(1+\epsilon)m$. □

Lemma 4. Algorithm IPRR, upon $t \geq \ln(1+\epsilon)m$ repetitions of step 3 returns R so that:

1. $R \subseteq \hat{R}$

2. $E[s_D(R)] \leq B \ln(1+\epsilon)m$

3. $Pr(s_D(R) > (1+\epsilon)B \ln(1+\epsilon)m) \leq \frac{1}{1+\epsilon}$.

Proof. Part (1) is straightforward since only the elements in $\hat{R}$ have nonzero probability of being included in R. Part (2) follows immediately from linearity of expectation, together with the fact that we repeat step 3 of the algorithm $t \geq \ln(1+\epsilon)m$ times, incurring at most cost B on each iteration as observed above. For part (3), Markov's inequality states that $Pr(X > kE[x]) \leq \frac{1}{k}$, and applying this to part (2) gives the bound. □

We have shown that, given $t \geq \ln(1+\epsilon)m$ repetitions of step 3, the probability no query is left uncovered is at least $1 - \frac{1}{1+\epsilon}$, and in the previous lemma we see that this same number of repetitions yields that the probability the total storage required is less than $(1+\epsilon)B\ln(1+\epsilon)m$ is at least $1 - \frac{1}{1+\epsilon}$.

Lemma 5. With probability at least 1/3, algorithm IPRR yields a solution R of weight $s_D(R) \leq 3B\ln 3m$ covering the queries in Q.

Proof. Set $\epsilon = 2$. Then the previous two lemmas tell us that the probability of failing to cover some query is at most 1/3, and that the probability that the cost of R is not bounded as stated is at most 1/3. The probability that either of these occur is at most 2/3, so the probability that neither do is at least 1/3 as desired. □

Finally, we will argue that the expected total query cost given this coverage and storage amount, is no more than $m \cdot opt_B$.

Lemma 6. If R is returned by algorithm IPRR and it covers each query in Q, then the expected total query cost is $E[\sum_{i \in Q} c_D(i, R)] \leq m \cdot opt_B$.

Proof. Let $\hat{Z}_i = \sum_{j \in C} \hat{x}_{ij} c_D(i, j)$ denote the contribution of query i in the optimal LP solution. We claim (proof below) that for all i, $E[c_D(i, R)] \leq \hat{Z}_i$. Assuming the claim is true, $E[\sum_{i \in Q} c_D(i, R)] = \sum_{i \in Q} E[c_D(i, R)] \leq \sum_i \hat{Z}_i = \hat{Z} \leq m \cdot opt_B$. □

Proof of Claim: Note that since repeated iterations of step 3 of the algorithm can only decrease the cost of a query, we prove that the expected cost of a query covered in the first iteration is less than its contribution in the LP solution. Let $|C| = s$ and $c_D(i, j_1) \leq c_D(i, j_2) \leq \ldots \leq c_D(i, j_s)$, and recall that $\hat{x}_{ij}$ from the solution to the LP is the probability

that query i is covered by index j in 1 iteration. For simplicity of notation, let

$$\begin{aligned} P_i^k &= \prod_{m=1}^{k}(1-\hat{x}_{ij_m}), \\ P_i^0 &= 1 \end{aligned}$$

denote the probability that query i is not covered by any of indices $j_1, j_2 \ldots j_k$. By the independence of the uniform random variables $r_j, j \in C$, the conditional probability that covered query i incurs cost $c_D(i, j_k)$ is

$$Pr(\text{ query } \quad i \quad \text{covered} \quad \text{by} \quad \text{index} \quad j_k \mid \text{query} \quad i \quad \text{is} \quad \text{covered})$$

$$= \frac{P_i^{k-1}\hat{x}_{ij_k}}{1-P_i^s}.$$

That is, given that a query is covered, the probability it incurs cost $c_D(i, j_k)$ is just the probability it has not been covered by an index of lower cost times the probability it is covered by index j_k, normalized by the probability of coverage. We must show that $E[c_D(i, R)] \leq \hat{Z}_i$, or

$$\frac{\sum_{k=1}^{s} P_i^{k-1}\hat{x}_{ij_k}c_D(i, j_k)}{1-P_i^s} \leq \sum_{m=1}^{s}\hat{x}_{ij_m}c_D(i, j_m).$$

As a base case for an inductive argument, let $s = 2$. We show that

$$\frac{\hat{x}_{ij_1}}{1-P_i^2}\cdot c_D(i, j_1) + \frac{P_i^1\hat{x}_{ij_2}}{1-P_i^2}\cdot c_D(i, j_2) \leq \sum_{m=1}^{2}\hat{x}_{ij_m}c_D(i, j_m).$$

Since the coefficients of the $c_D(i, j_k)$ sum to 1 on both sides of the inequality, we need only show that the coefficient of the more expensive index cost is larger on the right side, indicating that the index of higher cost has greater weight in the LP solution. In other

words, we just need to show that

$$\frac{P_i^1 \hat{x}_{ij_2}}{1 - P_i^2} \leq \hat{x}_{ij_2},$$

or

$$\frac{P_i^1}{1 - P_i^2} \leq 1,$$

which after expanding P_i^1 and P_i^2, is obviously true since $(1 - \hat{x}_{ij_2}) = \hat{x}_{ij_1}$.

Next, assume inductively that the assertion holds for $s = k - 1$ indices and use that assumption to prove the case $s = k$. To simplify notation, represent the inductive hypothesis by $S_{k-1} \leq L_{k-1}$, and the result we hope to prove by $S_k \leq L_k$. It is easy to see that

$$\begin{aligned} S_k &= S_{k-1} \cdot \frac{1 - P_i^{k-1}}{1 - P_i^k} + \frac{P_i^{k-1} \hat{x}_{ij_k}}{1 - P_i^k} \cdot c_D(i, j_k) \\ &\leq S_{k-1} + \frac{P_i^{k-1} \hat{x}_{ij_k}}{1 - P_i^k} \cdot c_D(i, j_k). \end{aligned}$$

Now by the inductive hypothesis and linearity of the objective function of the LP ($L_k = L_{k-1} + \hat{x}_{ij_k} c_D(i, j_k)$), we need only focus on the last terms of the sums and show that

$$\frac{P_i^{k-1} \hat{x}_{ij_k}}{1 - P_i^k} \cdot c_D(i, j_k) \leq \hat{x}_{ij_k} c_D(i, j_k),$$

or equivalently, that

$$\frac{P_i^{k-1}}{1 - P_i^k} \leq 1.$$

We now need to show that

$$\begin{aligned} P_i^{k-1} &\leq 1 - P_i^k \\ P_i^{k-1} + P_i^k &\leq 1 \\ P_i^{k-1}(1 + (1 - \hat{x}_{ij_k})) &\leq 1 \text{ (since } P_i^{k-1}(1 - \hat{x}_{ij_k}) = P_i^k) \\ P_i^{k-1} &\leq \frac{1}{1 + (1 - \hat{x}_{ij_k})}. \end{aligned}$$

Since $\sum_{m=1}^{k-1} \hat{x}_{ij_m} = 1 - \hat{x}_{ij_k}$ we have

$$\begin{aligned} P_i^{k-1} &\leq (1 - \frac{1 - \hat{x}_{ij_k}}{k-1})^{k-1} \\ &= (1 - \frac{1 - \hat{x}_{ij_k}}{k-1})^{\frac{k-1}{1-\hat{x}_{ij_k}} \cdot (1-\hat{x}_{ij_k})} \\ &\leq (\frac{1}{e})^{1-\hat{x}_{ij_k}}, \end{aligned}$$

and we thus must show that

$$(\frac{1}{e})^{1-\hat{x}_{ij_k}} \leq \frac{1}{1 + (1 - \hat{x}_{ij_k})}.$$

Taking logs shows the inequality holds when $1 - \hat{x}_{ij_k} \geq \ln(1 + (1 - \hat{x}_{ij_k}))$, which is true, with equality when $\hat{x}_{ij_k} = 1$.

Theorem 7. With probability at least 1/3, algorithm IPRR outputs a solution R requiring space $s_D(R) \leq 3B \ln 3m$ and expected cost per query opt_B.

Proof. By Lemmas 5 and 6, a set of indices R is obtained meeting the above space bound, and with expected total query cost $E[\sum_{i \in Q} c_D(i, R)] \leq \hat{Z} \leq m \cdot opt_B$, and so the expected query cost is at most opt_B. □

2.4 Applications and Related Work

Index selection in relational databases has received much attention [8, 7, 15, 2, 26] since the pioneering work of Chaudhury and his collaborators [7, 8, 2]. An assumption in most of these papers, and typical in relational databases, is the division of P into large subsets corresponding to particular attributes, and then a decision to build indices corresponding to each such subset either completely or not at all. For example, if the age attribute is selected as an index attribute in an Employee relation, the index set R will include predicates age=34, age=35, age=36, etc., one predicate for each possible value of the age attribute. Our more general formulation makes it possible to choose to index age=34 or $25 \leq \mathsf{age} \leq 37$ without having to index age=36 at the same time.

Index selection has also begun to receive some interest recently for XML databases [36, 32, 28, 24, 21, 9]. In addition to the issues parallel to those in relational databases, we also have the possibility of choosing path indices on selected paths in the database. Our work makes it possible to choose these indices in a data-driven manner rather than only in terms of schema and type. In fact, we can even construct and use "tree indices" that can return matches to portions of a query pattern tree. For example, queries that involve independent path segments could be indexed based on containment of one or more of the paths and algorithm IPRR used to choose which single segments, or collections of segments, to index on based on utility in the workload.

Unlike database systems, information retrieval systems explicitly choose the terms on which to construct an inverted index. A typical choice may be to index every single word occurrence, except for words on a *stop list*, and in addition to index certain common multi-word phrases. The approach in this paper may provide a principled basis for choosing exactly which words and which multi-word phrases to index. It even becomes possible to allow multi-word indices where these words are not in a single contiguous phrase.

Bioinformatics applications have recently become popular. In this context, a record might be a DNA sequence, and a query a request for all records containing a given subsequence [30].

Our problem is closely related to the problem of choosing which views to materialize on the datacube [16, 14, 20, 37]. In [16] an algorithm was given approximating within a factor of 0.63 the maximum amount of *gain* (the reduction in cost from the situation where no views are stored) that could be achieved on a workload by selecting certain views. In [20], it was pointed out that an approximation guarantee for maximum gain does not guarantee an approximation on performance cost over the workload, and demonstrated that the algorithm of [16] can do no better than factor of $M/12$, and as discussed in Section 2.2, that under standard complexity-theoretic assumptions, *no* algorithm could do significantly better even allowing any linear relaxation in the number of views the algorithm is allowed to select. Our results fit in nicely in this context; straightforward modifications show that expected optimal performance can be obtained by allowing a logarithmic relaxation in the number of views selected.

2.5 Sampling

In this section we generalize the index selection problem to that of selecting a set of indices that perform well on an unknown *distribution* of queries, rather than on a particular workload, as before. To this end, we consider a randomly selected query workload to be a representative sample from the distribution, and give an approximate solution on the sample workload as in Section 2.3. Using a uniform convergence lemma we argue that with high probability the approximate solution on the sample differs only slightly from the optimal solution on the distribution, given a sample of sufficient size.

Before we formalize and address the more general problem, we acknowledge the possibility of a degenerate solution under certain circumstances and suggest a solution. We are concerned that the approximately optimal solution for the random query workload found

by the algorithm will not be useful on future queries from the distribution. Specifically, suppose the approximately optimal solution found by the algorithm, that which gives least average query cost, is to store the exact query responses for every query. This is clearly the optimal solution given adequate storage, but it is unlikely to be useful on future queries from the distribution. To circumvent this possibility, we impose an additional constraint on the problem. Namely, we limit the cardinality of the index set so that even if there is enough space to store a unique index for every query, we cannot do so because we limit the *number* of indices allowed. In the context of the linear program from Section 2.3 this is just an additional feasibility constraint bounding the sum of the index variables, y_j, $j \in P$. In the remainder of this discussion, we refer to this bound as the index bound, N.

Following the analysis of Section 2.3, we must assure that this index bound can be met simultaneously with the storage bound while still assuring coverage of the queries. A lemma analogous to Lemma 5 states that if $\epsilon = 4$ then with probability 1/4 all conditions can be met simultaneously.

We are now ready to state the problem on a query distribution:

Definition 8. Let D be a (finite) set of data. Let P be a (possibly infinite) set of polynomially-computable predicates. Given a distribution $Q \subseteq P$ of queries, a candidate set of indices $C \subseteq P$ and bounds B and N, find a set $R \subseteq P$ such that $s_D(R) \leq B$, and $|R| \leq N$ and such that $E_Q[c_D(q, R)]$ is minimized.

Theorem 10 below is motivated by the following uniform convergence lemma which prescribes a sample size adequate for inferring the expected value of a class of functions over the sampled distribution:

Lemma 9 *([17, 31]).* Let F be a finite set of functions on X with $0 \leq f(x) \leq M$ for all $f \in F$ and $x \in X$. Let $S = x_1, \ldots, x_m$ be a sequence of m examples drawn independently and identically from X and let $\epsilon > 0$. If $m \geq \frac{M^2}{2\epsilon^2}(\ln |F| + \ln \frac{2}{\delta})$ then

$$Pr(\exists f \in F : |E_X(f) - E_S(f)| \geq \epsilon) \leq \delta.$$

Theorem 10. Given $\epsilon > 0$, $\delta \in [0,1]$, a storage bound B, an index bound N, a candidate set of indices C, and a random sample S of size $m \geq \max\{\frac{4}{\epsilon^2}\ln\frac{2}{\delta}, (\frac{32N\ln|C|}{\epsilon^2})^2\}$ independently chosen from query distribution Q, algorithm IPRR returns a set of indices R with $s_D(R) \leq 4B\ln 4m$, and $|R| \leq 4N\ln 4m$, so that with probability at least $1-\delta$,

$$E_Q[c_D(q,R)] - E_Q[c_D(q,R_{opt})] < \epsilon M,$$

where R_{opt} denotes the optimal set of indices for the distribution, with $s_D(R_{opt}) \leq B$ and $|R_{opt}| \leq N$.

Proof. The proof consists of a justification of the lower bound on the sample size together with an analysis of the accuracy. The proof of accuracy is transitivity applied to the following 4 steps which together hold with probability $1-\delta$ (steps 2 and 3 with certainty):

1. $E_Q[c_D(q,R)] - E_S[c_D(q,R)] < \frac{\epsilon M}{2}$
2. $E_S[c_D(q,R)] \leq E_S[c_D(q,R_S)]$, where R_S is the optimal set of indices for query set S.
3. $E_S[c_D(q,R_S)] \leq E_S[c_D(q,R_{opt})]$
4. $E_S[c_D(q,R_{opt})] - E_Q[c_D(q,R_{opt})] < \frac{\epsilon M}{2}$

Step 2 is proven in Section 2.3 of the paper, where algorithm IPRR is described. Step 3 is obvious by the optimality of R_S. Steps 1 and 4 of the proof require application of Lemma 9 above. To apply the lemma we define a family of functions F on Q, where $f_R \in F$ is characterized by a set of indices R from the set of possible indices. In our application, the set of functions F is just the set of query cost functions $c_D(\cdot,R)$, where the query cost is computed relative to a set R. The cardinality of F, $|F|$, is the number of ways of choosing such a set, or $\binom{|C|}{4N\ln 4m}$, which is at most $|C|^{8N\ln m}$. By Lemma 9, and given the sample size in the theorem, the sample and true average query costs for every $f \in F$ differ by no more than $\frac{\epsilon M}{2}$ with probability at least $1-\delta$. This proves both step 1 and step 4. The argument

that the sample size given in the theorem is sufficient to admit the described error bounds is standard:

According to Lemma 9, and using $|F| \leq |C|^{8N \ln m}$ we have the following lower bound on the sample size:

$$\begin{aligned} m &\geq \frac{2}{\epsilon^2}(\ln |C|^{8N \ln m} + \ln \frac{2}{\delta}) \\ &= \frac{2}{\epsilon^2}(8N(\ln |C|)(\ln m) + \ln \frac{2}{\delta}) \\ &= \frac{16N \ln |C|}{\epsilon^2}(\ln m) + \frac{2}{\epsilon^2}(\ln \frac{2}{\delta}) \end{aligned}$$

We can bound m by taking twice the larger term, so we have:

$$m \geq \max\{\frac{32N \ln |C|}{\epsilon^2}(\ln m), \frac{4}{\epsilon^2}(\ln \frac{2}{\delta})\}$$

Notice, however, that the lower bound is written in terms of m. To eliminate this complication, we recall that $\ln m \leq m^{\frac{1}{2}}$. Substitution yields:

$$m \geq \max\{(\frac{32N \ln |C|}{\epsilon^2})^2, \frac{4}{\epsilon^2}(\ln \frac{2}{\delta})\}.$$

□

Though we have considered the candidate set of indices to be an input to the problem, a similar sample size bound holds if we simply assume that the candidate set can be polynomially computable from the sample of queries. Specifically, if $|C| = p(m)$, where $p(m)$ is a polynomial of degree $c - 1$, sample size

$$m \geq \max\{(\frac{32Nc}{\epsilon^2})^2, \frac{4}{\epsilon^2}(\ln \frac{2}{\delta})\},$$

is sufficient to guarantee the accuracy stated in the theorem.

2.6 Conclusions

A general technique of (partial) index selection for expected cost minimization given space constraints was presented, and it was shown that relaxation of the space constraints by a logarithmic factor allows an optimal expected solution. Previous results are used to show that the space relaxation is necessary.

The cost model used was a simplification of a more natural one involving intersection of RIDs, which we hope to address in future work, along with other cost models.

Chapter 3

Optimized Association Rules

The search for meaningful patterns in large datasets is one of the main foci of data mining research. A well-investigated type of pattern is the *association rule* (introduced in [1]) a rule of the form $X_1 X_2 \ldots X_s \rightarrow C$, meaning, in essence, that "when $X_1, \ldots, X_s$ all hold about a datum, then C tends to hold also". Often, data is so-called "market-basket" data, and X_i and C are boolean variables indicating the presence of some item in a customer's order. Such a rule is useful when it has sufficient *support* and *confidence*. The support of a rule is the number or percentage of records for which the antecedent $X_1 \ldots X_s$ holds. The confidence is a measure of the validity of the implication – the percentage of time the consequent C holds given that the antecdent holds. The literature is rich with work on how to effectively find such rules and their variants (see [19] for a survey).

The idea of *optimized* association rules was introduced by Fukuda et al. [12]. Consider a single large relation. The motivation is that in many settings, a user is interested in a specific attribute c of the data as a consequent in an association rule (e.g., c corresponds to "good credit risk"), as well as a collection of antecedent attributes X_i whose values are likely to be predictive of c. The goal is to find one or more instantiations of the attributes X_i that will result in rules of reasonable confidence and as high support as possible. For example, in a credit-history database, $X_1 \in \{$ single, married, divorced$\}$ might represent marital status, X_2 and X_3 might be numeric attributes representing income and age, respectively, and $c \in \{0, 1\}$ might represent creditworthiness based on past history or by expert judgement.

A discovered rule in the existing database such as $(X_1 = \text{ married}) \wedge (X_2 > \$50K) \wedge (X_3 \in [30, 60]) \rightarrow (c = 1)$ might be useful in making decisions about new cases.

Though the problems of finding association rules and of finding optimized association rules share the common goal of identifying portions of data where Boolean conditions are satisfied, they differ markedly in their search domain. When finding association rules, we search among possible subsets of attributes for combinations that are predictive of the Boolean condition. On the other hand, the quest for optimized association rules requires no search among subsets of attributes, indeed the attribute(s) of interest are given, but rather, the hunt is among all possible *instantiations* of the given attribute(s) for descriptions of data satisfying the Boolean condition.

More formally, let D be a large relation. For any subset $S \subseteq D$ of data, define

- The *support* of S, written *support*(S), is just $|S|$, the cardinality of the set.
- The set $S^+ = \{s \in S : s.c = 1\}$.
- The *confidence* of S, written *conf*(S), is the fraction of S that is S^+. That is, *conf*$(S) =$ *support*(S^+)/*support*(S)
- If $S_1, \ldots, S_k$ are subsets of D, then the *cumulative support* and *cumulative confidence* of the sets are, respectively, the support and confidence of the union $\cup_{i=1}^{k} S_i$.

Results vary based on the form of the set S. Fukuda, et al. [12] first defined the problem, and considered the case that S is specified by a single numeric attribute, and later generalized the problem to that of two numeric attributes [12, 11]. Efficient algorithms were given for maximizing the cumulative support given a minimum confidence threshold, for the dual problem of maximizing the cumulative confidence given a minimum support threshold, and for maximizing the "gain" of a rule. Rastogi and Shim generalized the problem to allow unions of categorical attributes [35] and of one or two quantitative attributes [34]. Zelenko

gave a polynomial time algorithm for unions of categorical attributes [42]. Wijsen and Meersman offer an investigation into the inherent complexity of variants of the problem [40].

To better understand our results and their relationship to past work, we need just a couple more definitions. Let D be a data set of cardinality n, with each datum d containing a single real-valued attribute $d.r$ and a boolean "consequent" attribute $d.c$. Without loss of generality, assume that each value $d.r \in \{1, 2, \ldots, n\}$, reflecting the possibility of at most n distinct values.

An interval I is just a pair (a, b) with $a \leq b$, and represents the set $D(a, b) = \{d \in D : a \leq d.r \leq b\}$.

Given a data set D as described above, a minimum confidence value $\theta \in [0, 1]$, and a positive integer k, the *max-support-min-cumulative-confidence* problem is to find a collection $\mathcal{I} = I_1, I_2, \ldots, I_k$ of k intervals such that the cumulative confidence *conf*($\mathcal{I}$) is at least θ, and such that the cumulative support *support*($\mathcal{I}$) is maximized. Alternatively, the *max-support-min-independent-confidence* problem is to find a collection $\mathcal{I}$ of k intervals such that each interval of $\mathcal{I}$ has confidence at least θ and such that the cumulative support *support*($\mathcal{I}$) is maximized.

Our contributions are as follows:

- We provide the first polynomial time algorithm for the max-support-min-cumulative-confidence problem for a single numeric attribute (this problem had been thought NP-hard).

- For the max-support-min-independent-confidence problem, we analyze the performance of an algorithm of Rastogi and Shim which previously had been shown to scale well on random unpatterned data. We test the algorithm on several real-world data sets, and find that the algorithm does not scale as well, but rather behaves as it does on a proposed synthetic model of random *patterned* data.

- We propose sampling as a preprocessing phase for any algorithm addressing either the independent or the cumulative confidence problem, and derive theoretical bounds on the sample size sufficient to achieve near-optimal performance. Because the bounds are independent of the size of the original data set, dramatic speedups are possible.

- Experimental results demonstrate the utility of the sampling approach; a sample of size less than 500 was sufficient in all cases to obtain a solution within 5% of optimal on both real world and on synthetic data models.

- We give a straightforward, linear time greedy algorithm for solving the max-support-min-independent-confidence problem, and provide performance guarantees. Specifically, our algorithm achieves support level at least 1/3 of the optimum, and that this is the strongest guarantee possible for this natural greedy approach.

The remainder of this paper is as follows. In Section 3.1 we give a polynomial time algorithm for solving the max-support-min-cumulative-confidence problem. Section 3.2.1 briefly reviews the algorithm of Rastogi and Shim, its complexity, and their results on random data. Following this, in Sections 3.2.2, and 3.2.3 we present a synthetic model for patterned data, and analyze the performance of the RS algorithm on this data, as well as on several real-world data sets. In Section 3.3 we turn to sampling as a means of speedup, and demonstrate the utility of sampling both theoretically and empirically. Finally, in Section 3.4 we give a $O(nk)$ greedy algorithm with performance guarantees.

Some of the results reported in this chapter are based on joint work with J. Elble and L. Pitt, and appear in preliminary form in [10].

3.1 Maximum support meeting minimum cumulative confidence

In [35] a reduction from the NP-hard (Weighted) Set Cover problem to the max-support-min-cumulative-confidence problem is given, showing that this latter problem is NP-hard. The reduction translates weights from the set cover problem directly into support and confidence values for the max-support problem, rather than creating a data set D that realizes these values. Because the NP-hardness relies on very large values of support and confidence that cannot be realized by a polynomially-sized database, it does not neccessarily apply to the problem in which a database is given as part of the input. But it is exactly this latter problem that is of interest to us; it is only natural to allow an algorithm for mining a database to at least scan the data, spending time polynomial in $n = |D|$ and k. The NP-hardness result of [35] does not preclude the existence of an algorithm that takes time poly(n, k). We give just such an algorithhm, admittedly impractical due to the degree of the polynomial. However, the algorithm stands as a challenge for improvement to a practical polynomial-time solution. Alternatively, perhaps some of our ideas on sampling for data reduction for problems of this type, as discussed in Section 3.3, can be employed here.

It is perhaps worth noting that a trivial exhaustive algorithm solves the problem in time $O(n^{2k})$: There are at most $\binom{n}{2} = O(n^2)$ distinct intervals. After tallying the support and confidence of each, the cumulative support and confidences for each of the at most $O(\binom{n^2}{k}) = O(n^{2k})$ distinct choices of k intervals can be computed, and the optimal k-tuple selected.

However, we would like an algorithm that runs in time polynomial in k, not exponential. Ideally, it would be linear in n. We will leave as an open question the exact complexity of the problem, and offer in this section an embarrassing $O(n^5 k)$ algorithm. This is not offered as a practical approach, but rather as a "proof of concept".

We use dynamic programming in a manner similar to that of [34], taking advantage of a bound on confidence c as was done in [42]. For every interval $[i, j]$ on n data points, and every number of disjunctions l, the maximum support l-interval on $[i, j]$ exceeding minimum confidence has confidence tally c, where c is in the range $[1..n]$. Intuitively, for every possible confidence level c and for every interval $[i, j]$, and for every disjunct size l, store the largest support region exceeding minimum confidence (θ), together with its support. Call that value $S([i, j], l, c)$. To solve the problem, we want $S([1, b], k, n\theta)$.

Consider $S([i, j], 1, c)$. From Fukuda et al. [12], these n^3 values can each be computed in $O(n)$ time. For the general case, subsequent values of l can be computed in terms of previous values, for each interval and confidence level. Specifically, notice that every solution for l intervals on $[i, j]$ of confidence c can be partitioned into a disjoint union of an optimal solution for 1 interval on $[i, m]$ with confidence c', and an optimal solution for $l - 1$ intervals on $[m+1, j]$ with confidence $c - c'$, for some choice of m and c'. The task, then is to find the best choice of $m, c' \in [1 \ldots n]$. We know of no other way to do this than searching through all their possible values. Thus we have:

$$S([i, j], l, c) = \max * _{1 \leq m, c' \leq n} \{S([i, m], 1, c') \cup S([m + 1, j], l - 1, c - c')\},$$

where $max*$ denotes the set of maximum support among the n^2 unions.

The total running time of the algorithm is $O(n^5 k)$, which corresponds to $O(n^2)$ update time for $n^3 k$ tabulated values. We leave open the question of whether or not the entire table is necessary, and whether or not updates can be done more quickly.

3.2 Maximum support meeting minimum independent confidence

3.2.1 The RS algorithm

Rastogi and Shim [34] attack a more tractable version of the problem, in which a disjoint union is found consisting entirely of intervals *each* meeting the minimum confidence constraint. Because of the additional constraint, they are able to obtain a relatively fast algorithm (which we denote "RS"), whose performance will be of interest in our discussion of the efficacy of sampling.

The RS algorithm, like [12, 11] assumes initially that the data has been pre-bucketed, so that the input consists of n buckets $b_1, \ldots, b_n$, where each bucket corresponds to a particular discretized value of the quantitative attribute's range. The problem is to find k intervals each of which contains some contiguous collection of buckets. Each bucket may be thought of as an indivisible unit, corresponding to a weighted point with weight equal to the support of the bucket, and with "confidence" value c a real number in the *interval* $[0, 1]$, as opposed to the *set* $\{0, 1\}$. They assume also that the buckets have been sorted in order of increasing value of the attribute.

The RS algorithm has three phases:

1. Preprocess the data by merging all adjacent buckets that have confidence at least θ (the minimum confidence threshold). There is never any reason to include one of these without its sufficiently high confidence neighbors - they will all be in the same interval.

2. Find in linear time, all "partition points". A partition point is one that cannot possibly be in any interval of confidence at least θ. Intuitively these are buckets with low confidence that are situated in the neighborhood of sufficiently many low confidence buckets (or sufficiently high-support low-confidence buckets) that no interval could possibly contain them and still meet the θ threshold. (In our implementation of the

RS algorithm we also merged adjacent partition points, thereby further reducing the number of buckets. The amount of data reduction was similar to that seen for step 1, above.) Notice that this approach would not work for the minimum cumulative confidence version of the problem discussed in Section 3.1, since even very low confidence intervals can participate in a global solution of high cumulative confidence.

3. Solve the k-interval problem separately on the subproblems between the partition points, using a dynamic programming approach, and merge the solutions to obtain a global choice of the k best intervals.

If there are no partition points, and there are b buckets after the merge procedure, then the running time of RS is that of the dynamic programming of the 2^{nd} step running on a single group of b buckets, which is $O(b^2k)$. Suppose, on the other hand, that the n original buckets are divided into m subgroups of $n_1, n_2, \ldots, n_m$ buckets separated by $m-1$ or more partition points, with $\sum_i n_i = m$. Then if $b_{\max} = \max\{n_i\}$ is the number of buckets in the largest subproblem, then they show their algorithm takes time at most $O(b_{\max}^2 mk + mk^2)$, where the first term is for solving the m subproblems, and the second term is the cost of combining the m local solutions into a single global solution. When there are many partition points, the divide and conquer step dramatically reduces the running time, as demonstrated by [RS]. In essence, this is because the sum of the squares is typically much smaller than the square of the sum. ($\sum_i n_i^2 << (\sum_i n_i)^2$). It is useful to consider the two extremes:

- If there are many partition points, the data can splinter into a linear number of constant-sized sets. In this case, the run time will be dominated by the second term mk^2, and will grow only linearly with the size m of the data set.

- If there are few partition points, the data will not lend itself to the divide-and-conquer approach, since $b_{\max}$ will be large ($O(n)$). In this case the run time will be dominated by the first term, and will increase quadratically with the size of the data set.

The natural question then is: "Where between these two extremes will the algorithm's performance fall *in practice*?"

In order to empirically test the utility of the preprocessing and divide and conquer approach, in [34] the algorithm was run on synthetic data. In this model, "phantom" binned data was created: For each b_i, $1 \leq i \leq n$, a support value s_i and confidence value c_i was created to represent s_i data points, $c_i \cdot s_i$ of which corresponded to data points x satisfying the consequent requirement $x.c = 1$.

The values s_i were chosen uniformly in $[0, \frac{2}{n}]$ (as in [34]), and then normalized so that the sum of all s_i was 1. The confidence values c_i were chosen uniformly in $[0, 1]$. As a consequence, there is no correlation between the confidence values in bucket b_i and b_{i+1}. The net result of using random data was that the partitioning algorithm performed very well, because, as noted by the authors, long stretches of high confidence were unlikely. The algorithm was able to handle problems with up to 100,000 initial buckets in less than 15 seconds. Even with this much data, $b_{\max}$ was typically around 10 or 20. In other words, the first case (linear growth) was observed.

It is not difficult to show using Chernoff bounds that if $\theta > .5$, with probability approaching 1 exponentially quickly, the problem fragments into constant-sized chunks separated by partition points, explaining the dramatic improvement offered by the partitioning algorithm on such random data. On the other hand, if $\theta < .5 - \epsilon$, then with probability approaching 1 exponentially fast as n increases, the single interval encompassing all of the data is an optimal solution, and, a constant time algorithm suffices (choose the entire data set!). It is only when θ is close to .5 (the mean confidence) that the algorithm's performance degrades, as observed empirically in [34].

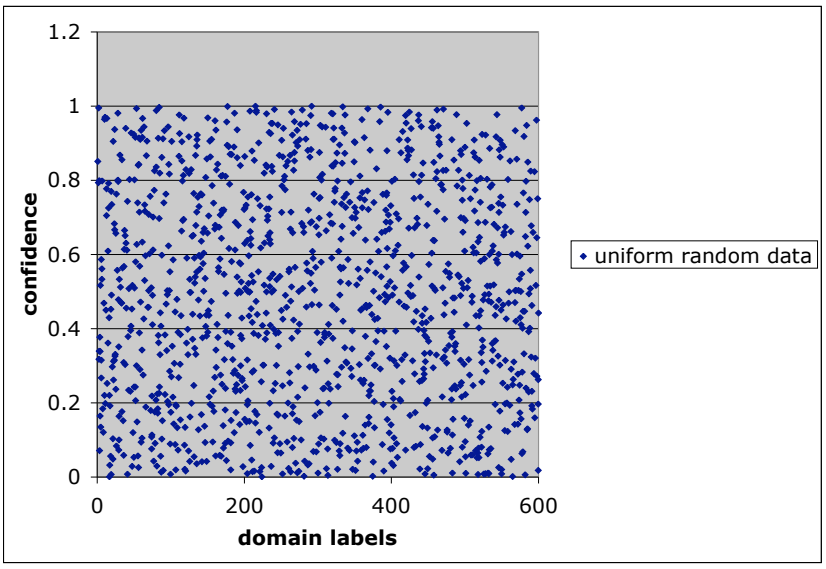

Figure 3.1: Uniform random data.

3.2.2 Synthetic Patterned Data and Real World Data: Description

While the RS algorithm was shown to provide dramatic speedups, this was only for a data model corresponding to "unpatterned," uniform random data. It is reasonable to assert that real world data sets of interest would be less random than the artificial data generated in [34] (see Figure 3.1). It is precisely the randomness of their data model that creates the fortuitous partitioning whose result is speedy run times.

The true utility of any method can only be demonstrated empirically by performance analysis on many different instances of real-world data. On the other hand, the availability of parameterized data models may be used to advantage to gain insight into how an algorithm's performance depends on various data characteristics. Toward this end, to better understand how the RS algorithm performs, we develop a model of random patterned data, and carefully examine the role played by various parameters controlling the data distribution and the effect

on the run time of the algorithm. We also consider the behavior of the algorithm on real world data selected from a census database (prediction of marital status from income), and from forestry data (prediction of ground cover type from elevation). For the patterned synthetic data and for the real world data sets we find that the dramatic improvement observed on unpatterned random data is not typical, and that the behavior of the RS algorithm on the real world data more closely matches our patterned data model, for which quadratic runtime dominates. We conclude that such adverse data is not rare.[1]

Random patterned data

Our synthetic data was generated according to the following method. Each bucket received support as in [34]. We assumed, however, that confidence would be a function of the numeric attribute value (i.e., the bucket number), so that low confidence buckets would tend to cluster, as would high confidence buckets. We generated a simple triangular wave-form with varying numbers of peaks and valleys.

A typical peak had confidence values rising from .2 to .8, and then falling back to .2. Call this multipeak piecewise linear function function f. We randomly generated the confidence for bucket i to be a binomially distributed ratio with mean $f(i)$, and $N = 20$ trials. Figure 3.2 shows a typical example data set generated in this manner.

Perhaps a more natural distribution would be gaussian, or a mixture of gaussians. In any case, we do not mean to suggest that this synthetic model is "the right" one - indeed, we do not believe such a thing exists. The point is to consider how in practice the algorithm performs as parameters such as the size and frequency of peaks change, perhaps empirically validating the tradeoff suggested between the two terms in the complexity bounds for the divide and conquer algorithm.

[1]No special effort was made to find these data sets - they were selected due to their availability, their size, and the presence of a quantitative attribute with many possible values that was likely to be predictive of an associated categorical attribute.

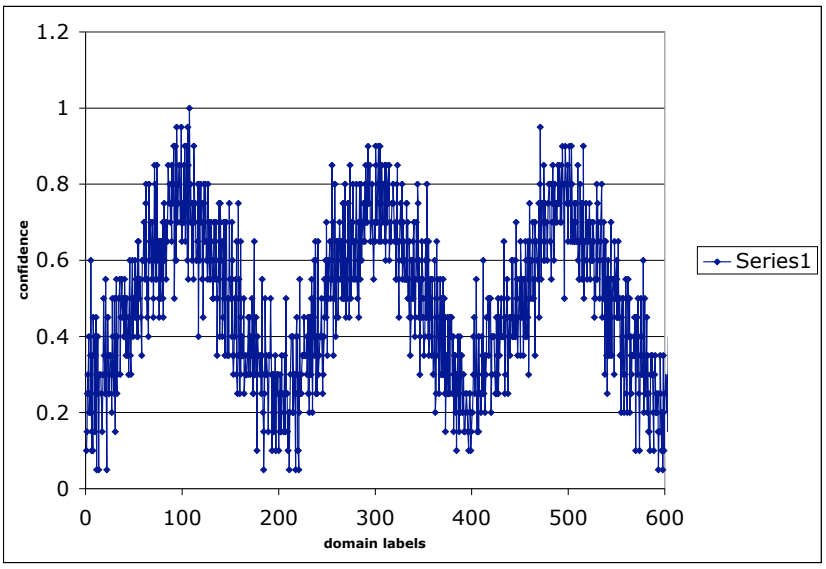

Figure 3.2: Peaked data

We generated data with $n = 2000$, 4000, 8000, and 16000 buckets (distinct domain values for the quantitative attribute). This was done in two ways. For "fixed peaks" data, the number of peaks for these datasets were, respectively, $n/50$, $n/100$, $n/200$, and $n/400$, thus keeping the number of peaks constant at 40, and hence the peak width increasing from 50 to 400. For "fixed peak width" data, the number of peaks was set at $n/200$ for each data set, so that the peak width was fixed at 200, and the number of peaks varied from 10 to 80. Because each valley was likely to contain at least one partition point, and the neighborhood around each peak was unlikely to do so, this effectively allowed control of the parameters $b_{\max}$ (here, the peak width) and m (here, the number of peaks) in the RS algorithm.

Real-world data

In addition to the patterned synthetic data, we extracted a real world data set from the 1999 census data for the Los Angeles/Long Beach area. The total-family-income and marital-

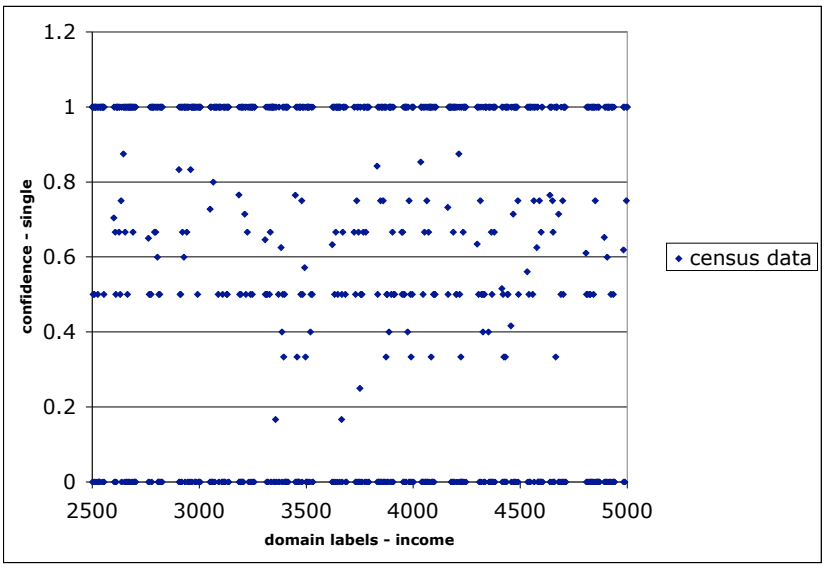

Figure 3.3: Census data

status attributes were projected from a database of 88443 households with positive income levels. The marital-status attribute was summarized by a boolean attribute whose values were 0 for a married head of household, and 1 otherwise. Finally, the data was bucketed according to the total-family-income, so that one record existed for every value in the income domain. A record consisted of an income level together with a tally of the number of households with that income, and a tally of the number of unmarried heads of household. The largest final data set consisted of 11990 records. This same procedure was followed on smaller census data sets to achieve smaller domains for our tests. Figure 3.3 shows a small portion of the census data set. As a simple test point for scientific data in this context, we tested the RS algorithm on the forestry data available from the UCI KDD archive at http://kdd.ics.uci.edu/. We used a similar procedure to that of the census data to extract the elevation and forest cover type for 581,012 soil samples. The elevation and cover type attributes were projected from the set of observations. The cover type attribute was

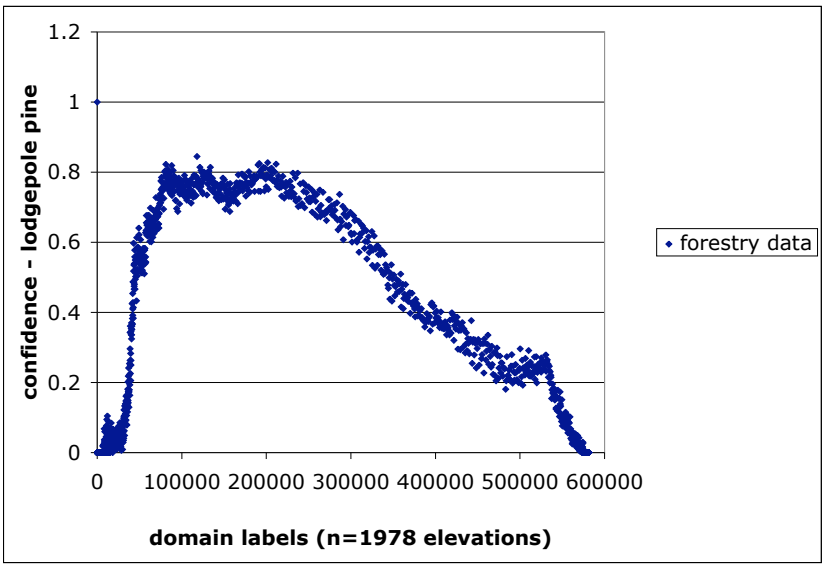

Figure 3.4: Forestry data

condensed into a single boolean attribute whose value was 1 if the forest cover was "lodgepole pine", and 0 otherwise. Finally, the data was bucketed according to unique elevations. Each record in the final data set consisted of an elevation followed by a tally of the number of observations that were taken at that elevation and another tally of the number of samples at that elevation classified as cover type "lodgepole pine". The final data set had only 1978 elements (elevations). Figure 3.4 shows the entire forestry data set.

3.2.3 Results of RS algorithm on patterned and census data

We ran[2] the RS algorithm on each patterned data set for $k = 5$ intervals. We fix the value of k, since we are most interested in characteristics of the data, and how those characteristics

[2]All of the experiments presented in this paper were implemented in Perl and performed on an iBook laptop. Note that the point of our investigation was not to determine the maximum problem sizes that can be handled by the algorithm based on the limits of current technology, but rather to see how well the algorithms scale with various parameters. Our results will be off by a fixed constant factor when compared to implementations with faster hardware and software. (For example,the run times presented here are consistently 50 times slower than that of the implementation used by Rastogi and Shim.)

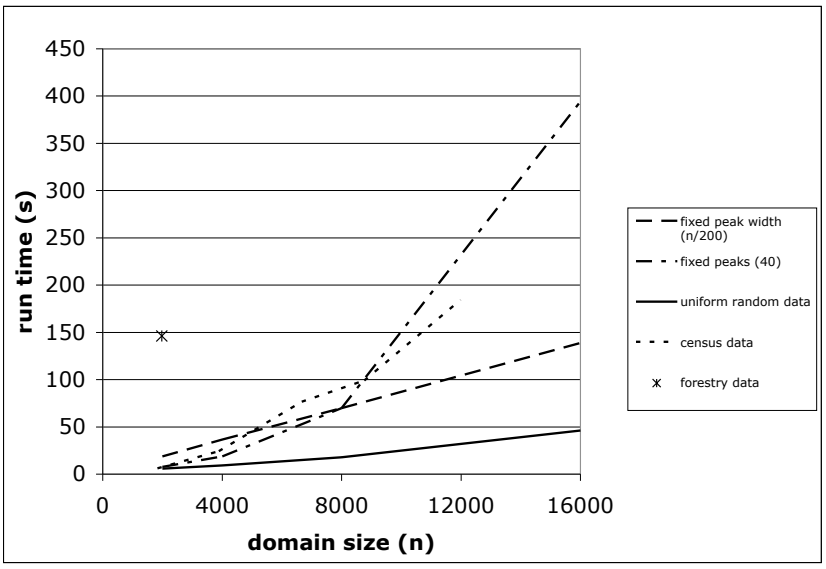

Figure 3.5: Data size vs. runtime

affect running time. For each setting of the parameters, each experiment was run 30 times, and the average run times were recorded. As expected, because there were stretches of correlated confidence, within each peak there were sequences of buckets that neither merged together in the initial preprocessing phase, nor resulted in a partition point.

Figure 3.5 gives the running time for finding 5 intervals, as a function of the domain size. We expect the most salient parameter to be the peak width ($b_{\max}$). As discussed earlier, this value is a small constant for the uniform random data, so run time scales linearly with a small slope as the amount of data increases. For the fixed peak-width data ($n/200$), we also have a linear increase in running time, but with a larger coefficient as $b_{\max}$ is likely to be near 200. However, when the number of peaks is fixed, the number of buckets b_{max} within each peak grows with the amount of data, and the algorithm exhibits its characteristic quadratic increase in running time.

Also shown are the running times for the census datasets. Our admittedly subjective evaluation is that this data behaves more like fixed-peak data, hence exhibits quadratic running time. Also striking is the amount of time required for finding the best 5 intervals on the fewer than 2000 records of the forestry data set. Referring back to Figure 3.4 we see that the data has a single peak, and it appears that not much can be gained by partitioning the peak into five intervals so as to exclude some small amount of "unconfident" data. We suspect that the algorithm spends a lot of time fighting the law of diminishing returns. This view is supported by our sampling results in the next section, where most of the gain in support can be obtained by looking at relatively few records.

Another view is given in Figure 3.6, where for a fixed data size, runtime is plotted against the number of peaks in the synthetic data set. (The census and uniform random lines are plotted for comparison, and do not vary.) As the number of peaks increases for a fixed data size, the number of domain values within a peak decreases ($b_{\max}$), and the running time decreases quadratically, consistent with the complexity bounds of the RS algorithm.

Finally, Figure 3.7 shows the effects of varying the demanded minimum confidence, while holding the other parameters fixed. Rastogi and Shim noted that as the minimum confidence threshold approached the mean confidence (.5) for their uniform random data, the runtime increased dramatically. This phenomenon holds as well for our synthetic patterned data, as well as for the census data (which had mean confidence .63, explaining why the curve is shifted). It seems reasonable that for any data set there would be many possible feasible intervals with confidence close to that of the mean, so setting θ near the mean is inviting an algorithm to consider perhaps far more possibilities than is practical, with perhaps only nominal gain in support.

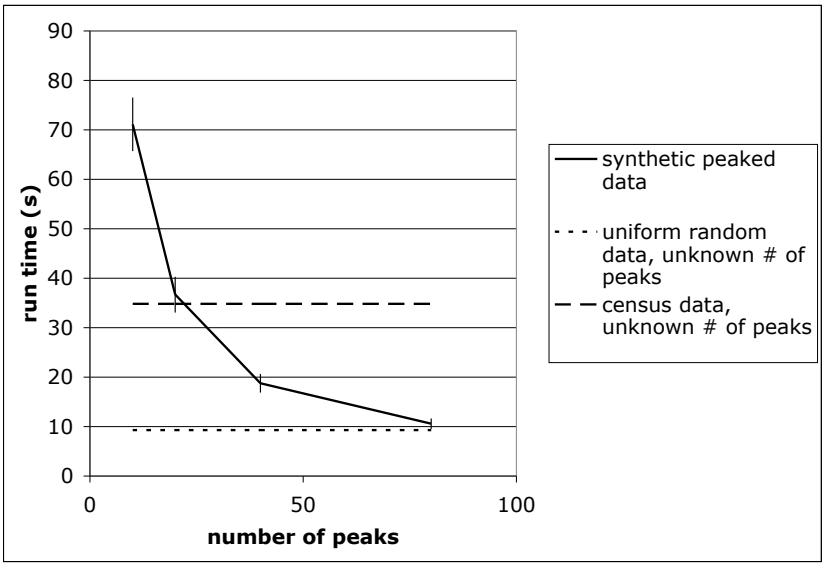

Figure 3.6: Number of peaks vs runtime

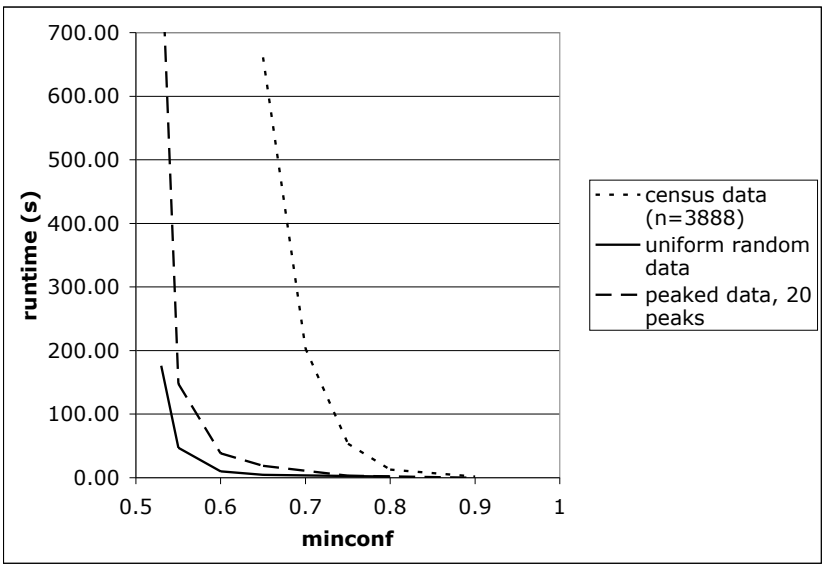

Figure 3.7: Minimum confidence vs. runtime

3.3 Sampling

We consider sampling as an alternative means for efficiently handling data with large domains and for which the divide and conquer algorithm offers no significant speed up.

While a worst-case quadratic (or any polynomial) time algorithm is theoretically acceptable, in data mining applications it is often impractical to implement an algorithm which requires more than a fixed number of scans of the data. Absent a linear time exact algorithm, we turn to sampling to reduce the problem size, and, when we run the RS algorithm on the sample, thereby reduce the computation time.

The burning question is, how many examples do we need to assure that the support of an optimal solution on our sample is likely to deviate only slightly from the support of an optimal solution on the whole data set? (There is also the practical question of how to efficiently sample from a large database; this problem has been well studied, e.g., Vitter's work on reservoir sampling [39] and follow-ups. We assume such an implementation and focus on the sample size.)

There is another, more subtle benefit, offered by sampling that warrants discussion. Besides a natural method for reducing the size of the data set, sampling can be used as a means of data summarization, and as such, can help prevent an algorithm from wasting time on idiosyncracies of data in order to eke out a slight improvement in quality at the expense of a significant increase in running time. Our data summarization technique closely follows that of Fukuda et al. [12] on unsorted data, whereby a random sample is selected from the data set, sorted, and then used as a set of bucket boundaries into which the *original data* is inserted. That is, the coarsened data set includes a bucket for each sampled point, together with a new bucket for every interval between sampled points. They show empirically that roughly equi-height buckets are obtained from relatively small sample sizes, and prove that this is sufficient to bound the error in their problem. Instead, we directly derive bounds on the sample size sufficient to solve the max-support-min-confidence problem within error

tolerance ϵ. (Another difference is that in [12] this technique is not employed if the data is originally sorted, as their problem admits a linear-time algorithm. We employ the technique regardless of whether or not the data is sorted because the quadratic algorithm RS may necessitate data reduction.)

The RS algorithm requires that the input data of size n be sorted by the numerical attribute of interest. If it is not, an additional $O(n \log n)$ time cost is incurred. Sampling is used not only as a way to address the issue of worst case quadratic performance of RS on large data sets, but also, as in [12], to allow for an $O(n \log s)$ semi sort of the data, where s is the sample size. Our sample bounds will give the sample size necessary to assure that this semi-sort of the data does not compromise the solution quality (or at least it quantifies the compromise). Note that our bounds on sample size are not dependent on the domain size n.

We choose a dense enough sample so that, with high probability, any interval containing more than some small, user specified, amount of data is likely to be sampled, and thus, when the actual data is compiled into the new buckets, no interval's support deviates from its original support by more than this small amount. Then, this new set of bucketed data is used as input to algorithm RS. The choice of parameters gives the user a tradeoff between the error that she is willing to tolerate, and the amount of time required by the RS algorithm on the sample.

Following the derivation, we will compare our bounds and approach with those obtained earlier in related settings, e.g., Toivonen [38], and Zaki et al. [41].

Let D denote the original data, $|D| = n$, and let B denote the bucketed data determined by the sample, $|B| = b$. Create B as follows: Select a set of b buckets randomly according to the support distribution in the original data set. Sort the buckets according to the value of the numerical attribute labeling the bucket (if original data was unsorted). Use the bucket boundaries of the sampled set of buckets as a set of bucket boundaries for B. Insert the original data into the new buckets. This requires $O(n \log b)$ time if original data is unsorted, $O(n)$ time, otherwise, and results in at most $2b + 1$ total buckets. Note that there are two

kinds of buckets: those which were elements of the sampled set, and those which represent the accumulation of the data between sampled buckets.

Definition 11. An ϵ-*significant* interval, I, is an interval with *support*$(I) > \epsilon$.

We choose a sample of sufficient size from D to assure, with high probability, that a bucket is sampled for every ϵ-significant interval in D. In so doing, we assure (whp) that the buckets representing the accumulation of data between sampled buckets have weight no more than ϵ. Call the resulting coarsened data B an ϵ-coarsening of D.

Lemma 12. Given an independent uniform sample S of size s from a relation R containing n rows, if

$$s \geq \max(\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{16}{\epsilon}\log\frac{13}{\epsilon})$$

then $Pr(\exists$ an ϵ-significant interval I with $I \bigcap S = \emptyset) < \delta$.

Proof. The proof follows from the observation that the VC-dimension of the class of intervals is 2, and the sufficient sample size bounds given by Blumer et al. [3]. □

Consider any interval I on the original data D and notice that the support of I can be estimated on the bucketed data B by selecting the largest interval I', $I' \subseteq I$, so that the endpoints of interval I' fall on bucket boundaries. Then, *support*(I) − *support*$(I') < 2\epsilon$, since the ϵ error can occur at each of the 2 endpoints. Now, let $I = \bigcup I_k$ be a k-interval, and let $I' = \bigcup I'_k$, where $I'_k \subseteq I_k$, are the intervals of largest support whose endpoints fall on sampled bucket boundaries, as in the single interval case. Then, because there are k intervals, *support*(I) − *support*$(I') < 2k\epsilon$. Thus we have:

Lemma 13. Given a data set D, and B, an $\frac{\epsilon}{2k}$- coarsening of D, for any k-interval I on D, and I' on B with $I' = \arg\max_{i \subseteq I}$ *support*(i), *support*(I) − *support*$(I') < \epsilon$.

We have assured that, with probability more than $(1 - \delta)$, the support of any k-interval can be computed on our coarsened data set with error no more than ϵ. That is, for association rule $F : A \in I \implies C$, *support*$(A \in I)$ can be approximated by some interval I' on the

set of buckets B so that $\mathit{support}(A \in I) - \mathit{support}(A \in I') < \epsilon$. Notice that the argument holds for the condition $A \in I \wedge C$ as well. That is, there exists an interval $I' \in B$ such that $\mathit{support}(A \in I \wedge C) - \mathit{support}(A \in I' \wedge C) < 2\epsilon$. This observation aids in analyzing the error in estimating confidence.

Finally, we are in a position to bound the error incurred by running the RS algorithm on our coarsened data set B. The optimal solution to the problem is a measure of support. There are two types of error in support we can incur. First, we may overlook small intervals. We have bounded the magnitude of this type of error by ϵ. Second, algorithm RS searches among intervals exceeding the minimum confidence threshold for the optimal solution. Danger lies in the case we fail to consider some sufficiently confident interval because we underestimate the confidence of that interval on the bucketed data. If an interval is actually confident, and we cannot detect it, we are at risk of tossing out large chunks of support. Instead of quantifying this neglected support, we create a new, slightly lower confidence bound that these deficient intervals will almost certainly meet, and so they will be considered by the algorithm for inclusion in the optimal solution to the problem. In the next lemma we show that any confident interval on the data set D can be approximated by an interval of slightly lower confidence on the coarsened data set B. Hence, the theoretical results show that for a suitably smaller confidence threshold (which necessarily depends on the optimal support), with high probability only the first kind of error (where small intervals are overlooked) can occur.

Lemma 14. For any k-interval I on D, let k-interval $I' \subseteq I$ be the largest subinterval of I on B. Suppose the minimum confidence threshold is θ, and that $\mathit{support}(I) - \mathit{support}(I') < \epsilon$. Then, with probability greater than $1 - \delta$,

$$\mathit{conf}(I) \geq \theta \implies \mathit{conf}(I') \geq \theta - \frac{\epsilon(1-\theta)}{\mathit{support}(I')}.$$

Proof. Suppose

$$conf(I) = \frac{support(I^+)}{support(I)},$$

and

$$conf(I') = \frac{support(I'^+)}{support(I')}.$$

Then equivalently, we show that if

$$support(I^+) - \theta support(I) \geq 0$$

and

$$support(I'^+) - \theta support(I') < 0,$$

then

$$support(I'^+) - \theta support(I') > \frac{(1-\theta)\epsilon}{support(I')}.$$

Immediately we have the following chains of inequalities:

$$support(I'^+) < \theta support(I') < \theta support(I) < support(I^+),$$

and

$$support(I^+) - support(I'^+) \leq support(I) - support(I') < \epsilon.$$

Let $\theta(support(I^+) - support(I'^+)) = \theta\gamma < \epsilon\gamma$. Then

$$\theta support(I') - support(I'^+) < \epsilon - \gamma\theta < \gamma - \gamma\theta < \epsilon(1-\theta).$$

Multiplying by (-1) and dividing by $support(I')$ gives the result. □

Here's an intuitive explanation: The fact that the original region meets minimum confidence is an indication that the average confidence over the region is no less than the threshold.

Since the sampled portion of the region is deficient, the unsampled region must have excess confidence. How deficient can the confidence of the sample be? At most, the unsampled region has $(1-\theta)\epsilon$ excess, where the $(1-\theta)$ arises because we are measuring excess above θ. This excess, in terms of average confidence for the sample, must be normalized by the support of the sample. In effect, we are redistributing the excess across a region of size $support(I')$.

The previous lemmas combine to give the following bounds demonstrating that small error (ϵ) in total support can be achieved on a sample if a suitably smaller value of confidence (depending on the support) is chosen. Our empirical results demonstrate, however, that this reduction in the minimum confidence θ is not necessary in practice.

Theorem 15. Let $RS(D,\theta)$ denote the maximum support k-interval exceeding confidence θ on data set D. Let $B = \{b_1, b_2, \ldots, b_m\}$ be an $\frac{\epsilon}{2k}$-coarsening of D. Then

$$support(RS(D,\theta)) - support(RS(B, \theta - \frac{\epsilon(1-\theta)}{support(RS(D,\theta))})) < \epsilon.$$

These analytic results imply a reasonable practical approach to sampling, with the only complication arising from the fact that the error in confidence depends on the support of the interval of interest. The sampling bounds are independent of the size of the domain, and thus, tradeoffs are simply between sample size and accuracy. In the next section we demonstrate that convergence to optimal solutions occur surprisingly quickly across different data sets.

The bounds we obtained are slightly different than those given by Toivonen [38], and Zaki et al. [41], for example. In both of these papers, a goal is to estimate the support of an itemset so as to determine with high probability whether or not it qualifies as "frequent" by meeting a minimum support threshold. Toivonen avoids the error of missing frequent itemsets by setting the threshold lower than that desired, and arguing using Chernoff bounds that with high probability, a truly frequent set would not have observed frequency less than this smaller threshold. Zaki et al. similarly compute Chernoff bounds for sample size

sufficient to accurately estimate support regions, and demonstrates that the bounds are very conservative, and many fewer samples are needed in practice.

In each of these investigations, dependence on the error parameter ϵ is quadratic (that is, there is a factor of $\frac{1}{\epsilon^2}$), whereas our bound improves this by relying only linearly on $\frac{1}{\epsilon}$. The key point is that we are using sampling not to obtain uniformly good estimates of the support of intervals, but rather to select admissible *endpoints* of intervals. The *actual* support of these intervals are computed exactly by a linear scan of the original data. So, the error induced by sampling is not from inaccurate estimation of support, but rather solely from the necessity of representing an arbitrary interval using only "admissible" intervals (with endpoints in the sample).

An additional twist in our setting is that we must not only obtain an estimate of the frequency of an interval (support), but also the confidence, which is a ratio of the number of "good" points in the interval to the total number of points in the interval. Unless the interval is well-sampled, no reasonable guaranatee can be obtained. We circumvented this problem by arguing that low support regions (where we cannot accurately estimate confidence) cannot significantly affect the overall solution.

3.3.1 RS Algorithm run on sampled data

Our sampling experiments were conducted as follows: for a data set D, and for sample size s and for 30 repetitions, a sample of s buckets was randomly selected from D and the optimized support set was found on the sample. The maximum support discovered among the 30 runs is compared to optimal for the data set in Figure 3.8, Figure 3.9, Figure 3.10, and Figure 3.11.

All of our experiments demonstrate that convergence to the optimal support is quite rapid for all data sizes and independent of the variability in our synthetic data. The results are remarkably consistent. The same results are observed for the census data, as well as for the forest cover data.

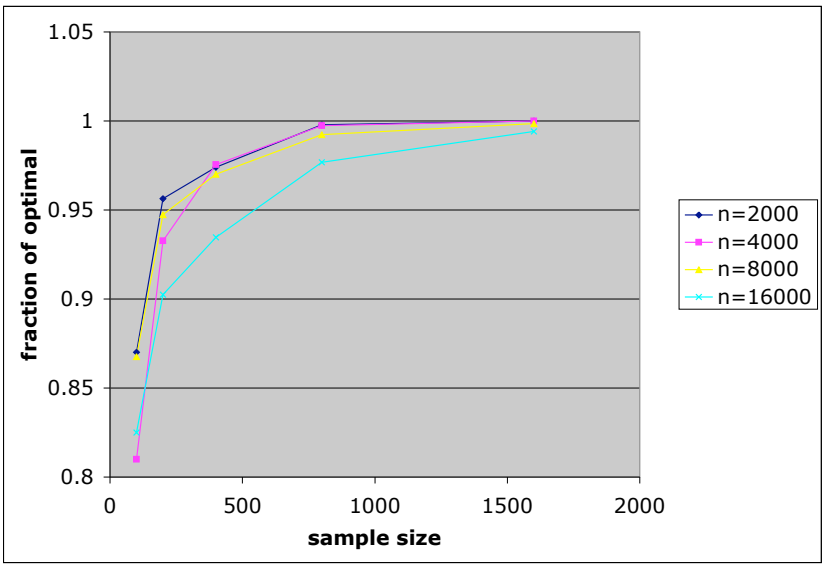

Figure 3.8: Synthetic data sample convergence. Fixed number of peaks = 40.

These experimental results are consistent with the theoretical results derived in the previous section. Indeed, the theory promises that convergence to optimal for a particular level of minimum confidence, θ, will occur, given a relaxation in θ. Our experiments indicate that such a relaxation is unnecessary.

When viewed together, the theoretical and experimental results demonstrated here offer sampling as a reasonable approach to data reduction in the case of optimized support association rule finding. The theoretical results suggest the appropriate tradeoffs between accuracy and sample size to be considered by the practitioner.

3.4 A Linear Time Approximation

Fukuda et al. in [12], when they initially formulated the idea of optimized association rules on a numerical attribute, solved the max-support-min-independent-confidence problem for

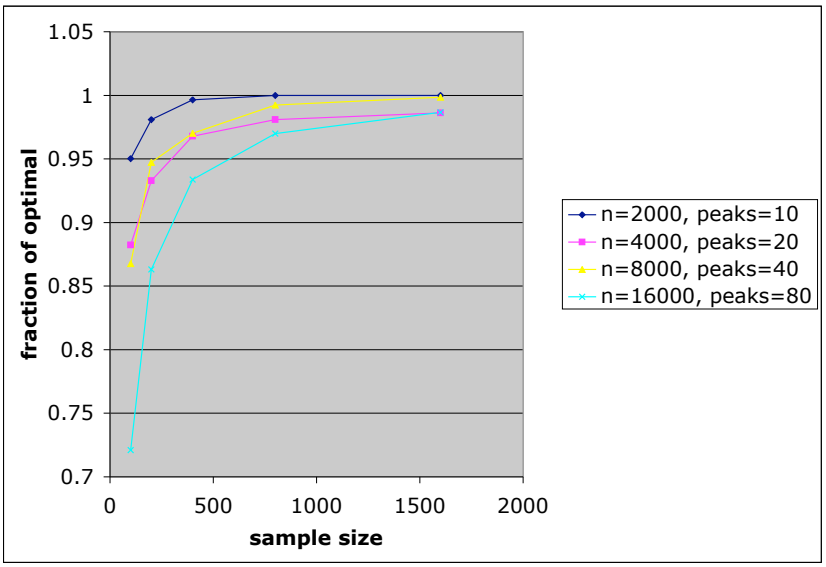

Figure 3.9: Synthetic data sample convergence. Fixed peak width $= \frac{n}{200}$.

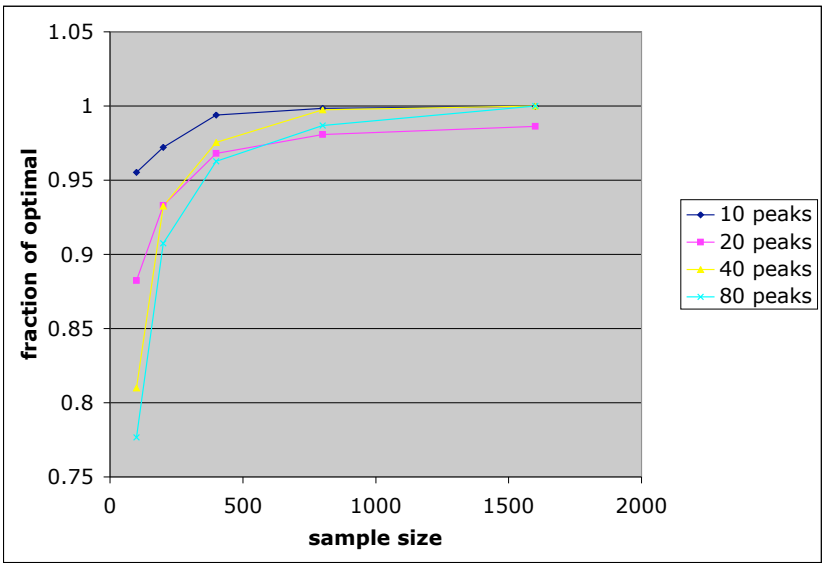

Figure 3.10: Synthetic data sample convergence. Fixed data size $n = 4000$.

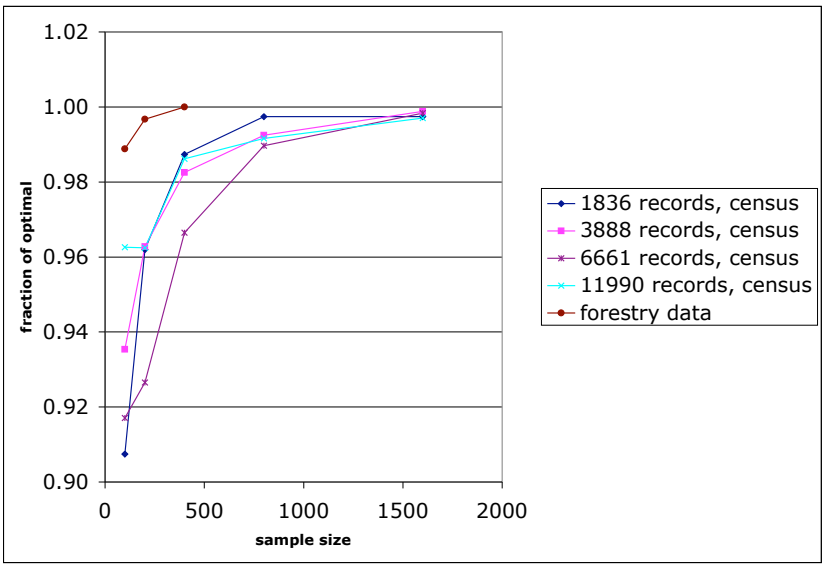

Figure 3.11: Census data sample convergence.

$k = 1$ interval in $O(n)$ time. We modify and extend the Fuduka algorithm, summarized below, to greedily select the k best intervals from among a candidate list of no more than n intervals. The running time of our algorithm is $O(nk)$, and we can guarantee that our algorithm achieves at least 1/3 of the optimal support. Furthermore, we show that no stronger guarantee is possible for this greedy approach.

A brief summary of Fukuda's Algorithm (FA):

If θ is the minimum confidence threshold, The goal of the algorithm is to find an interval (s, t) that satisfies $\mathit{conf}(s, t) \geq \theta$ and that maximizes $\mathit{support}(s, t)$.

- A linear pass of the data suffices to create a list of all possible leftmost points s of the optimal interval (s, t). This part of the algorithm depends on the fact that if s is such a candidate endpoint, then no interval whose right endpoint is $s - 1$ can have confidence exceeding θ. To see that this is true, suppose such a confident interval $(r, s - 1)$ exists,

and observe that in that case, the interval (r, t) would also exceed θ, and would have greater support than (s, t), making it a bad candidate.

- Another linear pass alternating between the data and the list of left endpoints, finds, for all s in the left endpoint candidate list, the largest index t such that $s \leq t$ and $\mathit{conf}(s, t) \geq \theta$. Call this largest index $\mathit{top}(s)$, for each s. That is, $\mathit{top}(s) = \max\{t : s \leq t, \mathit{conf}(s, t) \geq \theta\}$. The pairs $(s, \mathit{top}(s))$, ordered by s, are a list of candidate intervals for maximality. This backward scan through the list of candidate left endpoints and the list of indices is a linear scan (rather than quadratic) because of a lemma, proven in [12], that states if $s < s'$ are both candidate left endpoints, then $\mathit{top}(s) \leq \mathit{top}(s')$. This characteristic also assures that our list is sorted both by left interval endpoint, and by right.

- Finally, the maximum support interval is chosen by a linear scan from among all $(s, \mathit{top}(s))$ intervals.

A linear scan of the data produces the desired result in each of the three steps, for a total running time of $O(n)$.

Our goal is to find k intervals, each of whose confidence exceeds the minimum confidence threshold, and whose total support is maximal.

Suppose the data is contained in an array $A[\cdot]$, each of whose elements consist of a support and confidence level for the numerical value i, denoted by *supp* and conf. That is, $A[i].\mathit{supp}$ is the number of occurrences of numerical value i, and $A[i].\mathit{conf}$ is the number of *those* that also satisfy the boolean condition of interest. Then the confidence of element i is just the ratio of $A[i].\mathit{conf}$ to $A[i].\mathit{supp}$. Further, assume the data has been preprocessed into another array $B[]$, also with conf and *supp* fields, whose i^{th} element, consisting of $B[i].\mathit{supp}$ and $B[i].\mathit{conf}$ contains the partial sums $\sum_{k=1}^{i} A[k].\mathit{supp}$ and $\sum_{k=1}^{i} A[k].\mathit{conf}$, respectively. This preprocessing step allows computation of support and confidence of any interval to be done in constant time.

The motivation for our algorithm is the observation that, if we are searching for k intervals, and if we start by greedily selecting the interval of maximal support from the candidate list, many of the remaining intervals in the list of candidates returned in step two of FA are valid candidates for successive greedy selections. Let $(s', top(s'))$ be the greedily selected interval, and remove it from the candidate list. Then, since the k intervals in the solution must be disjoint, any interval in the candidate list with an endpoint in the interval $(s', top(s'))$ is now invalid. We need not be concerned with intervals whose left endpoint is to the left of s' and whose right endpoint is to the right of $top(s')$ because if such an interval existed, it would have been chosen as best by the greedy algorithm. There are two steps to repairing the list:

- Any interval whose left endpoint is in the interval $(s', top(s'))$ is removed from the candidate list. The removal is achieved via a linear scan of no more than $top(s') - s' + 1$ steps.

- We repair intervals whose right endpoint (only) is in $(s', top(s'))$, by recomputing a new right endpoint, if possible. Let $(s, top(s))$ be such a violating interval. Then, though $(s, top(s))$ is no longer a candidate interval, there may be some other right endpoint for s so that $(s, top(s))$ exceeds θ, and that should thereby be considered a candidate interval. We recompute $top(s)$ by employing step 2 of FA, starting the backward scan for $top(s)$ at $s' - 1$, since that is the rightmost possible location of $top(s)$. Again, this requires at most $top(s') - s' + 1$ steps, since if s is farther to the left of s' than that, $(s, top(s))$ would have been a better original greedy choice.

Our algorithm, then, makes a greedy choice and then updates the candidate list, k times.

The time spent updating the candidate list is proportional to the magnitude of the greedy selection at each step. Since the solution for k intervals is bounded by $O(n)$, the updates take only amortized linear time. The search through the candidate list for each of the k iterations requires $O(n)$ time, so the total running time of the algorithm is $O(nk)$.

Algorithm GAR(B,k):

Input: $B[1 \ldots n]$, described above, and k the number of intervals to find.

Output: k disjoint intervals $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ from the interval $[1 \ldots n]$ each of which have confidence exceeding the minimum confidence threshold, $\boldsymbol{conf}(s_i, t_i) \geq \theta$.

1. Run the first two steps of FA to get a list of intervals whose elements are $(s, top(s))$, each of which is confident. Note that the list is ordered by s.

2. For k iterations:

   In a linear scan extract the interval of maximum support, and report it. Fix the $(s, top(s))$ list so that it contains the list of valid candidates for the next iteration.

Figure 3.12: Find All Max Claims Algorithm

Theorem 16. Suppose $|Opt(I,k)|$ is the support achieved by optimally selecting k intervals on interval I. Then the support of the intervals selected by greedy algorithm GAR, $|GAR(I,k)|$, is given by:

$$|GAR(I,k)| \geq \frac{1}{3}|Opt(I,k)|.$$

Proof. Let $s_1, s_2, \ldots s_k$ be the optimally selected intervals arranged in decreasing order of support, and let $g_1, g_2, \ldots g_k$ be the corresponding sequence of greedy choices, and let $|s_i|$ and $|g_i|$ be the support of those choices. Suppose $k = 1$, then $|g_1| = |s_1|$, and the greedy choice is optimal. Now, suppose inductively that, given a disjoint set of intervals I and number of intervals k, $|GAR(I,k)| \geq \frac{1}{3}|Opt(I,k)|$. Now consider running $GAR(B, k+1)$, where B is some set of disjoint intervals from I. The following cases apply:

- Greedy selection g_1 is disjoint from all selections in the optimal solution. Inductively, with k choices, $|GAR(I - g_1, k)| \geq \frac{1}{3}|Opt(I - g_1, k)| \geq \frac{1}{3}\sum_{1 \leq j \leq k} |s_{i_j}|$, for any choice of k intervals from among the s_i. Furthermore, since g_1 is the first greedy choice, it is at least as big as the s_i not included in the sum.

- Now suppose greedy selection g_1 is not disjoint from the set of s_i. Then we can certainly approximate the non-intersection intervals, by inductive hypothesis:

$$|GAR(I - g_1, k)| \geq |GAR(I - g_1 - \{s_i : s_i \cap g_1 \neq \emptyset\}, k)|$$

$$\geq \frac{1}{3}|Opt(I - g_1 - \{s_i : s_i \cap g_1 \neq \emptyset\}, k)|$$

$$\geq \frac{1}{3}|Opt(I - g_1 - \{s_i : s_i \cap g_1 \neq \emptyset\}, m)|, m \leq k.$$

  On the interval of overlap, the total support can be no more than 3 times the greedy support, because if it were, then there would have been a larger greedy choice, corresponding to a selection that overlapped the greedy choice on an endpoint. In other words,

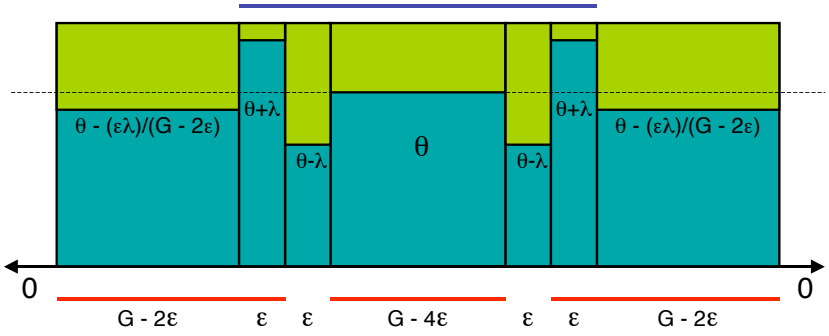

Figure 3.13: Greedy meets its bounds.

$$|GAR(I, k+1)| \geq |g_1| + |GAR(I - g_1, k)|$$

$$\geq |g_1| + |GAR(I - g_1 - \{s_i : s_i \cap g_1 \neq \emptyset\}, k)|$$

$$\geq |g_1| + \frac{1}{3}|Opt(I - g_1 - \{s_i : s_i \cap g_1 \neq \emptyset\}, m)|, \text{ for } m \leq k,$$

$$\geq \frac{1}{3}|Opt(I, k+1)|,$$

since $|g_1|$ is greater than any other single interval, and since $3|g_1| \geq |\{s_i : s_i \cap g_1 \neq \emptyset\}|$,

□

The following example demonstrates that no tighter analysis of our algorithm is possible. In the limit, the example provokes $|GAR(I, k)| = \frac{1}{3}|Opt(I, k)|$.

In figure 3.13, the optimal solution is denoted by bars on the bottom of the interval, and the greedy solution is denoted by a single longer bar along the top. Suppose the greedy

choice has support G, and that the dashed line represents minimum confidence threshold θ. The supports of the optimal intervals are as shown in the diagram. Then, the ratio of greedy to optimal is given by the expression

$$\frac{GAR}{Opt} = \frac{G}{3G - 6\epsilon}$$

whose limit as $\epsilon \to 0$ is $\frac{1}{3}$.

Chapter 4

Maximal Boasting

In this chapter we consider a novel variant of the problem of finding optimized association rules. Our problem involves the search for useful trends in time-stamped performance data that rely on ordinal (ranking) information against a population.

For example, suppose a company has thirty-seven branch stores, and is interested in how they have been performing, relative to each other, over the last several years. Sales data for each store could be transformed to ordinal values giving rank-among-other-stores as a function of year. Then the value of a store to the company might be summarized by "branch A had highest sales in 4 out of the last 7 years", or "branch B had sales among the top quantile in 8 of the last 10 years". Such information is somewhat qualitative. Information about actual percentage of company-wide sales is not present, but other quantitative information is, which carries possibly more actionable information than a single value might.

Another example might involve decisions to be based on medium-term (not real-time because our methods are too slow) statistics about server usage. Quickly determining which servers most often provided search results ranked among the top three in utility provides information on how better to focus resources in response to future queries.

Our problem originates from the following real, though seemingly whimsical, example:

Take a walk through your favorite wine shop. In addition to advertisements proclaiming "a smoky bouquet with a berry finish," or some such collection of flowery adjectives, you will notice advertisements making claims about the sustained quality of a particular wine,

based on adjudicated rankings from a variety of wine competitions. Indeed, claims of the form,"Our shiraz was ranked 4 or better in 3 of the last 5 years!" are common.

The goal of this work is to examine such claims, formalize them, and to determine optimality. Finally, we make a surprising connection between these boastful claims and optimized association rules as defined by Fukuda et al. [12].

There appears to be scant work that relates directly to this problem. Several authors [5, 13, 27] have considered association rules for ordinal data. Such a rule is of the form "when attributes $\{a_1, \ldots, a_n\}$ are present in the data, then $P(a_1, ..., a_n)$ holds", where P is a relational predicate defined only on the ordinal values, for example, "$a_1 \leq a_3 \wedge a_2 \geq a_7$". Thus, the problem is similar to traditional association-rule problems, but here the set of attributes is extended to include ordinal relationships among the data. The search is over a large set of data, to find ordinal relationships that hold significantly often. Results have been used to find errors in data (ordinal relationships that hold frequently help to point out possible anomalous data when these relationships are violated), and also as a means of minimizing the vast number of (traditional) association rules that might otherwise be found on domains with numeric values.

This contrasts with our problem, where we have a single data stream, and seek strong summative claims. In some sense, our problem might be viewed as the "optimized association rule" version of finding ordinal association rules: Rather than the complexity arising from searching among possible attributes to be included in an unknown pattern, all relevant attributes are known (in this case, every moment in time is relevant). The problem is to find instantiations of a known rule template that maximize some objective function.

The other aspect of our problem that is not addressed by that of finding association rules over ordinals is its treatment of the different attributes as points in time. In that vein though, there has been an enormous amount of work (see, for example, summaries [22, 23]) on mining time-series data for periodic patterns, episodes, subsequences, or other patterns that are germane to temporal data. Most often, the goal is to find patterns arising from

causal relationships among the data, and the time ordering is critical in identifying candidate patterns. Again, our goal is substantially different. We do not seek particular patterns woven through the time sequence whose ordering presumably has some significance arising from the temporal relationship of the events. Instead, the timeline is used only as a discrete parameter for a single attribute that gives ordinal values reflecting performance relative to other entities. Again, rather than finding all frequently occurring patterns, our goal is to summative information that is maximal in the senses to be described below.

The data for the problem consists of at least a rank variable, and a unique ordered time variable, $D = \{rank, time, \ldots\}$, with $|D| = n$. Without loss of generality, we can consider the time variable to consist of the ordinal values 1 through n. We can thus define a function on the data $rank(t)$ for $t = 1, \ldots n$, that reports the value of the rank for the data entry whose time value is t. A claim, based on the data, is a triple $\langle r, x, t\rangle \in N^3$ corresponding to the statement "our item was ranked r or better in x of the last t years." Given this meaning, and given a particular data set D, not all claims are true.

We define accumulation functions $f_r(t) = |\{t' : rank(t') \leq r, t' \leq t\}|$. That is,$f_r(t)$ is just the number of years in which the object meets or betters rank r in the time span $[0, t]$. Some characteristics of $f_r(t)$: for any r, $f_r(t)$ is monotonically non-decreasing in t. Furthermore, if $r < s$, $f_r(t) \leq f_s(t)$. That is, we have a non-decreasing family of non-decreasing functions.

Definition 17. A *valid* claim is one of the form $\langle r, x, t\rangle$, where $x \leq f_r(t)$.

We investigate the maximal claims of this kind that can be made about a particular object.

Depending on the application or interpretation of the problem, different criteria may be used to evaluate the relative merits of one claim versus another. As such, there are many partial orders one can choose for making comparisons. We select and analyze two different partial orders.

4.1 A First Partial Order

Definition 18. Define two claims $C_1 = \langle r_1, x_1, t_1 \rangle$, and $C_2 = \langle r_2, x_2, t_2 \rangle$. The two claims are *equivalent*, $C_1 = C_2$, if $x_1 = x_2$, $r_1 = r_2$, and $t_1 = t_2$. The partial order we define specifies that $C_1 >_1 C_2$ (read C_1 dominates C_2, or C_1 is better than C_2,) if $C_1 \neq C_2$ and $r_1 \leq r_2$, $x_1 \geq x_2$, and $t_1 \leq t_2$.

So, for example, a claim like "We were ranked 4 or better in 3 of the last 5 years," is better than "We were ranked 4 or better in 3 of the last 8 years," which is better than "We were ranked 4 or better in 2 of the last 8 years," which is better than "We were ranked 5 or better in 2 of the last 8 years."

$$\langle 4, 3, 5 \rangle >_1 \langle 4, 3, 8 \rangle >_1 \langle 4, 2, 8 \rangle >_1 \langle 5, 2, 8 \rangle$$

Note, however, that claims like "We were ranked 4 or better in 3 of the last 5 years," and "We were ranked 4 or better in 4 of the last 6 years," are incomparable.

We solve the problem of finding maximal claims of this type by processing the data in historical order, and asking, at each time, and among reasonable claims for that time, which claims are maximal.

At any time t, there is a large set of valid claims that can be made. Many of these claims, however are not particularly reasonable. For example, suppose that $f_5(20) = 3$. That is there were exactly 3 observations of rank 5 or better in the time span $1 \ldots 20$. We *could* make a claim such as "We were ranked 5 or better in 2 of the last $20 \ldots$," but why would we ever do so? Furthermore, suppose, as before, that $f_5(20) = 3$, and also suppose that a rank of 5 was observed in time $1 \ldots 20$, and that no rank of 6 was observed in that time. While it's true that we could make the claim "We were ranked 6 or better in 3 of the last $20 \ldots$," we know that the same claim could be made about rank 5, which is stronger according to our partial order. In particular, *no* claim about 6 at any time prior to its first occurrence is maximal, since the same claim could be made about rank 5 (or less).

Definition 19. The set of *reasonable* claims at time t, C_t, is

$$C_t = \{\langle r, f_r(t), t\rangle : \exists t' \leq t, rank(t') = r\}.$$

The reasonable claims include only those claims that are of the form $\langle r, f_r(t), t\rangle$, and where an observance of r has occurred. The only claims excluded from the reasonable set are those that are dominated in this way. Hence, the set of reasonable claims contains the set of maximal claims.

We next consider this set of reasonable claims at time t and argue which of them must be maximal. Note that at any time t there is exactly one reasonable claim for each unique rank seen in time $1 \ldots t$, $|C_t| \leq t$. Finally, observe that every claim $\langle r, x, t-1\rangle$ in the set C_{t-1} of reasonable claims for time $t-1$ has a corresponding claim $\langle r, y, t\rangle$ in the set C_t, where $y = x$ if $r < rank(t)$, and $y = x + 1$ if $r \geq rank(t)$.

Let $A_t = \{\langle r, f_r(t), t\rangle : \exists t' \leq t, rank(t') = r, r < rank(t)\}$, be the set of reasonable claims at time t whose ranks are less than $rank(t)$, and let $B_t = \{\langle r, f_r(t), t\rangle : \exists t' \leq t, rank(t') = r, r \geq rank(t)\}$, the set of reasonable claims at time t whose ranks are at least $rank(t)$. Then $C_t = A_t \cup B_t$, and $A_t \cap B_t = \emptyset$.

The set A_t captures all claims in C_t whose corresponding claims from time $t-1$ are not improved by the observation of $rank(t)$. For these, $f_r(t) = f_r(t-1)$. The set B_t shows all claims whose tallies *are* improved by an observation of $rank(t)$. For these, $f_r(t) = f_r(t-1)+1$.

Lemma 20. If $c = \langle r, f_r(t), t\rangle$ is a reasonable claim at time t, $c \in C_t$, and $r \geq rank(t)$ then c is a maximal claim. That is, every claim in set B_t is maximal.

Proof. Suppose c is not maximal. Then, there exists another valid claim $d = \langle s, f_s(u), u\rangle$ so that $d > c$. We assume that d is a reasonable claim, since if it isn't, then it can be replaced by an even better claim, that is reasonable.

By definition of the partial order on claims, $s \leq r, f_s(u) \geq f_r(t)$, and $u \leq t$. But $u \leq t$ implies $f_r(u) \leq f_r(t)$ by monotonicity of accumulation functions. Furthermore, $s \leq r$ implies

$f_s(u) \leq f_r(u)$ since the accumulation functions are a monotone family. Thus, $f_s(u) \leq f_r(t)$, which means that actually, they are equal.

Now, suppose $u < t$. Claim $\langle s, f_s(u), u\rangle$ is a valid claim, and since $s \leq r$, $\langle r, f_s(u), u\rangle$ is a valid claim as well. But since $r \geq rank(t)$, $f_r(t) > f_r(u)$, contradicting $f_s(u) = f_r(t)$. This implies $u = t$.

Finally, we have the claims $d = \langle s, f_r(t), t\rangle$ and $c = \langle r, f_r(t), t\rangle$. Suppose $s < r$. Since d and c are both reasonable, ranks r and s appear sometime in time $0 \ldots t$. But if that is the case, then at time t, we have seen at least one more instance of rank r or better than we have of rank s or better, which contradicts $f_s(u) = f_s(t) = f_r(t)$. So $s = r$. But then, $d = c$, and we are done. Claim c is maximal. □

Next we prove that at time t, no claim with rank less than $rank(t)$ (no claim in A_t) is maximal. Let $c = \langle r, f_r(t), t\rangle \in A_t$, hence $r < rank(t)$. But this implies $f_r(t) = f_r(t-1)$, or the number of instances of rank r or better was not increased by observing $rank(t)$. But the claim $\langle r, f_r(t-1), t-1\rangle$ dominates c and so c is not maximal.

We have characterized the set of maximal claims for each time by B_t, and now we give an algorithm for producing them. The idea of the algorithm is to maintain over time a sorted list of observed ranks, together with the appropriate value of the accumulation function for each rank and time.

The algorithm is shown in Figure 4.1.

Notice that for each t, the algorithm prints exactly the elements in B_t, which was proven to be the set of maximal claims.

In the worst case, and if n is the size of the history, or the size of the database, or the total number of time periods, there are at most $O(n^2)$ maximal claims, and there are at least n maximal claims. Our algorithm is optimal: it runs in time linear in the number of maximal claims, since one claim is delivered on every iteration of every loop.

1. Create an empty list, $C = \{rank, tally\}$, consisting of rank, and accumulation tally fields. This list contains, for each t, an entry for every element of C_t.

2. for each time $t = 1 \ldots n$:

    get $rank(t)$

    while $C.rank > rank(t)$

        $C.tally++$

        output maximal claim $\langle C.rank, C.tally, t \rangle$

        next C

    if $C.rank \neq rank(t)$ then

        insert new C element $\langle r, C.tally + 1 \rangle$

    else

        $C.tally++$

    output maximal claim $\langle C.rank, C.tally, t \rangle$

Figure 4.1: Maximal Claims for $>_1$

4.2 A Second Partial Order

Under the partial order described in section 4.1, such claims as "we were ranked 4 or better in 3 of the last 5 years," and "we were ranked 4 or better in 2 of the last 3 years," were incomparable. In fact, a reasonable alternative viewpoint is that the second claim is stronger because the fraction of the time the ranking was achieved is higher ($2/3 > 3/5$). Our second partial order formalizes this intuition.

Definition 21. If $C_1 = \langle r_1, x_1, t_1 \rangle$ and $C_2 = \langle r_2, x_2, t_2 \rangle$, then $C_1 >_2 C_2$ (or C_1 is stronger than C_2, or C_1 dominates C_2) if $C_1 \neq C_2$, $r_1 \leq r_2$, and $f_{r_1}(t_1)/t_1 \geq f_{r_2}(t_2)/t_2$. In other words, one claim dominates another different claim by having better or equal rank and better or equal fraction of success.

Note that we do not distinguish between claims like "2 or better in 9 of the last 10 years," and "2 or better in 90 of the last 100 years." Under our current metric, these claims are considered equal.

For a given rank r, consider the function $f_r(t)$ (augmented with the value $f_r(0) = 0$), and notice that every function value corresponds to a claim whose fraction $f_r(t)/t$ – and thereby, whose value with respect to the partial order since r is fixed – is the slope of the line through $(0, 0)$ and $(t, f_r(t))$. Call this slope the *cumulative* slope. We could find all the maximal claims whose rank is r by examining every point on $f_r(t)$ and reporting the claim(s) corresponding to the point of maximal cumulative slope on the function. If this is done for all r, in increasing order by r, the maximal claims result. Unfortunately, this naïve implementation has a worst case quadratic running time. In contrast, we offer an algorithm that finds all maximal claims in time $O(n \log^3 n)$.

Our solution requires an efficient data structure for computing $f_r(t)$ for all r and t, together with a data structure that supports efficient dynamic maintenance of convex hulls.

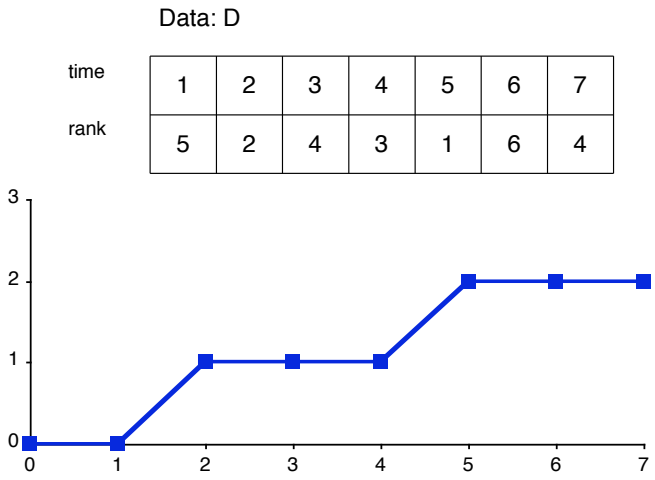

Figure 4.2: Function $f_2(t)$, for the data shown.

Before discussing the convex hull data structure and demonstrating its utility, we must discuss the maintenance of the functions $f_r(t)$. In $O(n \log n)$ time, we build a tree structure that

- allows us to compute the value of $f_r(t)$ for any r and t, and
- demands $O(\log^2 n)$ time every time we use it.

This usage cost can result in a net savings, because we would spend $O(n^2)$ time to create a table of $f_r(t)$ values whose elements are available in constant time. As long as we can satisfactorily bound the number of times we employ this function, we are happy.

4.2.1 The function data structure.

Without loss of generality assume $n = 2^k$ for some k. The data structure is a complete binary tree consisting of n leaves, where n is the size of the (augmented by 0) data. If the

leaves of the binary tree are considered in level order, the t^{th} leaf is labeled by the rank assigned at time t, $rank(t)$, as given in the data (assign the rank at time 0 to be some very awful large rank M).

If $height(v)$ is the height of node v in the tree, then every internal node v contains an array whose length is equal to $2^{height(v)-1}$. This particular array size is chosen to correspond to the number of leaves in the left subtree of v because the array will have an entry corresponding to each such leaf. We refer to these arrays as the *rank-tally arrays*, $T_v[\cdot]$. In addition, the t^{th} leaf node, v, contains a rank-tally array of a single entry.

For internal vertices v, the i^{th} element of the rank-tally array T_v, is defined by the minimal rank r such that $i = |\{rank(t) \leq r : t \text{ is in the left subtree of } v\}|$. That is, the number of leaves in the left subtree of v with rank $T_v[i]$ or better, is i. For the t^{th} leaf v, $T_v[1] = rank(t)$. Another way of characterizing rank-tally array $T_v[\cdot]$ is to say $T_v[i] = \min\{r : |\{t : t \in \text{ left subtree of } v\}| = i\}$. Finally, define a variable "$current_v$" for each array that indicates the most recently inserted array element (initialized to 0).

Lemma 22. Define $P_t = \{v : v \text{ is a node in the path from the } t^{th} \text{ leaf to the root}\}$. For each rank r we define a function on the nodes $g_r(v)$ as follows:

$$g_r(v) = \begin{cases} 0 & \text{if } T_v[1] > r, \\ \max_{T_v[i] \leq r} i & \text{otherwise,} \end{cases}$$

Then $g_r(v)$ is the exact number of leaves in the left subtree of v with rank less than or equal r. For any r, t, $f_r(t) = \sum_{v \in P_t} g_r(v)$.

Proof. We wish to tally the number of leaves in the tree to the left of leaf t, whose ranks are no more than r. Let $v_1, v_2 \ldots v_t$ be the leaves to the left of the t^{th} leaf. Each of the v_i shares a least common ancestor with v_t on path P_t, where v_i is in the left subtree, and v_t is in the right (otherwise, $v_i = v_t$). Since every internal node v records a tally of the number of leaves

in v's left subtree with rank less than or equal to r, for any r, the sum of these tallies on P_t is what we seek. □

Using the tree data structure described above, we can compute $g_r(v)$ in $O(\log n)$ time by doing binary search on the rank tally array for node v. Furthermore, $f_r(t)$ is the sum of $\log n$ terms corresponding to the $\log n$ vertices in $P(t)$. Thus, $f_r(t)$ is computed in time $O(\log^2 n)$.

Data is inserted into the data structure by processing the data in rank order as follows: Let r be the current rank, and let $time(r)$ be the time it occurs. (If $time(r)$ is not unique, that is, if there are multiple occurrences of rank r, simply process each of these in time order.) Let $v_0, v_1, \ldots v_k$ denote the nodes on the path from the $time(r)^{th}$ leaf to the root. Traverse the tree along this path updating the tally-array, if necessary, at each node. If v_{i-1} is the *left* child of v_i, increment $current_{v_i}$ and then insert r into array element $current_{v_i}$. If v_{i-1} is the *right* child of v_i, do nothing. Note that the elements of any tally array are inserted in increasing order by rank. The time taken to build the function tree is thus $O(n \log n)$.

4.2.2 Convex Hulls

The following two lemmas connect the problem of finding maximal claims to that of maintaining a convex hull of a set of points.

Lemma 23. Consider the function $f_r(t)$ for some r, and suppose we are given $H(S)$ the (upper) convex hull, of the set of points $S = \{(0,0)\} \cup \{(t, f_r(t)) : t = 1 \ldots n\}$. Let U_r be the set of vertices on the upper hull, $H(S)$. If

$$(x, y) = \arg \min_{t>0, (t, f_r(t)) \in U_r} t$$

then $\langle r, y, x \rangle$ is a maximal claim.

Proof. The lemma states that the vertex on the upper hull with least t greater than 0, point (x, y), gives maximal claim $\langle r, y, x\rangle$. The upper hull is characterized as a collection of vertices and consecutive sequence of adjacent segments whose slopes are monotonically decreasing, where the first segment has infinite positive slope, and where all points not on the hull are below the hull. The second segment on the hull, in our case the segment whose endpoints are $(0, 0)$ and (x, y), has maximum slope among the rest. Since the slope is y/x, we have our maximal claim. □

Furthermore, we have the following:

Lemma 24. Given a maximal claim $\langle r, y, x\rangle$, all maximal claims with rank r have the form $\langle r, ky, kx\rangle$ for some positive k.

Proof. Suppose $C_2 = \langle r, x, y\rangle$ is a maximal claim of rank r, and suppose there is some other maximal claim $C_1 = \langle r, kx, jy\rangle, k \neq j$. But then, either $C_2 > C_1$ or vice versa since $\frac{jy}{kx} \neq \frac{y}{x}$. This contradicts the maximality of either C_1 or C_2. □

The question, then, is how fast can we maintain convex hulls? We apply a classic data structure from computational geometry for dynamically maintaining convex hulls. The data structure was devised by Overmars and van Leewen in 1981 [29]. Their algorithm computes the convex hull of a set of data points as the union of an upper hull and lower hull. In fact, we are not interested in the entire convex hull structure for the points in a $f_r(t)$ function, but rather we require only the upper hull.

Before giving a technical description of the data structure itself, we describe how it is used: We initialize the hull data structure based on the function $f_0(t) = 0, \forall t$, in $O(n)$ time. Then, for each rank in increasing order, we update the data structure to represent the upper hull of $f_r(t)$ in time $O(\log^3 n)$. Finally, we report the maximal claim by extracting the second point on the hull in $O(\log n)$ time. Repeating this process for n (not necessarily unique) ranks gives a total running time of $O(n \log^3 n)$. Efficient updates to the hull are the key to our efficient algorithm.

As in [29], we represent the convex hull by the sequence of points on the hull sorted by t, in a concatenable queue structure that allows all general queue operations like search, split, and concatenate, in $O(\log n)$ time. Let queue Q_v represent the convex hull of the points in the subtree rooted at v.

Using the notation from [29]: for queue Q, let $Q[k \ldots l]$ denote the queue consisting of the k^{th} through l^{th} elements of Q. For queues Q_1 and Q_2, let $Q_1 \cup Q_2$ denote the concatenation of Q_2 onto Q_1.

To motivate the data structure, we give the following theorem:

Theorem 25. [29] Let $p_1, \ldots, p_n$ be n arbitrary points in the plane, ordered by x coordinate. If the representations of the upper hull of $p_1, \ldots, p_i$ and of $p_{i+1}, \ldots, p_n$ are known for any $1 \leq i < n$, then the upper hull of the entire set can be built in $O(\log n)$ steps.

The fundamental data structure supporting this hierarchical relationship between hulls for the entire data set is a complete binary tree with $n + 1$ leaves. Each internal node v includes the following information:

- $p(v)$, a pointer to the parent of v,
- $left(v)$ and $right(v)$, pointers to the left and right children of v,
- $max(v)$ the largest value of t of the points in subtree rooted at $left(v)$
- $rem(v)$, the part of Q_v that is not also a part of $Q_{p(v)}$,
- $B(v)$ the number of points in Q_v that do belong to $Q_{p(v)}$.

The first two items are the infrastructure of the binary tree. The final three allow reconstruction of Q_v from a segment of $Q_{p(a)}$, and their particular values come from the choice of the concatenation that creates $Q_{p(a)}$ from its left and right children.

Let $Q_{left(v)}$ be the queue representing the hull of the points in the left subtree of v, and let $Q_{right(v)}$ be the queue representing the hull of the points in the right subtree of v.

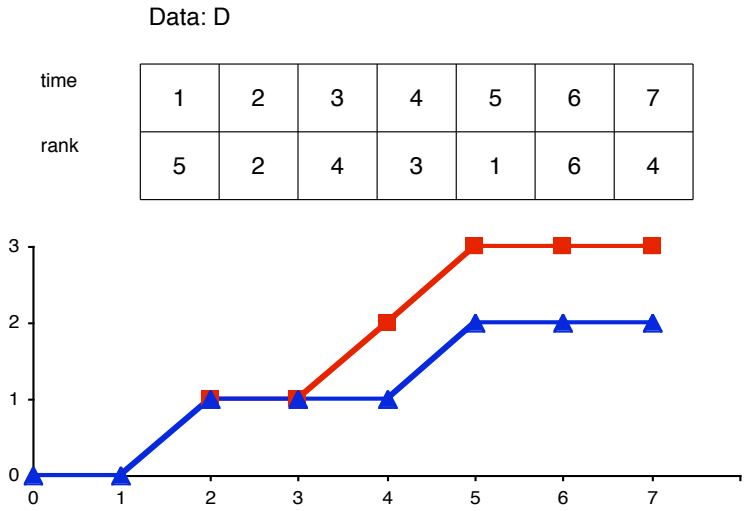

Figure 4.3: Functions $f_2(t)$ and $f_3(t)$, for the data shown.

Then, the queue representing the hull of the points in the subtree of node v consists of the head of $Q_{left(v)}$, concatenated with the tail of $Q_{right(v)}$. $Q = Q_{left(v)}[1 \ldots B(left(v))] \cup Q_{right(v)}[(length(Q_{right(v)}) - B(right(v)) + 1) \ldots *]$. We examine the choice of the head and tail pieces later.

Now how do we use and update this structure? Close examination of the $f_r(t)$ ordered by rank reveals that as the rank increases from r to $r+1$, $f_{r+1}(t)$ is not very different from $f_r(t)$. Specifically, suppose rank $r+1$ occurs at time s, $time(r+1) = s$, then $f_{r+1}(t) = f_r(t)$ for all $t < s$, and $f_{r+1}(t) = f_r(t) + 1$ for all $t \geq s$. That is, at time s, a vertical shift of 1 occurs in $f_{r+1}(t)$. See Figure 4.3.

Consider the shift that occurs between ranks r and $r+1$ in the context of upper hulls. Since the function has changed, the upper hull will likely change. We know, from Theorem 25 that we can create a new convex hull if we have a reliable left and right hull from which to build it. Suppose, without loss of generality, that the shift occurs at a point in the left

```
fixhull(t, v, Q_v) -- t is shift point, v is current node in
tree, Q_v is the queue representing the hull of the points
in the subtree rooted at v.

if left(v)= null  then
    Q_v = enqueue(t)
else
    Q_R = rem(right(v)) ∪ Q_v[B(left(v)) + 1 ... end]
    Q_L = Q_v[1 ... B(left(v))] ∪ rem(left(v))
    if t ∈ subtree rooted at left(v) then
            Q_L = fixhull(t, left(v), Q_L)
    else
            Q_R = fixhull(t, right(v), Q_R)
    Q_v = Patch(Q_L, Q_R, v)
return Q_v
```

Figure 4.4: Hull Update Algorithm

subtree. Then every point in the right subtree is shifted up by 1, and no point's position changes with respect to any other. That means, to recreate the right hull, all we have to do is move the appropriate tail from the parent back down to the right child. (We do not make a copy, but rather we completely deconstruct the queue at the root.) On the other hand, the hull represented by the left subtree, after we recreate it by passing down the appropriate portion of the head from the parent, may now be invalid with respect to the new positions of its points. In turn, *this* hull can be updated if it has an accurate left and right hull. The argument continues recursively, down the tree to the leaves, where the hulls are trivial to reconstruct. We observe that the nodes representing invalid hulls, are precisely the nodes on the path from the point of the shift, through the root.

The update algorithm is Figure 4.4.

Finally, we discuss the hull "Patch" algorithm: the process by which we reconstruct a hull out of a left hull and a right hull that are separated by a vertical line. To update a hull, we create an arbitrary partition in each of its two child hulls and then modify the partition points, using binary search, to achieve convexity. The characteristics of the upper convex hull that we exploit are the monotonic, non-increasing slope of sequential line segments of the hull, and the fact that all the points in the point set must fall below every tangent to the hull. Our discussion, albeit more detailed, closely follows that of Preparata and Shamos [33].

The queue representing the upper hull of a set of points in the plane ordered by x-coordinate, is a set of point labels corresponding to these points in the plane. In our case, the point labels are exactly the x-coordinate of the points. That is, point t in the queue represents point $(t, f_r(t))$ in the plane, for some r. The line segments between adjacent pairs of points in the queue are elements of the hull, though we represent them only implicitly.

Definition 26. Let $t_1, t_2 \in Q$ be the labels of two adjacent points on the upper hull represented by Q, so that t_1 is a point in the subtree rooted at $left(v)$ and t_2 is a point in the subtree rooted at $right(t)$. Then the *bridge* between the two points, denoted (t_1, t_2) is the segment of slope $m(t_1, t_2) = \frac{f_r(t_2) - f_r(t_1)}{t_2 - t_1}$ tangent to both t_1 and t_2.

Definition 27. Let point $t = (t, f_r(t))$ be a point on an upper hull. Define m_t^- and m_t^+ to be the slopes of the left and right hull segments adjacent to t, respectively.

Note that m_t^- and m_t^+ can be computed in $O(\log n)$ time by three calls to the function computing $f_r(t)$.

At each node v in the hull tree, there is an implicit bridge (t_L, t_R) defined by the last point in the hull of $left(v)$, t_L, and the first point in the tail of the hull of $right(v)$, t_R, used to create the hull at v. This bridge must maintain the property of monotonically non-increasing slopes of segments on the upper hull. That is, for bridge (t_L, t_R),

$$m_{t_L}^- \geq m(t_L, t_R) \geq m_{t_R}^+.$$

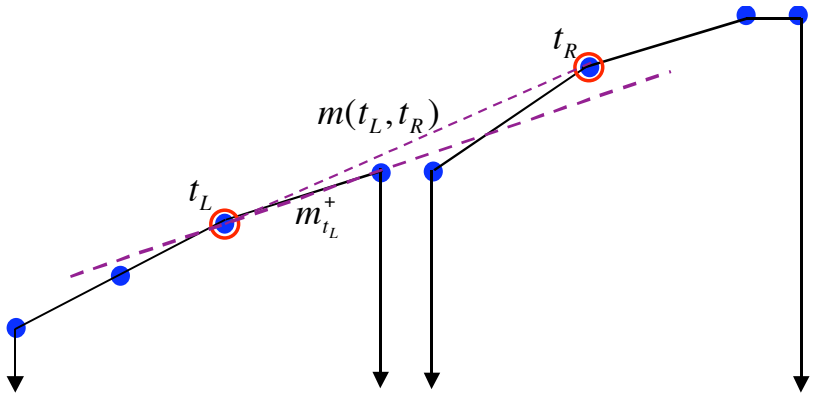

Figure 4.5: The search continues only to the left in the left hull.

Furthermore, to guarantee that the points of both child hulls all fall below the bridge, it must be true that

$$m^-_{t_L} \geq m(t_L, t_R) \geq m^+_{t_L} \quad and \quad m^-_{t_R} \geq m(t_L, t_R) \geq m^+_{t_R}.$$

Given a left hull represented by $Q_{left(v)}$, and a right hull represented by $Q_{right(v)}$, we create Q_v by finding a bridge (t_L, t_R) for which all these slope inequalities hold.

If we choose an arbitrary bridge between $Q_{left(v)}$ and $Q_{right(t)}$, either the selected bridge satisfies the inequalities, or one of the following is true:

- If $m(t_L, t_R) > m^+_{t_L}$:

  As shown in Figure 4.5, we can eliminate every point in the left queue to the right of t_L as a candidate for the left endpoint of the bridge, and continue our binary search in

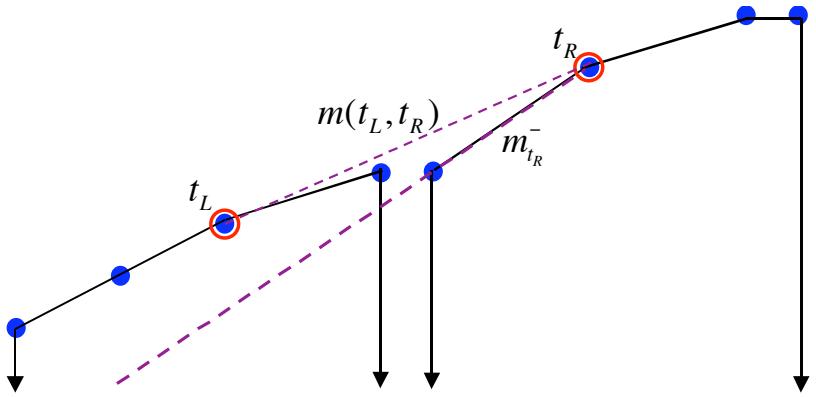

Figure 4.6: The search continues only to the right in the right hull.

the left part of that queue. This is because the point t_R will fall above every tangent to every point on the right.

- If $m^-_{t_R} > m(t_L, t_R)$:

  This case is shown in Figure 4.6. This is the mirror image of the previous case. Here we can eliminate every point to the left of t_R of the right queue, and continue our binary search in the right part of that queue.

- If $m^+_{t_L} > m(t_L, t_R) > m^-_{t_R}$:

  In this case, shown in Figure 4.7 a test must be performed in order to determine which half of which hull we can eliminate. Let t be the rightmost point in the subtree rooted at $left(v)$. This point gives the vertical dividing line between the left and right child

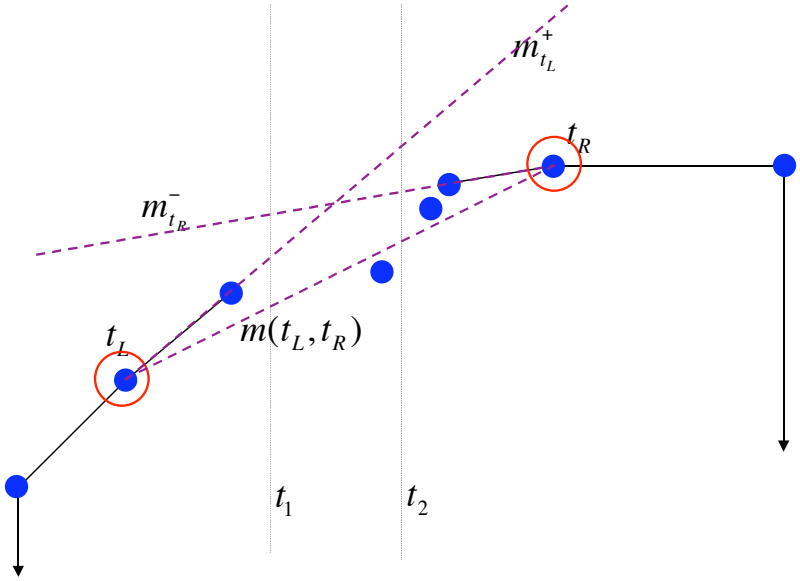

Figure 4.7: If $t = t_1$ then eliminate points to the right of t_R. If $t = t_2$ then eliminate points to the left of t_L.

hulls. Let $\lambda = \frac{t - t_L}{t_R - t_L}$. If

$$m(t_L, t_R) < \lambda m^+_{t_L} + (1 - \lambda) m^-_{t_R},$$

then eliminate the points of Q_L to the left of t_L from consideration. This is because no point to the left of t_L can have a common tangent with any point in the right hull, and therefore, no bridge is possible. Otherwise, eliminate the points to the right of t_R in Q_R. This case is similar. No point to the right of t_R can have a common tangent with a point in the left hull.

Our binary search algorithm then, is to test these slope conditions at each endpoint of the current bridge, eliminate the corresponding interval from the search space, and update the bridge choice to the middle of the remaining intervals. If there is nothing to eliminate, we are done, our bridge has been found. In the worst case, assume that only half of the

$patch(Q_L, Q_R, v)$ `-- returns upper hull` Q_v `by concatenating appropriate portions of` Q_L `and` Q_R`.`

$(t_L, t_R) = search(Q_L, Q_R, v)$
$B(left(v)) = t_L$
$B(right(v)) = length(Q_R) - t_R + 1$
`return` $Q_L[1 \ldots t_L] \cup Q_R[t_R \ldots end]$

Figure 4.8: Patch Algorithm

largest of the left and right hulls is eliminated at each step of the algorithm. The running time is just $O(2 \log n)$ for this double binary search. The details of the algorithm are found in Figure 4.8 and Figure 4.9.

The total running time of $fixhull$ is $O(\log^4 n)$. This can be decreased to $O(\log^3 n)$ by computing $f_r(t)$ when the hull for rank r is updated, rather than a priori. (Due to more recent work on the dynamic maintenance of convex hulls in the plane by Brodal and Jacob [4], and earlier by Chan [6], we suspect that another log factor can be eliminated, either in the amortized sense, or asymptotically, respectively. However, the data structures used in each case are complex enough to obscure the structure of the problem, and to make implementation nearly impossible.)

Given a queue Q representing the convex hull of the points $(t, f_r(t))$, for all t, we can extract the maximal claim from the hull in time $O(\log n)$. We maintain a pointer to the second point on the queue, and then we require $O(\log n)$ time to find $f_r(t)$ so that we can report maximal claim $\langle r, f_r(t), t \rangle$. We imagine this function defined as $getmaxclaim(Q)$.

Figure 4.10 is a summary of our entire algorithm.

The algorithm, as stated here, runs in time $O(n \log^3 n)$, an improvement over the naïve $O(n^2)$ algorithm.

$search(Q_L, Q_R, v)$ `-- returns bridge` (t_L, t_R)

$t_L = midpoint(Q_L)$

$t_R = midpoint(Q_R)$

$m =$ `slope of bridge` (t_L, t_R)

m_L^- = `slope of segment adjacent to` t_L `on left`

m_L^+ = `slope of segment adjacent to` t_L `on right`

m_R^- = `slope of segment adjacent to` t_R `on left`

m_R^+ = `slope of segment adjacent to` t_R `on right`

`if` $m_L^- \geq m \geq m_L^+$ `and` $m_R^- \geq m \geq m_R^+$ `then`

    `return` (t_L, t_R)

`elseif` $m > m_L^+$ `then`

    `return` $search(Q_L[1 \ldots t_L], Q_R, v)$

`elseif` $m_R^- > m$ `then`

    `return` $search(Q_L, Q_R[t_R \ldots end], v)$

`elseif` $m_L^+ > m > m_R^-$ `then`

    $\lambda = \frac{t_R - max(v)}{t_R - t_L}$

    `if` $m > \lambda m_R^- + (1 - \lambda) m_L^+$ `then`

        `return` $search(Q_L[t_L \ldots end], Q_R)$

    `else`

        `return` $search(Q_L, Q_R[1 \ldots t_R])$

Figure 4.9: Search Algorithm

- initialize empty $f_r(t)$ tree and initial hull for $f_0(t) = 0, \forall t$.

- for each rank from 1 to n:

  insert $rank$ into $f_r(t)$

  $Q_v = fixhull(time(rank), root, Q_v)$

  $newmax = getmaxclaim(Q_v)$

  if $newmax >_2$ most recently added claim then

  $claimlist = claimlist \cup newmax$

Figure 4.10: Find All Max Claims Algorithm

4.3 Optimized Confidence Association Rules

Recall the founding work in the area of optimized association rules by Fukuda et al. [12], who posed the following question (among others): Given a single numerical attribute A, and a Boolean condition on records, what is the interval (x, y) in A, whose support meets some minimum threshold, and whose confidence (with respect to the Boolean condition) is maximized? They gave an insightful $O(n)$ algorithm for the problem. Now, suppose that we wish to find such intervals on a single numerical attribute for a *sequence* of n Boolean conditions. Employing their algorithm for each of the n problems requires, in total, quadratic running time.

Before we adapt our boasting algorithm in a straightforward way to address a simplification of the problem, we give a definition.

Definition 28. Consider a sequence of Boolean conditions, b_i, on a data set D, so that $b_i(x)$ is either true or false for every $x \in D$, and $\forall i$. Let $B_i = \{x : b_i(x) = true, x \in D\}$ be the set of records in D satisfying b_i. Then the sequence b_i is *monotonic* if $\forall i$, $B_i \subseteq B_{i+1}$.

The problem we are solving, then, can be stated as follows:

Definition 29. Given a single numerical attribute A, and a sequence of Boolean conditions b_i on the records in the database, if the sequence of Boolean conditions are monotonic, find the

interval in A, $(0, y)$, whose support meets some minimum threshold, and whose confidence (with respect to Boolean condition b_i) is maximized, for each b_i.

We apply a slight adaptation of the boasting algorithm to solve this problem in $O(n \log^3 n)$ time, where n is the number of records in the database. First, we point out that the boasting problem can be posed as an instance of this optimized association rules problem, where the numerical attribute on which we find association rules is the history attribute from the boasting problem, and the ranking conditions such as $(D.rank \leq r)$, when processed in rank order, are a sequence of monotone Boolean conditions on the data. Furthermore, the fraction we are trying to maximize in seeking the maximal claim using partial order $>_2$ is exactly the confidence of the interval. Finally, the boasting problem has no minimum support threshold, so we call the minimum support threshold 0. Since the problems are so similar, we make only minor adaptations to the boasting algorithm to solve the slightly more general problem.

The first generalization applies to the sequence of Boolean conditions. In the case of the boasting problem, the number of records satisfying $(D.rank \leq r)$ changes by 1 as the rank is incremented to $r + 1$. That is, $B_i \subseteq B_{i+1}$ and $|B_{i+1}| = |B_i| + 1$. In the association rules problem, on the other hand, we only require that $|B_{i+1}| = |B_i| + k$ for some $k \geq 0$. This is an easy generalization to accommodate, however, because we follow the boasting algorithm exactly, making a sequence of k updates to the upper hull as in the boasting algorithm, but we only search for a maximal claim (the maximum confidence interval $(0, y)$ for claim B_i) after all k updates to the hull are complete. Note that this modification also allows for non-unique ranks in the boasting problem.

The second generalization results in a simple adaptation of the $f_r(t)$ function in the boasting algorithm to accommodate the minimum support threshold requirement of the more general problem. In fact, the data structure and function $f_r(t)$ are created in exactly the same way as before, the adaptation only occurs in its use. Suppose the minimum support threshold is γ. Define a new function $g_r(t)$ as follows:

$$g_r(t) = \begin{cases} 0 & \text{if } t < \gamma, \\ f_r(t) & \text{otherwise.} \end{cases}$$

Then we use $g_r(t)$ rather than $f_r(t)$ when computing function values. By making this adaptation, we assure that the ranks that occur before time γ are tallied in $f_r(t)$ for all $t > \gamma$, but we also guarantee that no segment of the upper hull in the interval $(0, \gamma)$ will be considered for maximality.

Any sequence of Boolean conditions arranged so as to require $O(n)$ updates to the hull structure can be answered efficiently using our solution. Characterizing a given set of Boolean conditions as bounded in this was is surely a hard problem. On the other hand, if our Boolean conditions are mutually exclusive, that is, no record satisfies more than 1 of the list of conditions, we can answer efficiently. This is an important class of problems since it corresponds to asking Boolean questions about categorical data such as "find the optimal age ranges among those who drive x cars," where x is one of a set of colors.

Chapter 5

Conclusion

Our main results are summarized:

- The problem of (partial) index selection for query cost minimization is defined.
- We give a linear-programming based approximation algorithm that takes as input a space bound B and a collection of m queries to a large database of size M, and in polynomial time outputs a collection of indices using (relaxed) space $O(B \ln m)$ and which has expected query cost at most opt_B, the optimal solution actually meeting the space bound.
- We provide an algorithm that achieves expected query cost ϵM times optimal, finding $O(N \ln m)$ indices using space $O(B \ln m)$, when queries are drawn from an arbitrary distribution as opposed to originating from a fixed workload. Here, N is an additional constraint on the number of indices built. The algorithm needs only to observe $m = \text{poly}(1/\epsilon, B, N)$ random queries.
- We show that the index selection problem is NP-hard, and not approximable unless the space constraint is relaxed in this manner. No polynomial time algorithm can guarantee average query cost less than $M^{1-\epsilon} opt_B$ using space αB, for any constants $\alpha, \epsilon > 0$, unless $NP \subseteq n^{O(\log \log n)}$.

- We give an $O(n^5 k)$ algorithm for finding the union of k intervals on numeric data that maximize total interval support and meet a given minimum cumulative confidence constraint.

- We simulate the RS algorithm for finding the union of k intervals that maximize support and meet a given minimum independent confidence constraint, on both artificial patterned data and on real-world data. In each case quadratic run time is observed.

- Sampling dramatically decreases the running time. We show a sample s of size

$$s \geq \max(\frac{4}{\epsilon} \log \frac{2}{\delta}, \frac{16}{\epsilon} \log \frac{13}{\epsilon})$$

  is sufficient. We validate this analysis experimentally, and demonstrate that near-optimal solutions are found after only 500 examples.

- We show that the natural greedy algorithm for optimized association rules finds k intervals in linear time, and these have support at least $1/3$ that of the best k. We show that no stronger approximation guarantee can be made about the greedy approach.

- We define the novel problem of searching for maximal boasts in time-stamped performance data.

- For a natural partial order, we give an optimal algorithm that enumerates all maximal boasts in time linear in the number of such boasts.

- For a second natural partial order, we give an algorithm that enumerates all maximal boasts in time $O(n \log^3 n)$.

This body of work leaves some open questions of interest. First, the problem of selecting an optimal set of partial indices for a database begs an efficient, greedy solution. We are curious about the performance guarantees that can be made about such an approach, and we

have preliminary results suggesting a log (in the size of the database) factor relaxation in the storage constraint gives a near optimal solution to the problem. Second, there is room for improvement in the $O(n^5k)$ time dynamic programming algorithm for finding the optimal set of k intervals whose cumulative confidence exceeds a minimum threshold. Finally, we know there are generalizations of the boasting problem to which our techniques can be applied. These should be formalized and analyzed. In addition, there may be other partial orders worth exploring, and other connections to optimized association rules worth pursuing.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 26–28 1993.

[2] S. Agrawal, S. Chaudhuri, and V. Narasayya. Automated selection of materialized views and indices for sql databases. In *VLDB*, pages 496–505, 2000.

[3] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, October 1989.

[4] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. 43th Annu. IEEE Sympos. Found. Comput. Sci.*, 2002. to appear.

[5] O. Buchter and R. Wirth. Discovery of association rules over ordinal data: A new and faster algorithm and its application to basket analysis. In Xindong Wu, Kotagiri Ramamohanarao, and Kevin B. Korb, editors, *Research and Development in Knowledge Discovery and Data Mining, Second Pacific-Asia Conference, PAKDD-98, Melbourne, Australia, April 15-17, 1998, Proceedings*, volume 1394 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 1998.

[6] T. M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. pages 92–99, 1999.

[7] S. Chaudhuri and V. Narasayya. An efficient cost-driven index selection tool for microsoft sql server. In *VLDB*, 1997.

[8] S. Chaudhuri and V. Narasayya. Auto-admin 'what-if' index analysis utility. In *SIGMOD*, 1998.

[9] C-W Chung, J-K Min, and K. Shim. Apex: An adaptive path index for xml data. In *SIGMOD*, 2002.

[10] J. Elble, C. Heeren, and L. Pitt. Optimized disjunctive association rules via sampling. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, pages 43–50. IEEE Computer Society, 2003.

[11] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules: scheme, algorithms, and visualization. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 13–23. ACM Press, 1996.

[12] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Mining optimized association rules for numeric attributes. In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 182–191. ACM Press, 1996.

[13] S. Guillaume. Discovery of ordinal association rules. In Ming-Shan Cheng, Philip S. Yu, and Bing Liu, editors, *Advances in Knowledge Discovery and Data Mining, 6th Pacific-Asia Conference, PAKDD 2002, Taipei, Taiwan, May 6-8, 2002, Proceedings*, volume 2336 of *Lecture Notes in Computer Science*, pages 322–327. Springer, 2002.

[14] H. Gupta. Selection of views to materialize in a data warehouse. In *ICDT*, pages 98–112, 1997.

[15] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *ICDE*, pages 208–219, 1997.

[16] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *In Proc. ACM SIGMOD '96*, pages 205–216, 1996.

[17] D. Haussler. Decision-theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 1992.

[18] C. Heeren, H. V. Jagadish, and L. Pitt. Optimal indexing using near-minimal space. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 244–251. ACM Press, 2003.

[19] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining – a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, July 2000.

[20] H. Karloff and M. Mihail. On the complexity of the view selection problem. In *In Proc. of the 18th ACM SIGMOD-SIGACT-SIGART Symp. PODS*, pages 167–173, 1999.

[21] R. Kaushik, P. Bohannon, J. F. Naughton, and H. F. Korth. Covering indexes for branchin path queries. In *SIGMOD*, 2002.

[22] E. Keogh. Mining and indexing time series data. In *The 2001 IEEE International Conference on Data Mining. San Jose, California, November 29, 2001*, 2001.

[23] M. Last, A. Kandel, and H. Bunke. *Data Mining in Time Series Databases*. World Scientific Publishing Co., 2004.

[24] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *The VLDB Journal*, 2001.

[25] J. Lin and J. S. Vitter. e-approximations with minimum packing constraint violation. In *Proceedings of the 24th Ann. ACM Symp. on Thry. of Computing*, pages 771–782, 1992.

[26] G. Lohman, A. Skelley, G Valentin, D. Zillio, and M. Zuliani. Db2 advisor: An optimizer smart enough to recommend its own indices. In *ICDE*, pages 101–110, 2000.

[27] Andrian Marcus, Jonathan I. Maletic, and King-Ip Lin. Ordinal association rules for error identification in data sets. In *CIKM*, pages 589–591, 2001.

[28] T. Milo and D. Suciu. Index structures for path expressions. *Lecture Notes in Computer Science*, 1999.

[29] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[30] W. R. Pearson, G. Robins, D. E. Wrege, and T. Zhang. On the primer selection problem for polymerase chain reaction experiments. *Discrete and Applied Mathematics*, 1996.

[31] D. Pollard. *Convergence of Stochastic Processes*. Springer-Verlag, 1984.

[32] L. K. Poola and J. R. Haritsa. Sphinx: Schema-conscious xml indexing, 2002.

[33] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[34] R. Rastogi and K. Shim. Mining optimized support rules for numeric attributes. *Information Systems*, 26(6):425–444, 2001.

[35] R. Rastogi and K. Shim. Mining optimized association rules with categorical and numeric attributes. *Knowledge and Data Engineering*, 14(1):29–50, 2002.

[36] F. Rizzolo and A. Mendelzon. Indexing xml data with toxin. In *WebDB*, pages 49–54, 2001.

[37] A. Shukla, P. Deshpande, and J. Naughton. Materialized view selection for multi-cube data models. In *EDBT*, 2000.

[38] H. Toivonen. Sampling large databases for association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *In Proc. 1996 Int. Conf. Very Large Data Bases*, pages 134–145. Morgan Kaufman, 09 1996.

[39] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, March 1985.

[40] J. Wijsen and R. Meersman. On the complexity of mining quantitative association rules. *Data Mining and Knowledge Discovery*, 2(3):263–281, 1998.

[41] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of sampling for data mining of association rules. In *7th International Workshop on Research Issues in Data Engineering (RIDE'97)*, Birmingham, UK, April 1997.

[42] D. Zelenko. Optimizing disjunctive association rules. In *Proe. of PKDD '99, Lecture Notes in Computer Science (LNAI 1704)*, pages 204–213. Springer-Verlag, 1999.

Vita

Cinda Heeren received a B.S. in mathematics and computational sciences from Stanford University in 1990, a M.S. in Operations Research from Stanford in that same year, and a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign, in 2004.

Dr. Heeren has held positions as a Lecturer in the Departments of Computer Science, Statistics, and Mathematics at the California Polytechnic State University at San Luis Obispo (1990-1993), as a Teaching Assistant and Teaching Associate in the Department of Statistics at the University of California at Santa Barbara (1993), as a Research Assistant in the Department of Computer Science, University of Illinois at Urbana-Champaign (1997-2003), and as a Lecturer in the Department of Computer Science at University of Illinois at Urbana-Champaign (various times during the period 1998-2004).

Dr. Heeren has a strong interest in K-12 outreach and education. In addition to teaching middle school math at Countryside School (1997-2004), she has presented at numerous workshops including UIUC/ROE Novice Teacher Support Program, Expanding Your Horizons in Science and Mathematics conference for middle school girls, UIUC Engineering Open House, AIMS Drive-In Workshop for middle school teachers, MathManiaCS summer teacher training workshops, and the UIUC and College Board sponsored Java Engagement for Teacher Training (JETT) workshop. In addition, she is the advisor to the Women in Computer Science organization at the University of Illinois.

Her publications include:

1. Maximal Boasting. In preparation for submission, with L. Pitt.

2. Optimized Disjunctive Association Rules via Sampling. 2003 IEEE International Conference on Data Mining (ICDM03) Melbourne, FL, November, 2003, with J. Elble and L. Pitt.

3. Optimal Indexing Using Near-Minimal Space. 2003 ACM Symposium on Principles of Database Systems (PODS03) San Diego, CA, June, 2003, with H. V. Jagadish, and L. Pitt.

4. Constrained 2D Space-Time Meshing with All Tetrahedra. Proc. 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation, 2000, with A. Ungor, X.-Y. Li, A. Sheffer, R. Haber, and S.-H. Teng.