Finding Minimal Keys in a Relation Instance

C. Giannella and C.M. Wyss

May 14, 1999

Abstract

Mannila [11] cites as an open problem in Data Mining the problem of finding all minimal keys in a relation instance using only time that is polynomial in the number of relation attributes and number of minimal keys and sub-quadratic in the size of the relation instance. This paper investigates the efficacy of an "A Priori"-type approach to the problem, applied simultaneously in a level-wise top-down and bottom-up manner.

1 Introduction

1.1 Background and Motivation

The problem of finding all minimal keys given a relation schema, R, and a set of functional dependencies, F, has been widely studied in the literature. However, the problem of finding all minimal keys given a particular relation instance (without recourse to a pre-existing set of functional dependencies) has barely been touched upon. Yet Mannila cites the latter key-finding problem (given only a relation instance) as an important open problem in data mining (see [11]).

It is hard, at first glance, to think of any important data mining applications of finding minimal keys given a relation instance. In data mining, we are most often interested in functional dependencies or association rules in large bodies of data, not keys. What would be the advantage of having an efficient algorithm to find all minimal keys in a large body of existing data?

One possible use of the minimal keys is to scale down the storage requirements of the data, since the tuples could then be represented via a (presumably smaller) set of attribute values. In fact, one wonders if some data mining could profitably be carried out on key-set attributes (instead of having to include all attribute values in the search).

Another application might be in terms of index creation. If one could deduce a set of minimal keys that sufficiently overlapped the attribute space, then queries requiring certain attributes be repeatedly accessed or compared could profit. For example, some platforms allow one to specify which indices to create to optimize queries. If one knew of minimal keys, one could then create appropriate indices.[1]

Although it is not immediately clear if these uses would justify the search for an efficient minimal key-finding algorithm, in the rest of the paper we move away from this issue, in favor of an investigation of the problem in its own right.

1.2 Definitions and Problem Statement

In the rest of the paper we assume the following definitions. Let $R = \{A_1, \ldots, A_n\}$ be a relation schema and r an instance of R. A *key* of R with respect to r is $\emptyset \neq X \subseteq R$ such that: for all tuples $t_1, t_2 \in r$, $X[t_1] = X[t_2] \Rightarrow t_1 = t_2$. A *minimal key* of R with respect to r is $X \subseteq R$ such that: X is a key of R with respect to r and for all $Y \subsetneq X$, Y is not a key of R with respect to r. Often we will abbreviate the statement "X is a (minimal) key of R with respect to r" to "X is a (minimal) key" when R and r are clear from the context.

The problem indicated by Mannila ([11]) is the following (which we call *the key finding problem*):

> Given R (of size n) and r of size m, does there exist an algorithm for finding the set K of all minimal keys of r, using time polynomial in $max\{n, |K|\}$ and sub-quadratic in m.

Note that an algorithm that solves the problem may still be infeasible in practise if $|K|$ is exponential. A well-known result of Sperner [12] shows that the largest incomparable set, $J \subseteq \mathcal{P}(R)$ (for any $X, Y \in J, X \subseteq Y \Rightarrow X = Y$), is the set $\{X \in \mathcal{P}(R) || X| = \lfloor n/2 \rfloor\}$. Demetrovics [6] proved that there is an instance r of R for which the set of minimal keys is exactly J. Therefore, a sharp bound for the number of minimal keys is given by:

$$\binom{n}{\lfloor n/2 \rfloor}.$$

A simple argument can be used to generalize the result of Sperner mentioned previously. Namely to show that the largest incomparable set $J_i \subseteq \mathcal{P}(R)$ of size at most $1 \leq i \leq n$ (for any $X \in J_i, |X| \leq i$) is the set $\{X \in \mathcal{P}(R) || X| = \min\{i, \lfloor n/2 \rfloor\}\}$. Further, Beeri, et. al. [4] provide a generalization of the result of Demetrovics mentioned previously. They show that for any non-empty incomparable set, J, over a finite set R

[1] This suggestion regarding the use of minimal key sets was provided by Dennis Groth.

there is a relation instance, r, over R which has minimal keys precisely J. Therefore, a sharp bound for the number of minimal keys of size at most $1 \leq i \leq n$ is given by

$$\binom{n}{min\{i, \lfloor n/2 \rfloor\}}.$$

This result will prove useful in the complexity analysis of the bottom-up algorithm given later.

1.3 Related NP-Completeness Results

Since we are ultimately interested in determining whether the key finding problem can be solved, examining related NP-completeness results is a valuable endeavor. The following two questions are of interest.

1. Does there exist an NP-complete problem which can be solved *if* the key finding problem can be solved?
2. If not, does there exists a problem, not known to be NP-complete but also not know to be solvable in polynomial time, which can be solved *if* the key finding problem can be solved?

A problem similar to the key finding problem is known to be NP-complete. Given a relation schema R, $k \in \mathbb{N}$, and a relation instance (e.g. an Armstrong relation characterizing a set of functional dependencies on R), the problem of determining whether there is a minimal key of size at most k is NP-complete. We call this problem *the k-size key finding problem*. It was discovered to be NP-complete by Beeri, et. al. [4].

The crucial difference between the k-sized key finding problem and the key finding problem is that if $|K|$ (set of minimal keys) is exponential in n, a solution to the key finding problem is permitted to take exponential time in n. In this case, a solution to the key finding problem would not be an efficient solution to the k-sized key finding problem. Therefore, an algorithm for solving the key finding problem does not necessarily solve the k-sized key finding problem efficiently and so does not provide a positive answer to question 1.

In order to get an answer to question 1, the k-sized key finding problem must be restricted to the following problem (which we call *the sparse k-sized key finding problem*):

Given R, $k \in \mathbb{N}$, r such that $|K|$ is polynomial in $max\{|R|, |r|\}$, is there a minimal key of size at most k?

If this problem is NP-complete and the key finding problem can be solved, then a positive answer to question 1 is obtained (e.g. P=NP). To investigate whether the

sparse k-sized key finding problem is NP-complete, we look at the NP-completeness proof of the k-sized key finding problem given in [4].

Theorem 1 *The k-sized key finding problem is NP-complete.*

Proof: Given relation schema R, instance r, and $k \in \mathbb{N}$; let $mx = max\{|R|, |r|\}$. Checking whether a subset of R is a key and is of size less than or equal to k can be carried out in time $O(mxlog(mx))$. Hence the problem is in NP.

The remainder of the proof shows that the vertex cover problem is reducible to the k-sized key finding problem is NP-complete. Recall the vertex cover problem. Given an undirected graph $G = (V, E)$ and $k \in \mathbb{N}$, does there exist a minimal vertex cover of size at most k? By minimal we mean that no proper subset is also a vertex cover. The vertex cover problem is known to be NP-complete.

Given undirected graph $G = (V, E)$ where $V = \{v_1, \ldots v_p\}$ and $E = \{e_1, \ldots e_q\}$ a relation schema R_G and instance r_G is constructed. R_G has attributes V and r_G has $q + 1$ tuples. The first q tuples correspond to the edges in E. The last tuple consists of all zeros. Tuple t_i for $1 \leq i \leq q$ has the following form. Let $e_i = (v_a, v_b)$. Let x_ℓ denote the entry in t_i corresponding to attribute v_ℓ. If $\ell = a$ or $\ell = b$, then $x_\ell = e_i$; otherwise, $x_\ell = 0$.

Consider the following example. Let $G = (V = \{v_1, v_2, v_3, v_4\}, E = \{e_1 = (v_1, v_2), e_2 = (v_2, v_3), e_3 = (v_3, v_4)\})$. R_G and r_G are as seen in the table below. The column labels correspond to the attributes in R_G and the rows to the tuples in r_G.

$\underline{v_1}$	$\underline{v_2}$	$\underline{v_3}$	$\underline{v_4}$
e_1	e_1	0	0
0	e_2	e_2	0
0	0	e_3	e_3
0	0	0	0

The following property holds for all $\hat{V} \subseteq V$.

Lemma 1 *$\hat{V}$ is a minimal vertex cover of G if and only if $\hat{V}$ is a minimal key of R_G with respect to r_G.*

Proof: ($\Rightarrow$:) Assume $\hat{V}$ is a minimal vertex cover of G. Let t_i, t_j be tuples in r_G where $1 \leq i < j \leq q+1$. Suppose $\hat{V}[t_i] = \hat{V}[t_j]$. By construction, the only entries in t_i and t_j which are the same contain 0. Hence $\hat{V}[t_i]$ and $\hat{V}[t_j]$ consist of all zeros. Since $\hat{V}$ is a vertex cover of G, then all edges are touched by $\hat{V}$. Therefore, $\hat{V}[t_i]$ cannot consist of all zeros, which is a contradiction. It follows that $\hat{V}$ is a key of R_G with respect to r_G.

Suppose there exists $W \subsetneq \hat{V}$ where W is a key of R_G. Let e_i be an edge of G. Consider tuple t_i in r_G $(1 \leq i \leq q)$. If $W[t_i] = W[t_{q+1}]$ then $W[t_i]$ consists

of all zeros. However, t_i does not consist of all zeros since $1 \leq i \leq q$. Therefore, $(R - W)[t_i] \neq (R - W)[t_{q+1}]$, so, W cannot be a key. This is a contradiction. Assume that $W[t_i] \neq W[t_{q+1}]$. Since t_{q+1} consists of all zeros, then $W[t_i]$ does not. It follows that edge e_i is touched by a vertex in W. W is a vertex cover of G. However, $W \subsetneq \hat{V}$, so $\hat{V}$ is not a minimal vertex cover. This contradicts our original assumption. We conclude that $\hat{V}$ is a minimal key.

($\Leftarrow$:) Assume $\hat{V}$ is a minimal key of R_G with respect to r_G. Let e_i be an edge in G ($1 \leq i \leq q$). Consider tuple t_i of r_G. Since $\hat{V}$ is a key and t_{q+1} consists of all zeros, then similar reasoning to above shows that $\hat{V}[t_i]$ does not consist of all zeros. Therefore, edge e_i is touched by some vertex in $\hat{V}$. $\hat{V}$ is a vertex cover of G.

Suppose there exists $W \subsetneq \hat{V}$ such that W is a vertex cover of G. Let t_i, t_j be a tuples in r_G where $1 \leq j < i \leq q+1$. Since W is a vertex cover, then similar reasoning to above shows that $W[t_j]$ does not consist of all zeros. Since $i \neq j$, it follows that: $W[t_i] \neq W[t_j]$. We conclude that W is a key. However, $W \subsetneq \hat{V}$, so, $\hat{V}$ is not a minimal key. This contradicts the original assumption. We conclude that $\hat{V}$ is a minimal vertex cover of G.

QED Lemma

Now back to the proof of the Theorem. Let $k \in \mathbb{N}$. From the above lemma, it follows that: there exists a minimal vertex cover of G of size at most k if and only if there exists a minimal key of R_G with respect to r_G of size at most k. Since R_G and r_G can be constructed from G in polynomial time in $max\{|V|, |E|\}$, then the vertex cover problem reduces to the k-sized key finding problem.

QED Theorem

The important thing to note in this proof, is the lemma proven along the way. This lemma implies that the set of minimal keys K of R_G is exactly the set of minimal vertex covers of G. Hence a similar restriction as above on the vertex cover problem can be used to answer question 1. Specifically, if the key finding problem can be solved and the following problem (which we call *the sparse vertex cover problem*) is NP-complete, then question 1 can be answered positively (e.g. P=NP).

> Given an undirected graph $G = (V, E)$ and $k \in \mathbb{N}$, such that the number of minimal vertex covers of G is polynomial in $max\{|V|, |E|\}$, is there a minimal vertex cover of size at most k?

To the best of the authors' knowledge, it is not known whether the sparse vertex cover problem is NP-complete. So, question 1 remains open. On to question 2. To the best of the authors' knowledge, it is also not known whether the sparse vertex cover

problem is solvable in polynomial time. So, question 2 is answered positively.

A final note concerning related NP-completeness results. The key finding problem places no restrictions on the domain of values over which tuples consist. We see above that question 2 can be answered positively in this setting and question 1 is open. However, if the domain is restricted to a finite set of values for all relation schemas and instances, then the situation changes. Question 2 becomes open as well as question 1. In particular consider the case where the domain of values is $\{0, 1\}$. A relation schema R and an instance r over R in which all tuples of r are made up of entries from $\{0, 1\}$ is called a *Bernoulli database*. Consider the following problem (which we call *Bernoulli key finding problem*) which is a special case of the key finding problem.

> Given Bernoulli database R (of size n), r of size m, does there exist an algorithm for finding the set K of all minimal keys of r, using time polynomial in $max\{n, |K|\}$ and sub-quadratic in m?

Questions 1 and 2 remain open for the Bernoulli key finding problem despite the fact that there do exist related NP-complete problems. For example, the following problem is NP-complete:

> Given Bernoulli database R, r, and $k \in \mathbb{N}$, does there exist a key of size at most k?

The proof involves reducing 3-SAT to this problem and is due to Ed Robertson [8]. A connection to a sparse 3-SAT problem (which would provide a positive answer to question 2) cannot, however, be made. The reason is that the Bernoulli database produced by the reduction *always* has an exponential number of minimal keys.

The fact that question 2 can be answered for the general key finding problem but not the Bernoulli key finding problem suggests that this special case is considerably easier to solve.

1.4 Previous Work

The first four algorithms considered in this paper are based on a level-wise "A-priori" approach similar to that used in mining association rules ([1], [2], and [9]). Mannila ([10]) relates the problem of minimal key-finding given a relation instance to the problem of transversing hypergraphs.

Demetrovics et. al ([7]) present some interesting work concerning the average length of keys in (random) databases. In particular, given a relation schema R such that $|R| = n$, consider a Bernoulli database of size m and a random distribution $p : \{0, 1\}^n \rightarrow \{0, 1\}$ giving the probability that a particular tuple appears in the database. If it is the case that $\exists N_1, N_2$ such that

$$\frac{N_1}{2^n} \leq p(t) \leq \frac{N_2}{2^n}$$

for any particular tuple, t (which surely is the case for any "reasonable" random database generation) then the minimal keys cluster around $2log_2(m)$ in size. Our empirical study (using random Bernoulli databases) bears this research out (§3). Albrecht et. al ([3], [7]) use this fact in a *Monte Carlo* method for finding minimal keys, which essentially looks for keys in the attribute subsets of size $2log_2(m)$ first.

1.5 Overview of this Paper

In the next section, we present four algorithms based on the level-wise A-priori approach for association rule mining. The first uses a "top-down" strategy to search through the power-set lattice $\mathcal{P}(R)$, the second uses a "bottom-up" strategy.

For both algorithms, the worst case performance is exponential in the number of minimal keys (although sub-quadratic in $|r|$) . So they do not have the required time-complexity of a solution to Mannila's problem.

The third and forth algorithms combines both techniques in a hybrid search strategy. The third combines the techniques in perhaps the most simple-minded way possible. The top-down and bottom-up parts run independently of each other and "meet" in the middle of the power-set lattice. At this point their key sets are merged into the output minimal key set. The forth algorithm combines the techniques in a more complicated way. The top-down and bottom-up parts "communicate" with each other while traversing the lattice. Each using the intermediate results of the other to prune the search space. All of the four algorithms are proven to correctly find all minimal keys, however none solve Mannila's problem in the worst case.

In §3, we report the results of an empirical study of the a-priori-type algorithms. This study generates (random) Bernoulli databases and answers various questions regarding the layout of the keys in these databases and the performance of our algorithms. In particular, the minimal keys seem to be of the sizes predicted in [7], namely clustered around $2log_2(m)$ in size in randomized Bernoulli databases. Also, a levelwise approach is seen to be infeasible from a space complexity point of view. Alternate depth-first versions of the a-priori algorithms are given.

Finally, we report our conclusions regarding the a-priori approach to Mannila's problem, and indicate some directions future research on the problem might take.

2 A Naieve "A-Priori" Approach

The problem involves searching through the power-set lattice of R, $\mathcal{P}(R)$. From the definition of minimal keys, there are two requirements of any subset $X \in \mathcal{P}(R)$ to be in the result set, K:

1. X must be a key.

2. X must be minimal (no proper subset of X is also a key).

From these two requirements we can extract the following two principles, which will lead to dual pruning mechanisms on $\mathcal{P}(R)$.

Principle 1. If $X \in \mathcal{P}(R)$ is *not* a key, then no $Y \subseteq X$ is a key.

Principle 2. If $X \in \mathcal{P}(R)$ is a minimal key, then no $Y \supseteq X$ is a minimal key and no $Y \subsetneq X$ is a key.

The idea behind our "A-Priori" approach (both top-down and bottom-up) is that, at each level, we will use our partial result set and a suitable principle to generate a frontier set which prunes the potential set of subsets of R that need be examined at the next level. In essence, the frontier set, F, guarantees that for all minimal keys, X, at some not-yet-reached level is a subset (top-down) or superset (bottom-up) of some element in F. As a result, only the subsets or supersets of F need be examined at the next level. This method (in conjunction with the dual pruning principles) is similar in spirit to the level-wise Apriori framework for mining association rules (see [1], [2] and [9]).

2.1 Searching Top-down

In this section we consider an algorithm that searches the power-set lattice $\mathcal{P}(R)$ top-down, using principle 1 to prune the search. During iteration $n-1 \geq i \geq 1$ of the algorithm, a frontier set, K_i, of subsets of R is generated from the frontier set, K_{i+1}, of the previous iteration, $i+1$. At the end of iteration i, K_i will contain *exactly* the keys of size i. Principal 1 allows K_i to be generated by only looking at the subsets of elements in K_{i+1}. By principal 1, any $X \subseteq R$ of size $i+1$ which is not a key will not have any keys as subsets. Because K_{i+1} contains exactly the set of keys of size $i+1$ before the end of iteration i, then $X \in K_{i+1}$. Therefore, any minimal key of size i will be a subset of some element of K_{i+1}. Looking at all the subsets of elements in K_{i+1} is all that is required at iteration i to generate exactly the set of minimal keys of size i.

Since we are interested in finding all *minimal* keys, we must remove all elements of K_{i+1} which have a subset in K_i before going on to iteration $i-1$. This is carried out by marking during iteration i all elements in K_{i+1} which contribute a subset to K_i. Hence, during iteration i, K_i is generated and all elements of K_{i+1} which have a subset in K_i are marked. After generating K_i, before proceeding on to the next iteration, all marked sets in K_{i+1} are removed. The output of the algorithm will be the union of all K_i sets.

Because only K_{i+1} need be looked at during iteration i, K_{i+1} was referred to as the frontier. The algorithm is presented in figure 1. Most of the notation is clear except, perhaps $\bigsqcup$, which refers to a non-duplicate removing union.

Input: R, a relation schema of size n and r an instance of R of size m.
Output: $K = \bigsqcup_{i=1}^{n} K_i$, the set of minimal keys of R with respect to r.

```
1    K_1, ..., K_{n-1} := ∅ and K_n := {R}
2    for i := n − 1 down to 1 do
3       for Y_td ∈ K_{i+1} do
4          for X_td ⊆ Y_td s.t. |X_td| = i do
5             if X_td is a key w.r.t. r then
6                K_i := K_i ⊔ {X_td}
7                mark Y_td
8          endfor
9       endfor
10      remove all marked Y_td ∈ K_{i+1}
11      remove duplicates from K_i
12   endfor
```

Figure 1: Top-Down Algorithm using A-Priori Pruning Approach

During the course of the for loop at line 4, many duplicate $X's$ may be generated. This is because different $Y's$ may share subsets of size i. The problem of duplicate removal is put off until step 11 so that all duplicates can be removed at the same time. As a result the $\bigsqcup$ operation is used in line 6.

Clearly, at the end of iteration i, K_i contains exactly the keys of size i and there does not exist an element of K_{i+1} which has a subset in K_i. These results generalize into the following invariant which directly yields a correctness proof.

Lemma 2 *For all $n \geq i \geq 1$ the following statements hold at the end of iteration i of the top-down algorithm:*

1. *For all $n \geq j \geq i$ and each $Y \in K_j$, there does not exist Z a key of size $\geq i$ and $Z \subsetneq Y$*
2. *For all keys Y of size $n \geq j \geq i$, if there does not exist a key Z of size $\geq i$ such that $Z \subsetneq Y$, then $Y \in K_j$.*

Proof Sketch: A simple induction on i will yield the desired result. The details are not very illuminating and are omitted.

QED

The desired correctness proof falls out directly.

Theorem 2 *At the termination of the top-down algorithm, $K = \bigsqcup_{i=1}^{n} K_i$ equals the set of minimal keys.*

Proof: Let $X \in K$ at the termination of the algorithm, then $X \in K_{|X|}$ at the end of iteration $|X|$. By 1. with $i = 1$ and $j = |X|$, there does not exist Z a key with $Z \subsetneq X$. X is a minimal key.

Let $X \subseteq R$ be a minimal key. The antecedent of 2. holds with $i = 1$ and $j = |X|$. Hence, $X \in K_{|X|}$ at the termination of the algorithm.

QED

Now we analyze the top-down algorithm to get a worst case complexity. We assume here and throughout the paper that the following are basic operations. This list is not exhaustive. Implicitly assumed to be basic are all of the obvious operations (e.g. arithmetic, etc.).

- arbitrary length tuple comparisons,
- arbitrary length attribute set comparison ($X \subseteq Y$ for $X, Y \subseteq R$),
- attribute set length computation ($|X|$),
- $\bigsqcup$ operation,
- marking and checking for marking on arbitrary length attribute sets.

First some preliminary complexity computations. The removal of marked $Y \in K_{i+1}$ in line 10 can be done in worst case time $O(|K_{i+1}|log(|K_{i+1}|))$ by recursively splitting then merging similar to the split-merge technique used for merge sort. The removal of duplicates from K_i in line 11 can be done in time $O(|K_i|log(|K_i|))$. First sort K_i (time

$O(|K_i|log(|K_i|)))$. Then scan K_i marking all duplicates (time $O(|K_i|)$). Then using the same technique as above, remove all marked sets (time $O(|K_i|log(|K_i|)))$.

Recall that in the main problem statement, we are interested in the complexity of our algorithms in terms of n, m, and $|K|$. At each iteration $1 \leq i \leq n-1$, the algorithm requires time

$$O(|K_{i+1}|(i+1)mlog(m) + |K_{i+1}|log(|K_{i+1}|) + |K_i|log(|K_i|)).$$

$(i+1)$ is an upper-bound on the number of subsets of size i for each element in K_{i+1}. $mlog(m)$ is the time required to identify whether a set X is a key by sorting the instance r. The $|K_{i+1}|log(|K_{i+1}|) + |K_i|log(|K_i|)$ terms come from lines 10 and 11 as explained earlier. Due to removal of marked sets, K_i is not monotonic in i. Let $k_{mx} = max_{j=1}^{n}(|K_j|)$. The worst case running time of the algorithm is

$$O(\sum_{i=1}^{n-1} k_{mx}(i+1)mlog(m) + k_{mx}log(k_{mx})).$$

Simplifying a bit we get a worst case running time of:

$$O(n^2 k_{mx}mlog(m) + (n)k_{mx}log(k_{mx})).$$

In the event that k_{mx} is polynomial in $|K|$, our algorithm runs in time polynomial in n and $|K|$ and sub-quadratic in m. This is exactly what we need to solve our problem. However, k_{mx} might be exponential in $|K|$. Consider a relational instance r_{td} in which $K = \{\{A_1\}, \{A_2\}, \ldots, \{A_n\}\}$ ($|r_{td}|$ contains a single tuple of all zeros). In this case k_{mx} is exponential in $|K|$ since $|K_{\lfloor n/2 \rfloor}| = \binom{n}{\lfloor n/2 \rfloor} \geq n2^{n-1}$ (for large n). The algorithm certainly does not solve the problem in this case. To get an idea of the complexity of the algorithm independent of k_{mx}, we compute the worst-case complexity in terms of n and m only.

Clearly, $|K_j| \leq \binom{n}{j}$ for all j. Hence the worst case running time of the algorithm is $O(\sum_{i=1}^{n-1}[\binom{n}{i+1}(i+1)mlog(m) + \binom{n}{i+1}log(\binom{n}{i+1}) + \binom{n}{i}log(\binom{n}{i})]) \subseteq O(n2^{n-1}mlog(m) + nlog(n)2^{n-1} + nlog(n)2^{n-1})$. Simplifying a bit we get a worst case running time of:

$$O(n2^{n-1}[mlog(m) + 2log(n)]).$$

Recall the instance r_{td} above. At each iteration, i, $|K_{i+1}| = \binom{n}{i+1}$ and $|K_i| = \binom{n}{i}$. Hence the worst case running time in terms of n and m is attained. Since $|K| = n$, then the algorithm runs in time exponential in the output size on r_{td}. Therefore, r_{td} is a prototypical example of an instance which causes the worst behavior in the top-down algorithm.

2.2 Searching Bottom-Up

In this section we consider an algorithm that searches the power-set lattice $\mathcal{P}(R)$ bottom-up, using principle 2 to prune the search. The way in which the frontier is maintained is different than the top-down algorithm. During iteration $1 \leq i \leq n$ of the bottom-up algorithm, a frontier set, NK_i, of subsets of R is generated from the frontier set, NK_{i-1}, of the previous iteration, $i-1$. At the end of iteration i, NK_i will contain *exactly* the non-keys of size i and the empty set. Principal 1 allows NK_i to be computed during iteration i by only looking at the supersets of elements of NK_{i-1}. To see this, let X be a non-key of size i. By principal 1, every subset of X of size $i-1$ is not a key and so is in NK_{i-1} at the end of iteration $i-1$. Therefore, X is a superset of some element in NK_{i-1} at the end of iteration $i-1$.

Principal 2 ensures that NK_{i-1} is truly a frontier set during iteration i. Specifically, any minimal key of size i must be a superset of some element of NK_{i-1}. To see this, let X be a minimal key of size i. By principal 2, no $Y \subsetneq X$ of size $i-1$ is a key and hence every subset of X of size $i-1$ is in NK_{i-1} at the end of iteration $i-1$. We are guaranteed, then, to find all minimal keys of size i during iteration i.

The only complication arises in ensuring that *exactly* the minimal keys of size i are found during iteration i. Let X be a set of size i which is a superset of an element in NK_{i-1}. X will be looked at during iteration i. Checking whether X is a key is easy, but does not ensure that X is a minimal key! There might exist a key $Y \subsetneq X$ of size $i-1$. The problem occurs because X might have some subsets of size $i-1$ which are non-keys and some which are keys. Therefore, all subsets of size $i-1$ of X must be checked. Given that X is a key, if all of these sets are non-keys, then X is a minimal keys, otherwise not.

We have arrived a procedure for finding all of the minimal keys of size i, given NK_{i-1}. For each $Y_{bu} \in NK_{i-1}$ consider each superset, X_{bu} of size i. If X_{bu} is not a key, then add X_{bu} to NK_i. Otherwise, if all subsets of X_{bu} of size $i-1$ are non-keys, then add X_{td} to the cumulative minimal key set K. The algorithm is presented in figure 2.

As in the top-down algorithm $\bigsqcup$ is used in line 6 and duplicate removal is dealt with all at once in line 11 (time $O(|NK_i|log(|NK_i|))$). Also, $\bigsqcup$ is used in line 8. There may well be duplicates in K. Duplicate removal from K is handled all at once in line 12.

Now we prove the correctness of the bottom-up algorithm. The following two useful facts, can be directly observed from the algorithm (proofs are omitted). At the end of iteration $1 \leq i \leq n$ of the bottom-up algorithm:

- **Fact 1:** if $X \in K$, then X is a key
- **Fact 2:** if $X \in NK_i$, then X is not a key.

The non-key invariance property in NK_i (stated earlier) can now be established.

Input: R, a relation schema of size n and r an instance of R of size m. Output: K, the set of minimal keys of R with respect to r.

```
1     K := ∅ and NK_0 = {∅}, NK_1, ... , NK_n := ∅
2     for i := 1 to n do
3        for Y_bu ∈ NK_{i-1} do
4           for X_bu ⊇ Y_bu s.t. |X_bu| = i do
5              if X_bu is not a key w.r.t. r then
6                 NK_i := NK_i ⋃ {X_bu}
7              else if ∀Z ⊆ X_bu with |Z| = i − 1, it follows
                          that: Z is not a key w.r.t. r then
8                 K := K ⋃ {X_bu}
9           endfor
10       endfor
11       remove duplicates from NK_i
12       remove duplicates from K
13    endfor
```

Figure 2: Bottom-up Algorithm using A-Priori Pruning Approach

Lemma 3 *For all $1 \leq i \leq n$, at the end of iteration i of the bottom-up algorithm, $NK_i = \{X \subseteq R || X| = i$ and X is not a key $\}$.*

Proof: A simple induction will show that for all $1 \leq i \leq n$, $\{X \subseteq R||X| = i$ and X is not a key $\} \subseteq NK_i$. From fact 2, the other direction of the inclusion follows for all i.

QED

Finally, we can show that the algorithm is correct.

Theorem 3 *At the termination of the bottom-up algorithm K is exactly the set of minimal keys.*

Proof: Let $X \in K$ at the termination of the algorithm. From fact 1, X is a key. We must show that X is minimal. Suppose not, then there exists a key $\hat{X} \subsetneq X$ of size $|X| - 1$. Since $X \in K$, then X was added to K during iteration $|X|$. Therefore step 7 must pass at some point during the iteration with $X_{bu} = X$. Since $\hat{X} \subseteq X$ and $|\hat{X}| = |X| - 1$, then it follows that: $\hat{X}$ is not a key. This is a contradiction.

Let X be a minimal key. We shall show that X is in K at the end of iteration $|X|$. If $|X| = 1$, then since $\in NK_0$, X is a key, and all subsets of X are non-keys, then X will be added to K during the 1^{st} iteration. Assume $|X| > 1$.

Since X is a minimal key, then the following statement holds:

$$\forall Z \subseteq X \text{ with } |Z| = |X| - 1, Z \text{ is not a key .} \qquad (\ddagger)$$

From the invariance of NK lemma above, there exists $\hat{Z} \subseteq X$ of size $|X| - 1$ in $NK_{|X|-1}$ at the end of the $(|X| - 1)^{st}$ iteration. Therefore, at some point during the $|X|^{th}$ iteration step 5 is reached with $Y_{bu} = \hat{Z}$ and $X_{bu} = X$. Since X is a key, then step 5 will fail and step 7 will be reached. By statement $(\ddagger)$ above, step 7 will pass. Hence, X will be added to K.

It follows that the set of minimal keys is exactly K at the termination of the algorithm.

QED

Now we analyze the bottom-up algorithm to find a worst-case running time. As in the complexity analysis of the top-down algorithm, we are interested in the worst case running time in terms of n, m, and $|K|$.

Let $\hat{K}_i$ denote K at the end of iteration i. At each iteration $1 \leq i \leq n$, the algorithm requires $O(|NK_{i-1}|i[mlog(m) + imlog(m)] + |NK_i|log(|NK_i|) + |\hat{K}_i|log(|\hat{K}_i|))$.

The $|NK_{i-1}|i[mlog(m)+imlog(m)]$ expression comes from lines 3-10. The $|NK_i|log(|NK_i|)$ and $\hat{K}_i|log(|\hat{K}_i|)$ expressions come from lines 11,12. Therefore, a worst-case running time of the following is obtained.

$$O(mlog(m)[\sum_{i=1}^{n} |NK_{i-1}|i^2] + [\sum_{i=1}^{n} |NK_i|log(|NK_i|)] + [\sum_{i=1}^{n} |\hat{K}_i|log(|\hat{K}_i|)]) \qquad (1)$$

Notice that $\hat{K}_i$ is monotonic increasing in i so for all j, $|\hat{K}_j| \leq |K|$. However, NK_i is not necessarily monotonic increasing in i. Let $nk_{mx} = max_{i=1}^{n}(|NK_i|)$. The worst case running time is:

$$O(\sum_{i=1}^{n}(nk_{mx}[imlog(m) + i^2mlog(m)] + nk_{mx}log(nk_{mx}) + |K|log(|K|))).$$

Simplifying a bit we get a worst case running time of:

$$O(n^3nk_{mx}mlog(m) + nk_{mx}log(nk_{mx}) + |K|log(|K|)).$$

Similar to the top-down algorithm, if nk_{mx} is polynomial in $|K|$, then the bottom-up algorithm solves our problem. However, nk_{mx} may be exponential in $|K|$. Consider

a relation instance r_{bu} with $K = \{\{A_1, \ldots A_n\}\}$ ($|r_{bu}|$ is linear in n and in Bernoulli). Clearly, $|NK_{\lfloor n/2 \rfloor}| = \binom{n}{\lfloor n/2 \rfloor}$, so, nk_{mx} is exponential in $|K|$. To get an idea of the complexity of the algorithm independent of nk_{mx}, we compute the worst-case complexity in terms of n and m only.

To do so we examine 1 more closely. Upper-bounds on $|NK_{i-1}|, |NK_i|$ and $|\hat{K}_i|$ are needed. Firstly, $|NK_{i-1}| \leq \binom{n}{i-1}$. Secondly, as mentioned earlier, a sharp bound on the number of minimal keys of size at most i is $\binom{n}{min\{i, \lfloor n/2 \rfloor\}}$. Hence $|\hat{K}_i| \leq \binom{n}{min\{i, \lfloor n/2 \rfloor\}}$. We conclude that the algorithm requires in the worst case time:

$$O(mlog(m) \sum_{i=1}^{n} \binom{n}{i-1} i^2 + \sum_{i=1}^{n} \binom{n}{i} log(\binom{n}{i}) + \tag{2}$$

$$\sum_{i=1}^{n} \binom{n}{min\{i, \lfloor n/2 \rfloor\}} log(\binom{n}{min\{i, \lfloor n/2 \rfloor\}})). \tag{3}$$

To simplify consider first the subexpression:

$$mlog(m) \sum_{i=1}^{n} \binom{n}{i-1} i^2. \tag{4}$$

By [5] 4 is bounded above by $mlog(m)n^2 2^{n-2}$. Now consider the second subexpression:

$$\sum_{i=1}^{n} \binom{n}{i} log(\binom{n}{i}). \tag{5}$$

By Stirling's approximation it follows that 5 is of order $O(\sum_{i=1}^{n} \binom{n}{i-1})(nlog(n) - (n-i)log(n-i) - ilog(i)))$. Hence 5 has upper-bound $O(nlog(n)2^n)$. Now consider the third subexpression:

$$\sum_{i=1}^{n} \binom{n}{min\{i, \lfloor n/2 \rfloor\}} log(\binom{n}{min\{i, \lfloor n/2 \rfloor\}}) \tag{6}$$

By splitting the sum and using Stirling's approximation as above we obtain an upper-bound on 6 of:

$$O(\sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} (nlog(n) - (n-i)log(n-i) - ilog(i) +$$

$$\sum_{\lfloor n/2 \rfloor + 1}^{n} \binom{n}{\lfloor n/2 \rfloor} (nlog(n) - (n - \lfloor n/2 \rfloor)log((n - \lfloor n/2 \rfloor)) - \lfloor n/2 \rfloor log(\lfloor n/2 \rfloor))).$$

Dropping the $(n-i)log(n-i) - ilog(i)$ expression and simplifying the second summation, we get upper-bound:

$$O(nlog(n)\sum_{i=0}^{\lfloor n/2 \rfloor}\binom{n}{i} + n^2log(n)\binom{n}{\lfloor n/2 \rfloor}).$$

Stirling's approximation yields an upper-bound of:

$$O(n^2log(n)2^{n-1}).$$

Plugging in the upper-bounds for the three subexpressions of and simplifying the following upper-bound in terms of m and n for the worst-case running time of the algorithm is obtained:

$$O(mlog(m)n^2 2^{n-2} + n^2log(n)2^{n-1}) \tag{7}$$

Can this worst case running time be achieved? Recall instance r_{bu} with minimal key set $\{\{A_1, \ldots, A_n\}\}$. Clearly, for all $1 \leq i \leq n-1$, $|\hat{K_i}| = 0$ and for all $1 \leq i \leq n$, $|NK_{i-1}| = \binom{n}{i-1}$. A straight-forward analysis will show that the algorithm comes within a factor of $nlog(n)$ of achieving the worst-case time in 7. We conclude that r_{bu} is a prototypical example which causes the worst behavior in the bottom-up algorithm.

2.3 Searching Top-down and Bottom-up

We have seen that the worst type of instances for the top-down algorithm are those which have $K = \{\{A_1\}, \ldots, \{A_n\}\}$ (r_{td} denoted an instance which has this minimal key set). We have also seen that the worst type of instances for the bottom-up algorithm are those which have $K = \{\{A_1, \ldots, A_n\}\}$ (r_{bu} denoted an instance which has this minimal key set). On these instances, the top-down and bottom algorithms, respectively, fail to solve our original problem. However, the bottom-up algorithm handles instance r_{td} easily and vice versa. We are motivated then to consider a "hybrid" algorithm which searches the power-set lattice $\mathcal{P}(R)$ top-down, *and* bottom-up using *both* principle 1 and 2 above. Our first and simplest approach is to have the top-down and bottom-up algorithms search the top and bottom half of the power-set lattice, respectively, and merge their results. The only interaction between the two algorithms occurs at the very end when the top-down key set and bottom-up minimal key set are merged. This algorithm is presented in figure 3. Since the top-down and bottom-up portions do not interact, we frequently refer to this algorithm as *the hybrid without communication algorithm.*

The correctness proof of the hybrid without communication algorithm follows almost directly from the correctness proofs of the top-down and bottom-up algorithms. The only complication involves the merging in lines 24-29. However, this is only a minor problem. The proof is, thus, omitted.

Input: R, a relation schema of size n and r an instance of R of size m.
Output: K, the set of minimal keys of R with respect to r.

1 $K, K_1, \ldots, K_{n-1} := \emptyset; NK_0 = \{\emptyset\}, NK_1, \ldots, NK_n := \emptyset$ and $K_n := \{R\}$
COMMENT Lines 2-11 are the top-down portion
2 **for** $i := n - 1$ **downto** $\lceil n/2 \rceil$ **do**
3 **for** $Y_{td} \in K_{i+1}$ **do**
4 **for** $X_{td} \subseteq Y_{td}$ s.t. $|X_{td}| = i$ **do**
5 **if** X_{td} is a key of R w.r.t. r, **then**
6 $K_i := K_i \bigsqcup \{X_{td}\}$
7 mark Y_{td}
8 **endfor**
9 **endfor**
10 remove all marked Y_{td} from K_{i+1}
11 remove all duplicates from K_i
COMMENT Lines 12 - 22 are the bottom-up portion
12 **for** $Y_{bu} \in NK_{n-i-1}$ **do**
13 **for** $X_{bu} \supseteq Y_{bu}$ s.t. $|X_{bu}| = n - i$ **do**
14 **if** X_{bu} is *not* a key w.r.t. r **then**
15 $NK_{n-i} := NK_{n-i} \bigsqcup \{X_{bu}\}$
16 **else if** $\forall Z \subseteq X_{bu}$ with $|Z| = n - i - 1$), we
17 have: Z is *not* a key w.r.t. r **then**
18 $K := K \bigsqcup \{X_{bu}\}$
19 **endfor**
20 **endfor**
21 remove duplicates from NK_{n-i}
22 remove duplicates from K
23 **endfor**
COMMENT 24-29 – merging of top-down and bottom-up
24 **for** $i := n - 1$ **downto** $\lceil n/2 \rceil$ **do**
25 **for** $X_{td} \in K_i$ **do**
26 **if** $\forall Z \in K$, with $|Z| \leq \lceil n/2 \rceil, Z \not\subseteq X_{td}$ **then**
27 $K := K \bigsqcup \{X_{td}\}$
28 **endfor**
29 **endfor**

Figure 3: Simple "Hybrid" Algorithm using top-down and bottom-up pruning

One of the primary motivations for the hybrid without communication algorithm is to combine the best features of the top-down and bottom-up algorithms. We investigate now how close the algorithm has come to this goal. Specifically, we develop a lower-bound on the running time. From this result, we describe a class of inputs on which the algorithm fails exponentially to solve Mannila's problem [2].

Theorem 4 *Given R a relation schema (of size n) and r an instance of R (of size m), the hybrid algorithm without communication (e.g. algorithm 3) requires time at least $\Omega(2^{n/2})$.*

Proof: Let α be a minimal key. There are two cases to consider.

1. **Case:** $|\alpha| \leq \lceil n/2 \rceil$. It follows that for every $Z_i \supsetneq \alpha$ of size $n-1 \geq i \geq \lceil n/2 \rceil + 1$, $Z_i \in K_i$ at the start of iteration $i-1$. Hence the top-down portion of the algorithm must examine at least the following number of sets:

$$\sum_{i=\lceil n/2 \rceil + 1}^{n-1} \binom{n-|\alpha|}{i-|\alpha|}.$$

   Since $|\alpha| \leq \lceil n/2 \rceil$, then a lower-bound of $2^{n/2-1} - 2$ can be obtained on the sum. Therefore the algorithm requires at least $\Omega(2^{n/2})$ time.

2. **Case:** $|\alpha| \geq \lceil n/2 \rceil + 1$. It follows that for every $Z_{n-i} \subsetneq \alpha$ of size $1 \leq n-i \leq \lceil n/2 \rceil$, $Z_{n-i} \in NK_{n-i}$ at the start of iteration $i-1$ By an analogous argument to the previous case, we conclude that the bottom-up portion of the algorithm must examine at least $2^{n/2-1}-1$ sets. Therefore, the algorithm requires at least $\Omega(2^{n/2})$ time.

QED

This theorem implies that on the class of inputs with a polynomial number of minimal keys, the hybrid without communication algorithm fails exponentially to solve Mannila's problem. In particular the algorithm fails exponentially on instances r_{td} and r_{bu}. We see clearly that the algorithm does not come close to achieving the goal of combining the best properties of the top-down and bottom-up algorithm.

2.4 Adding Communication Between the Top-down, Bottom-up Search

Let's examine a bit more closely the behavior of the hybrid algorithm on instance r_{bu} (over relation schema $R = \{A_1, \ldots, A_n\}$); recall that r_{bu} has minimal key set

[2] The algorithm is deemed to fail exponentially on an input if the time required on the input is exponentially greater than allowed.

$\{A_1, \ldots, A_n\}$. After iteration $n-1$, $K_{n-1} = \emptyset$ and $NK_1 = \{\{A_i\} | 1 \leq i \leq n\}$. Because, $K_{n-1} = \emptyset$, it is clear that any set of size smaller than $n-1$ is not a minimal key. Therefore, the algorithm need not proceed any further. However, the bottom-up portion will examine all sets of size $\leq \lceil n/2 \rceil$ in vain. This is a canonical example illustrating the need for communication between the top-down and bottom-up portions. In this example, $K_{n-1} = \emptyset$, should signal the bottom-up portion to halt after the first iteration.

More generally, we would like to utilize information contained in K_i during the bottom-up part of iteration i and information contained in K and NK_{n-i-1} during the top-down portion of iteration i. Elements of K_i are minimal keys down to size i. Elements of NK_{n-i-1} are non-keys and elements of K are minimal keys (of size $\leq n-(i+1)$).

The next lemma shows how K_i can be used to guide the bottom-up search.

Lemma 4 *At the end of step 11 during iteration $n-1 \geq i \geq \lceil n/2 \rceil$ of the hybrid algorithm without communication, $\forall Y_{bu} \in NK_{n-i-1}$ if $\nexists X_{td} \in K_i$ s.t. $Y_{bu} \subseteq X_{td}$, then $\forall X_{bu} \supseteq Y_{bu}$ of size $n-i$, X_{bu} is not a key.*

Proof: Straight-forward and omitted.

QED

At the end of step 27 of iteration i of the hybrid without communication algorithm, let $\hat{K}_i$ denote K (e.g. the set of minimal keys of size $\leq n-i$). $\hat{K}_{i+1}$ can be utilized to prune the top-down search by allowing certain elements of K_{i+1} to be ignored during iteration i.

At first glance, we would like to be able to prune all $Y_{td} \in K_{i+1}$ s.t. there exists $Y_{bu} \in \hat{K}_{i+1}$ s.t. $Y_{bu} \subseteq Y_{td}$ from consideration during the top-down part of the algorithm at iteration i. Clearly such a Y_{td} cannot be a minimal key since Y_{bu} is a key. However, Y_{td} may have a subset, X, of size $\geq \lceil n/2 \rceil$ for which there does not exist $Y_{bu} \in \hat{K}_{i+1}$ s.t. $Y_{bu} \subseteq X$. Our first glance pruning approach is not sound (it is too ambitious).

At second glance, it is clear that $Y_{td} \in K_{i+1}$ can be pruned from consideration during iteration i, if for *every* $X \subseteq Y_{td}$ of size $n-i$ there exists $Y_{bu} \in \hat{K}_{i+1}$ s.t. $Y_{bu} \subseteq X$. The rationale is that every $X \subseteq Y_{td}$ of size $n-i$ is not a minimal key and $n-i \leq n - \lceil n/2 \rceil \leq \lceil n/2 \rceil$. Since the bottom-up portion will get every minimal key of size $\leq \lceil n/2 \rceil$, then Y_{td} will not have any minimal key subsets which will be missed. The next lemma states precisely how $\hat{K}_{i+1}$ can be used to guide the top-down search.

Lemma 5 *At the end of step 23 of iteration $n-1 \geq i+1 > \lceil n/2 \rceil + 1$ of the hybrid algorithm without communication, $\forall Y_{td} \in K_{i+1}$ if $\forall X \subseteq Y_{td}$ of size $n-i, \exists Y_{bu} \in \hat{K}_{i+1}$ s.t. $Y_{bu} \subseteq X$, then $\forall Z \subseteq Y_{td}$ of size $\geq \lceil n/2 \rceil$, Z is not a minimal key.*

Proof: Straight-forward and omitted.

QED

This lemma suggests that all subsets of Y_{td} of size $n-i$ need be checked in the pruning approach above. Such a cost would make the approach prohibitively expensive. However, as will be shown later, the elements of NK_{n-i-1} allow this cost to be avoided. The cost will amount at worst to $O(n^2|NK_{n-i-1}|^2)$. Since the NK_{n-i-1} sets need be kept around by the algorithm anyway, the $|NK_{n-i-1}|^2$ factor is not an overhead problem.

We are now in a position to give a top-down, bottom-up, hybrid algorithm in which the top-down part utilizes temporary information gathered by the bottom-up part and vice versa. In particular this algorithm works very fast on instances r_{td} and r_{bu}. This algorithm is presented in figure 4. We frequently refer to this as *the hybrid with communication algorithm*

Input: R, a relation schema of size n and r an instance of R of size m.
Output: K, the set of minimal keys of R with respect to r.

```
1     K, K_1, ..., K_{n-1} := ∅ and NK_0 = {∅}, NK_1, ..., NK_n := ∅ and K_n := {R}
COMMENT   Lines 2-13 are the top-down portion
2     for i := n − 1 downto ⌈n/2⌉ do
3       for Y_td ∈ K_{i+1} do
COMMENT   Line 4 is communication with the bottom-up portion
4         if i = ⌈n/2⌉ or ∃X ⊆ Y_td of size n − i, s.t. ∀Y_bu ∈ K,
          Y_bu ⊈ X, then
5           for X_td ⊆ Y_td s.t. |X_td| = i do
6             if X_td is a key of R w.r.t. r, then
7               K_i := K_i ⊔ {X_td}
8               mark Y_td
9           endfor
10        else mark Y_td
11      endfor
12      remove all marked Y_td from K_{i+1}
13      remove all duplicates from K_i
COMMENT   Lines 14 - 26 are the bottom portion
14      for Y_bu ∈ NK_{n−i−1} do
COMMENT   Line 15 is communication with the top-down portion
15        if ∃X_td ∈ K_i s.t. Y_bu ⊆ X_td, then
```

```
16              for X_bu ⊇ Y_bu s.t. |X_bu| = n − i do
17                 if X_bu is not a key w.r.t. r then
19                    NK_{n−i} := NK_{n−i} ⨆ {X_bu}
20                 else if ∀Z ⊆ X_bu with |Z| = n − i − 1, we
21                       have: Z is not a key w.r.t. r then
22                    K := K ⨆ {X_bu}
23              endfor
24        endfor
25        remove duplicates from NK_{n−i}
26        remove duplicates from K
27     endfor
COMMENT    28-33 – merging of top-down and bottom-up
28     for i := n − 1 downto ⌈n/2⌉ do
29        for X_td ∈ K_i do
30           if ∀Z ∈ K with |Z| ≤ ⌈n/2⌉, Z ⊈ X_td, then
31              K := K ⨆ {X_td}
32        endfor
33     endfor
```

Figure 4: "Hybrid" Algorithm using top-down and bottom-up pruning with communication

A correctness proof of this algorithm can be obtained from the correctness proofs of the top-down and bottom-up algorithms and lemmas 4, 5. In particular, nearly the same invariance results are obtained.

Lemma 6 *For all $n-1 \geq i \geq \lceil n/2 \rceil$ the following statements hold at the end of step 27 of iteration i of the hybrid algorithm with communication (e.g. figure 4):*

1. *For all $n \geq j \geq i$ and each $Y \in K_j$, there does not exist Z a key of size $\geq i$ and $Z \subsetneq Y$*
2. *For all keys Y of size $n \geq j \geq i$, if there does not exist a key Z of size $\geq i$ such that $Z \subsetneq Y$, then $Y \in K_j$.*
3. *$NK_{n-i} = \{X \subseteq R | |X| = n - i$ and X is not a key $\}$.*

Proof: Very similar to the proofs of lemmas 2.1 and 3. Some extra detail is needed to handle the communication steps. However, these details are not very illuminating and so the proof is omitted.

QED

From this invariance lemma, a correctness proof of the hybrid with communication algorithm follow directly (and is omitted). Additionally, some complexity results explained later use this invariance lemma.

We are interested in a complexity analysis of the hybrid with communication algorithm. In particular we are interested in how effective the communication is with regard to Mannila's problem. To do so we consider three questions:

1. On which inputs does the hybrid algorithm without communication exponentially fail to solve Mannila's problem but the hybrid algorithm with communication solves the problem?
2. On which inputs does the hybrid algorithm with communication exponentially fail to solve Mannila's problem?
3. How does the worst-case running time of the hybrid with communication compare to the worst-case without communication (e.g. when is the communication overhead too much)? In particular, on which inputs does the hybrid algorithm with communication exponentially fail but the hybrid without communication does not?

2.4.1 Improvement Through Communication

To answer question 1., we develop a theorem which (together with theorem 4) describes a sub-class of inputs for which the hybrid with communication solves Mannila's problem but the hybrid without communication exponentially fails. The theorem is based on an observation concerning the behavior of the hybrid algorithm with communication on inputs for which the minimal keys are either very large or very small (or both). The communication allows the algorithm to ignore searching the "middle" part of the power-set lattice of R. As a result, Mannila's problem is solved on these inputs.

The next lemma makes precise the relationship of very small or very large minimal keys and communication.

Lemma 7 *Given $\theta \in (0, 1/2)$ and R a relation schema (of size n) and r an instance of R (of size m) such that: $n \geq 16$ and $\lceil \theta log n \rceil < (n-2)/3$ and all minimal keys of R with respect to r are of size $\leq \lceil \theta log n \rceil$, or of size $\geq n - \lceil \theta log n \rceil$. Let $i = n - \lceil \theta log n \rceil - 1$. At the end of iteration i of the hybrid algorithm with communication: $K_i = \emptyset$ and $NK_{n-i} = \emptyset$.*

[3] **Proof:** First we show that: after step 13 during iteration i, $K_i = \emptyset$ (and hence, at the end of iteration i, $K_i = \emptyset$). To do so, it suffices to show that nothing is added to K_i during steps 2-11.

[3] $log n$ is assumed throughout to be of base 2.

Let $Y_{td} \in K_{i+1}$ at the start of iteration i. Since $n \geq 16$, then $i > \lceil n/2 \rceil$, so, the second condition of step 4 must be checked for Y_{td}. Let $X \subseteq Y_{td}$ of size $n - i (= \lceil \theta log n \rceil + 1)$. Since $n - \lceil \theta log n \rceil > |X| > \lceil \theta log n \rceil$ (because $n \geq 16$), then, by assumption, X is not a minimal key.

If X is a key, then $\exists Y_{bu} \in K$ at the start of iteration i, such that: $Y_{bu} \subsetneq X$. Therefore, step 4 will fail for X. Thus, no subset of Y_{td} will be added to K_i during iteration i.

Assume X is not a key. By assumption, it follows that $\forall Z \supseteq X$ of size i, Z is not a key. It follows that: for all $\hat{X} \subseteq R$ of size $\leq i - |X|$, $\hat{X}$ is a subset of some $Z \supseteq X$ of size i (which is not a key). Therefore, for all $\hat{X} \subseteq R$ of size $\leq i - |X|$, $\hat{X}$ is not a key. We know that:

$$\begin{aligned} i - |X| &= i - \lceil \theta log n \rceil - 1 \quad (\text{ since } |X| = \lceil \theta log n \rceil + 1) \\ &= n - 2\lceil \theta log n \rceil - 2 \quad (\text{ since } i = n - \lceil \theta log n \rceil - 1) \\ &\geq (n-2)/3 \quad (\text{ since } \lceil \theta log n \rceil < (n-2)/3) \\ &\geq \lceil \theta log n \rceil \quad (\text{ since } n \geq 16). \end{aligned}$$

It follows that: for all $\hat{X} \subseteq R$ of size $\leq \lceil \theta log n \rceil$, $\hat{X}$ is not a key. Therefore, by original assumption regarding the size of the minimal keys, it follows that there are no keys of size $\leq i$. Hence, no subset of Y_{td} will be added to K_i during steps 2-11.

We conclude that $\forall Y_{td} \in K_{i+1}$, no subset of Y_{td} is added to K_i during steps 2-11.

To complete the proof we need show that: at the end of iteration i, $NK_{n-i} = \emptyset$. However, this follows from the fact (shown earlier) that at step 14 during iteration i, $K_i = \emptyset$.

QED

The first main theorem can now be proved. This theorem describes a sub-class of inputs on which the hybrid algorithm with communication solves Mannila's problem.

Theorem 5 *Given $\theta \in (0, 1/2)$ and R a relation schema (of size n) and r an instance of R (of size m) such that all minimal keys of R with respect to r are of size $\leq \lceil \theta log n \rceil$, or of size $\geq n - \lceil \theta log n \rceil$. The running time of the hybrid algorithm with communication is at most: $O(n^6|K|mlogm + n^7 mlogm + n|K|log(|K|) + n|K|^2)$.*

Proof: Let $n - i \geq i \geq \lceil n/2 \rceil$. Consider the amount of time required by the algorithm during iteration i (without merging). There are two cases to consider.

1. **Case:** $i \geq n - \lceil \theta log n \rceil - 1$. A Naive analysis will yield that steps 3-13 require time:

$$O(|K_{i+1}| \binom{n}{n-i} |K|(i+1)mlog(m) + |K_{i+1}|log(|K_{i+1}|) + |K_i|log(|K_i|)).$$

The $\binom{n}{n-i}|K|$ term comes from the communication overhead in step 4. Since $K_{i+1} \leq \binom{n}{n-i}$, then steps 2-13 require time at most:

$$O(\binom{n}{n-i}^2 |K|(i+1)mlog(m)).$$

Since $i \geq n - \lceil \theta log n \rceil - 1$, then an application of Stirling's approximation will yield an upper-bound of

$$O(n^5|K|mlogm).$$

A naive analysis will yield that steps 14-26 require time

$$O(|NK_{n-i-1}||K_i|(i+1)[mlogm+(i+1)mlog(m)]+|NK_{n-i}|log(|NK_{n-i}|)+|K|log(|K|)).$$

Employing bounds in an analogous fashion to that above an upper-bound yields an upper-bound of

$$O(n^6 mlogm + |K|log(|K|)).$$

We conclude that the algorithm requires time $O(n^5|K|mlogm + n^6 mlogm + |K|log(|K|))$.

2. **Case:** $i < n - \lceil \theta log n \rceil - 1$. By lemma 7 (for sufficiently large n), at the start of iteration i, $K_{i+1} = \emptyset$ and $NK_{n-i-1} = \{\emptyset\}$. It follows that steps 3-13 require time at most $O(1)$ and after step 13 during iteration i, $K_i = \emptyset$. Therefore, steps 14-24, require time at most $O(1)$. Since $NK_{n-i} = \{\emptyset\}$ (by lemma 7), then step 25 requires $O(1)$ time. Step 26 will require time $O(|K|log(|K|))$.

   We conclude that the algorithm requires time $O(|K|log(|K|))$ during iteration i.

Combining the cases and summing over all iteration, we have a running time on steps 1-27 (e.g. everything except merging) of

$$O(n^6|K|mlogm + n^7 mlogm + n|K|log(|K|))$$

A naive upper-bound on the time required for merging is:

$$O(n|K|^2).$$

Therefore, the algorithm requires a total running time of at most:

$$O(n^6|K|mlogm + n^7 mlogm + n|K|log(|K|) + n|K|^2).$$

QED

Theorems 4 and 5 imply that, on the following class of inputs, $\mathcal{C}_1$, the hybrid algorithm with communication solves Mannila's problem but the hybrid algorithm without communication exponentially fails. $\mathcal{C}_1$ is the class of all relation schema, instance pairs (R, r) (of sizes n, m, respectively) where there exists $\theta \in (0, 1/2)$ and:

- there exists only a polynomial number of minimal keys in n
- all minimal keys of R with respect to r are of size $\leq \lceil \theta log n \rceil$, or of size $\geq n - \lceil \theta log n \rceil$.

$\mathcal{C}_1$ provides an answer to question 1. Take note that with $R = \{A_1, \ldots, A_n\}$, both (R, r_{td}) and (R, r_{bu}) are in $\mathcal{C}_1$.

2.4.2 Lower-bound on the Running Time with Communication

To answer question 2., we develop a rough lower-bound on the running time of the hybrid algorithm with communication. This bound implies that, for a reasonably large class of inputs, the algorithm requires at least exponential time in n. Hence, for instances in this class with only a polynomial number of minimal keys in n, the algorithm fails exponentially to solve Mannila's problem. However, on these inputs the hybrid without communication also fails exponentially. So, the communication overhead is not to blame for the exponential failure in the hybrid with communication in this class. Nonetheless, the communication does not prevent exponential failure, either.

Assume there exists a minimal key, α, of size bigger than one. The next lemma will show that the communication does not allow any supersets of α to be pruned from K_i until iteration $n - |\alpha| - 1$. This is essentially due to the fact that K will not contain α until iteration $n - |\alpha|$. As a result step 4 will pass for supersets Y_{td} of α since X can be chosen as a subset of α.

Lemma 8 *Given R a relation schema (of size n) and r an instance of R (of size m) such that, there exists a minimal key α of size > 1, $\forall n - 1 \geq i \geq max\{n - |\alpha|, \lceil n/2 \rceil\}$, at the start of iteration i of the hybrid algorithm with communication, $S_\alpha^{i+1} =^{def} \{X \subseteq R || X| = i + 1$ and $\alpha \subseteq X\} \subseteq K_{i+1}$.*

Proof: By induction on i.

- **Base Case ($i = n - 1$):** Follows immediately from the initialization conditions.
- **Inductive Case:** Let $n - 1 \geq i > max\{n - |\alpha|, \lceil n/2 \rceil\}$. Assume that at the start of iteration i, $S_\alpha^{i+1} \subseteq K_{i+1}$. Consider now the situation at the start of iteration $i - 1$.

  Let $Z \in S_\alpha^i$. By definition, $|Z| = i$ and $\alpha \subseteq Z$ (so, Z is a key). Since $n - 1 \geq i > n - |\alpha|$, then $1 \leq n - i < |\alpha|$, so, there exists $X \subsetneq \alpha$ of size $n - i \geq 1$.

Let Y_{td} be a superset of Z of size $i+1$ (so $\alpha \subseteq Y_{td}$). By definition, $Y_{td} \in S_\alpha^{i+1}$. By induction assumption it follows that at the start of the i^{th} iteration, $Y_{td} \in K_{i+1}$. So at some point during the i^{th} iteration, step 4 will be reached with Y_{td} as above. Since X is of size $n-i$ and $X \subsetneq \alpha \subseteq Y_{td}$, then X applies to step 4. Suppose there exists $Y_{bu} \in K$ such that $Y_{bu} \subseteq X$, then $Y_{bu} \subsetneq \alpha$. Since $Y_{bu} \in K$, then Y_{bu} is a key, so, α is not a minimal key. This contradicts our original assumption, so, for all $Y_{bu} \in K, Y_{bu} \nsubseteq X$. Step 4 will thus pass.

Step 6 will be reached at some point with Y_{td} as above and $X_{td} = Z$ (since $Z \subseteq Y_{td}$ and $|Z| = i$). Since Z is a key, then step 6 will pass, so, Z will be added to K_i. Therefore, $Z \in K_i$ at the start of iteration $i-1$. We conclude that at the start of iteration $i-1$, $S_\alpha^i \subseteq K_i$.

QED

Lemma 8 will give a lower-bound on the size of K_i and so a rough lower-bound on the running time as the next lemma shows.

Lemma 9 *If there exists a minimal key of size, $\ell > 1$, then the hybrid algorithm with communication requires a running time of at least (where $\binom{x}{y}$ is assumed to be zero for $x < y$):*
$\Omega(\sum_{i=max\{n-\ell,\lceil n/2 \rceil\}}^{n-1} \binom{n-\ell}{n-i-1})$.

Proof: Assume there exists, α, a minimal key of size ℓ. By definition, $\forall n-1 \geq i \geq max\{n-\ell, \lceil n/2 \rceil\}, |S_\alpha^{i+1}| = \binom{n-\ell}{i+1-\ell} = \binom{n-\ell}{n-i-1}$. Hence from lemma 8, at the start of iteration i, $|K_{i+1}| \geq \binom{n-\ell}{n-i-1}$. Step 3, during iteration i, will need to look at at least $\binom{n-\ell}{n-i-1}$ Y_{td} sets. Hence, at iteration i the algorithm will require $\Omega(\binom{n-\ell}{n-i-1})$ time.

As a result the algorithm will require time at least: $\Omega(\sum_{i=max\{n-\ell,\lceil n/2 \rceil\}}^{n-1} \binom{n-\ell}{n-i-1})$.

QED

Finally, we can prove our main theorem. This theorem specifies a reasonably large class of relation instances on which the algorithm requires at least exponential time in n.

Theorem 6 *Let $\theta \in (0, 1/2)$. Given relation schema, R (of size n) and r a relation instance of R (of size m) such that R has a minimal key of size $\lfloor \theta n \rfloor$ with respect to r, then on R, r the hybrid algorithm with communication requires at least time $\Omega(2^{\lfloor \theta n \rfloor - 1})$.*

Proof: Since R has a minimal key of size bigger than one (for sufficiently large n) with respect to r where $n - \lfloor \theta n \rfloor \geq \lceil n/2 \rceil$, then by lemma 9, it follows that on R, r the

algorithm requires time at least:

$$\Omega(\sum_{i=n-\lfloor\theta n\rfloor}^{n-1}\binom{n-\lfloor\theta n\rfloor}{n-i-1}).$$

Simplifying the sum a bit we get:

$$\Omega(\sum_{j=0}^{\lfloor\theta n\rfloor-1}\binom{n-\lfloor\theta n\rfloor}{j}).$$

Since $n-\lfloor\theta n\rfloor \geq \lfloor\theta n\rfloor-1$, then $\binom{n-\lfloor\theta n\rfloor}{j} \geq \binom{\lfloor\theta n\rfloor-1}{j}$. Therefore, we get a lower-bound on the running time of:

$$\Omega(\sum_{j=0}^{\lfloor\theta n\rfloor-1}\binom{\lfloor\theta n\rfloor-1}{j}) = \Omega(2^{\lfloor\theta n\rfloor-1}).$$

QED

This theorem implies that the following class of inputs, $\mathcal{C}_2$, cause the hybrid algorithm with communication to exponentially fail to solve Mannila's problem. $\mathcal{C}_2$ is the class of relation schema, instance pairs (R, r) (of sizes n, m, respectively) where $\exists\theta \in (0, 1/2)$ and:

- R has a minimal key with respect to r of size $\lfloor\theta n\rfloor$
- $\lfloor n-\theta n\rfloor \geq \lceil n/2\rceil$.

$\mathcal{C}_2$ provides as answer to question 2..

2.4.3 Too Much Communication Overhead?

To answer question 3., we first compute the running time of the hybrid with communication algorithm with respect to the hybrid without communication algorithm. For $n-1 \geq i \geq \lceil n/2\rceil$, let $CommOverTD_i$ denote the worst case running time of step 4 of iteration i of the hybrid with communication algorithm. Let $CommOverBU_i$ denote the worst case running time of step 15 of iteration i of the hybrid with communication algorithm. These expressions represent the communication overhead.

Inspection of the hybrid with and without communication algorithms shows that the running time (pre-merging) of the hybrid with communication algorithm at iteration i is at most a factor of $CommOverBU_iCommOverTD_i$ more than the hybrid without communication algorithm. A naive analysis shows that $CommOverBU_i$ is

$$O(|K_i|).$$

Determining a reasonable upper-bound on $CommOverBU_i$ is more challenging. To do so, we show that for any $Y_{td} \in K_{i+1}$, all subsets of Y_{td} of size $n-i$ need not be examined. The close relation between NK_{n-i-1} and $\hat{K}_{i+1}$ allows a more reasonable approach. The following lemma makes this approach more clear.

Lemma 10 *For $n-1 > i \geq \lceil n/2 \rceil$, at the end of iteration, $i+1$, (pre-merging) of the hybrid with communication algorithm, $\forall Y_{td} \in K_{i+1}$:*

$\exists X \subseteq Y_{td}$ of size $n-i$ s.t. $\forall Y_{bu} \in K, Y_{bu} \nsubseteq X$ if and only if $\exists X \subseteq Y_{td}$ of size $n-i$ s.t. $\forall \hat{X} \subseteq X$ of size $n-i-1$, $\hat{X} \in NK_{n-i-1}$.

Proof: Straight-forward and omitted.
QED

Based on this lemma, the following algorithm can be used to check the conditional of step 4 at iteration i of the hybrid with communication algorithm on Y_{td}.

```
1    for Y_bu ∈ NK_{n-i-1}
2       for X ⊇ Y_bu of size n-i
3          if ∀X̂ ⊆ X of size n-i-1, X̂ ∈ NK_{n-i-1}, then
4             return true
5       endfor
6    endfor
7    return false
```

Therefore, A naive analysis shows that $CommOverTD_i$ is

$$O(|NK_{n-i-1}|^2(n-i)^2).$$

We have that the running time (pre-merging) of the hybrid with communication algorithm at iteration i is at most a factor of $|K_i||NK_{n-i-1}|^2(n-i)^2$ more than the hybrid without communication algorithm. Since the hybrid without communication algorithm must look at K_i and NK_{n-i-1} during iteration i anyway, then running time of the hybrid with communication is within a power of five of the running time of the hybrid without communication. In particular, there does not exists any instances on which the hybrid with communication fails exponentially to solve Mannila's problem but the hybrid without communication does not.

We conclude that the communication overhead is not infeasible with respect to the hybrid algorithm without communication.

2.4.4 Communication Conclusions

Providing communication between the top-down and bottom-up algorithms is a difficult problem. The hybrid with communication attempts to address this problem. In an effort to gauge the effectiveness of this algorithm three questions were put forth. The first two questions were designed to point out classes of inputs in which the communication works quite well and where the communication fails to work well, both with regard to Mannila's problem. The third question was designed to point out the size of the communication overhead with respect to the running time of the hybrid algorithm without communication.

A class of inputs, $\mathcal{C}_1$, provides an answer to question 1. This class, intuitively, includes all inputs which have only minimal keys which are very small (roughly $log(n)$ in size) or (inclusive) very large (roughly $n - log(n)$ in size). This result indicates that some success is achieved. Namely, the addition of communication yields a solution to Mannila's problem on a fairly wide class of input instances.

A class of inputs, $\mathcal{C}_2$, provides an answer to question 2. This class points out a major weakness in the hybrid algorithm with communication. On instances with a minimal key in the "middle" portion of the power-set lattice of R, the algorithm must do an exponential amount of work in n. However, the communication overhead is not to blame, because the hybrid without communication performs nearly as badly. Nonetheless, the communication does not prevent exponential failure, either. In this respect the communication added is too weak.

An answer to question 3 is provided as follows. The running time (pre-merging) of the hybrid with communication algorithm at iteration i is at most a factor of $|K_i||NK_{n-i-1}|^2(n-i)^2$ more than the hybrid without communication algorithm. Therefore, the running time of the hybrid with communication is within a power of five of the running time of the hybrid without communication. In particular, there does not exist any inputs on which the hybrid with communication fails exponentially to solve Mannila's problem but the hybrid without communication does not.

A much more interesting (and difficult) result to obtain would be the tradeoff between the communication overhead and the savings gained through pruning. For this result a finer analysis of the communication overhead is needed. Getting at this tradeoff analytically, is a difficult problem. One which we shall leave as future work.

In general we believe that the successes achieved though communication are noteworthy. The idea of developing a hybrid level-wise approach to key-finding has its merits. In order for this approach to be worthwhile, some assurance is needed that the merits will move us significantly closer to a solution to Mannila's problem. In this regard, a much better understanding of the tradeoff between the communication overhead and the savings gained through pruning is needed. At this point, our intuition of this tradeoff is primitive. A more detailed analysis would be helpful in clarifying

this intuition. However, such an analysis is likely to be quite difficult. In our opinion, it seems unlikely that developing a deeper understanding of this tradeoff will move us significantly closer without substantial effort and innovation.

3 Empirical Results

3.1 Levelwise A-priori

The algorithm in figure 1 (§2.1) was implemented for the case of Bernoulli databases. The code can be found in the following file:

`/u/crood/courses/DataMining/imp/ver4/main.c`

A randomized Bernoulli database (having no repeated tuples) is generated of the appropriate width and length, and then the top-down naieve key-finding function is applied. Table 1 shows the results obtained. Note that we use an efficient key-checking algorithm based on a version of quicksort. The algorithm is given in figure 5.

Input: $X = \{A_1, ..., A_k\}$, a set of attributes;
$r = \{t_1, ..., t_m\}$ a database instance
Output: TRUE if X is a key for r, FALSE otherwise

if $(|r| > 2^{|X|})$ return FALSE;
QuickSort(r, X); // Sort r on attributes in X
$i = 1$;
$j = 2$;
while $(i <= m - 1$ **and** $j <= m)$
  if $t_i = t_j$ return FALSE; // duplicates will be next-to-next
return TRUE;

Figure 5: Key-checking predicate: determines if the set X is a key according to the database instance r.

Table 1 shows results for the worst, average, and best cases of the top-down naieve algorithm, respectively. Note that the worst case occurs when there is only one tuple in the database, since in this case every subset of size 1 is a minimal key. The exponential behavior of the algorithm can be clearly seen in the right hand column (total number of subsets generated during the run). The running times (not shown) for the worst case seem to be $O(4^n)$. The running time is higher than expected due to several reasons, including:

- We must maintain the K_i subsets that contain the levelwise frontier sets. This requires quite a bit of overhead.
- Our method for generating the next level of subset candidates generates repeats. A method for generating levelwise successor subsets from K_i *without* any repeats was not found; however, a *depth-first* method for generating subsets without repeats is easy to code (§3.2).

Note that the average case (when $m = 2^{\lceil n/4 \rceil}$, so the expected minimal key size is $n/2$ as in [7]) is exponential as well. The running times in this case seem to more closely follow a $O(2^n)$ curve. The best case occurs when the minimal keys are large; this is most likely to occur when $m > 2^{\lceil n/2 \rceil}$ (as discussed in [7]). In the best case, the running time is $O(nmlog(m))$.

The other levelwise A-priori key-finding algorithms were not implemented. There are two main reasons for this:

1. It was determined early on that keeping the levelwise frontier sets around for the lifetime of the key-finding function was even more infeasible than the $O(2^n)$ running time. Maintaining the K_i in the top-down naieve a-priori is difficult and highly space-consuming.[4]

   The space complexity problem is exacerbated by the fact that we have discovered no good algorithm to generate the next level of (1 element larger) subsets given a particular K_i in a levelwise manner without generating duplicate subsets along the way.

2. The second reason for not continuing with the implementation of the levelwise a-priori algorithms is that we have discovered an interesting alternate approach: that of *partition trees*. Partition trees are discussed in appendix A. These data-structures arise when one views the key-finding search space not as the power-set lattice of R, but as a *partition tree*, in which the various ways of partitioning the relation instance r on the attributes in R are reflected.

In summary, the space complexity of the levelwise a-priori algorithms is a major problem in continuing any implementation. In the next section (§3.2), we indicate how *depth-first* versions of the top-down and bottom-up a-priori algorithms could be more feasible (in particular with regard to the space issue), and illustrate a simple, "switch hybrid" algorithm based on the expected size of minimal keys.

[4] There is also a mismatch in computing speed and memory resources: it is feasible, for example, to require 10^9 processor instructions routinely in an algorithm, but it is not feasible to require 10^9 bytes of storage space routinely.

m	n	$\lvert K\rvert$	average key length	number of subsets generated
1	10	10	1	5111
	11	11	1	11254
	12	12	1	24565
	13	13	1	53236
	14	14	1	114675
	15	15	1	245746
	16	16	1	524273
	17	17	1	1114096
	18	18	1	2359279
	19	19	1	4980718
	20	20	1	
$2^{\lceil n/4\rceil}$	10	41	5	2865
	11	30	4	4524
	12	113	4	17792
	13	144	6	22033
	14	456	6	65286
	15	491	6	137361
	16	972	6	347253
	17	2060	8	494174
	18	1800	8	708395
	19	6181	8	2852585
	20	8095	8	6631346
$2^{\lceil n/2\rceil+1}$	10	1	9	20
	11	1	11	12
	12	1	12	13
	13	1	13	14
	14	3	13	54
	15	1	15	16
	16	1	15	32
	17	1	16	34
	18	1	17	36
	19	1	19	20
	20	3	19	78

Table 1: Results of the top-down, naieve A-priori algorithm on randomized Bernoulli Databases.

3.2 Depth First A-priori

In this section, we present depth-first versions of the top-down and bottom-up a-priori search algorithms. Although their asymptotic running times are similar to those of the levelwise algorithms, depth-first search through the power-set lattice of subsets of R is more space efficient and allows us to solve the "subset duplication" problem. Furthermore, the depth-first versions of the a-priori algorithms will provide both motivation for and contrast to the partition trees introduced in appendix A.

Given $R = \{A_1, ..., A_n\}$, we introduce an ordering on the attributes, e.g. $A_1 < ... < A_n$. This will allow us to generate all subsets of R without repeats. The depth-first, bottom-up a-priori algorithm is given in figure 6.

Function DFBU(Att, X)
Input: Att, the current attribute;
X, the current subset to be checked;
Global variables: $R = \{A_1, ..., A_n\}$ a relation schema;
$r = \{t_1, ..., t_m\}$ an instance of R
Output: K, the set of minimal keys of r

if (Key(X, r)) insert X into result K;
else for $A_j > Att$ **do**
DFBU(A_j, $X \cup \{A_j\}$);

Figure 6: Depth-first, bottom-up, a-priori key finding.

The algorithm in figure 6 builds subsets of R recursively, one attribute at a time. The top-level call (in the main program function) is "DFBU(0, $\emptyset$)" where 0 is a special attribute that is less than all other attributes (in our ordering), i.e. $0 < A_1 < A_2 < ... < A_n$, to start the recursion off. The bottom-up "a-priori" principle is obscured by the simplicity of the algorithm: note that in the base case of the recursion (where the current attribute set X is a key), we do not make a recursive call, hence the subsets larger than that X will not be generated. This is in accordance with the "a-priori" principle that if X is a key then no proper superset of X can be a minimal key.

The depth-first, top-down, a-priori algorithm is similar; it is given in figure 7.

The top-down version looks very similar to the bottom-up version, but there are two crucial differences. The top level call for the top-down version is "DFTD(0, R)" (versus the top level call for the bottom-up version which started with $X = \emptyset$). The second crucial difference is that in the top-down version we are *removing* attributes one-by-one recursively, as opposed to adding them in. The top-down a-priori principle is not obvious in the simple algorithm in figure 7, but is encapsulated in the "found" boolean value. If "found" is true at the end of the **for** loop, that means every subset

Function DFTD(Att, X)
Input: Att, the current attribute;
X, the current subset to be checked;
Global variables: $R = \{A_1, ..., A_n\}$ a relation schema;
$r = \{t_1, ..., t_m\}$ an instance of R
Output: K, the set of minimal keys of r

found = TRUE;
for $A_j > Att$ such that Key($X - \{A_j\}$, r) **do**
found = FALSE;
DFTD(A_j, $X - \{A_j\}$);
if (found) insert X into result K;

Figure 7: Depth-first, bottom-up, a-priori key finding.

of size one less than X failed to be a key. In this case, X is the minimal key sought. This is in accordance with the top-down a-priori principle, namely that a subset X is a minimal key only if no proper subset of it is a key.

A simple hybrid algorithm, "SwitchHybrid" is given in figure 8. SwitchHybrid uses the results of [7] to predict the average size of minimal keys. If these keys are expected to be larger than $n/2$ in size (corresponding to $2^{\lceil n/2 \rceil} \leq m \leq 2^n$), the top-down algorithm is called. If the keys are expected to be small (less than $n/2$ in size), the bottom-up algorithm is called.

Function SimpleHybrid()
Global variables: $R = \{A_1, ..., A_n\}$ a relation schema;
$r = \{t_1, ..., t_m\}$ an instance of R
Output: K, the set of minimal keys of r

if $m > 2^{\lceil n/2 \rceil}$
DFTD(0, R);
else
DFBU(0, $\emptyset$);

Figure 8: Simple "switch" hybrid based on the expected minimal key size.

The function SwitchHybrid is simple, and will likely perform well in cases where the

expected key size is either small or large – recall the class $\mathcal{C}_1$, page 25. SimpleHybrid will perform well (polynomially) on most of the Bernoulli members of this class. When the expected minimal key size is close to $n/2$, the algorithm's performance will be exponential in n. Recall the class $\mathcal{C}_2$ (page 27); SimpleHybrid will perform poorly on the Bernoulli members of this class. In fact, SwitchHybrid's performance in the average case should meet or exceed that of the levelwise hybrid with communication (in the case of Bernoulli databases). SwitchHybrid serves as a simpler and more feasible version of the hybrid a-priori algorithms. It would be interesting to code a version, for use as a benchmark for comparing the partition tree algorithms of appendix A.

Note that the SwitchHybrid only works in the case of Bernoulli databases. In the general case, it is less clear what the expected size of minimal keys is. Furthermore, it is unclear how deeper communication between the depth-first versions of the top-down and bottom-up algorithms would be achieved.

4 Conclusions

4.1 Problems with the A-Priori Approach

There are several problems with the a-priori approach we have been pursuing in the bulk of this paper, including:

1. Implementation issues with the levelwise approach:
   (a) The space-complexity is prohibitive.
   (b) The problem of generating repeated subsets is intrinsic and difficult to overcome.

   Both these issues can be overcome in the Bernoulli case with a depth-first implementation of the bottom-up and top-down a-priori algorithms, however for the general case it is less clear what a depth-first version of the communication hybrid would entail.

2. Communication issues with the hybrid algorithm: as discussed in §2.4.1, enabling the top-down and bottom-up levelwise a-priori algorithms to communication information is tricky. Due to space problems mentioned earlier, this approach is not of practical interest. However, there does seem to be theoretical motivations for further study. In particular, the development of a deeper understanding of the tradeoff between the communication overhead and savings gained though pruning is interesting. On the other hand, such a development is likely to be a very difficult problem. It is unclear whether the potential discoveries would be of enough interest to justify the effort.

3. One issue not previously discussed is the issue of *informed* versus *uninformed* search. The a-priori algorithms are essentially *uninformed*, in the sense that they do not take advantage of the actual database instance r in deciding which subsets of R to generate. A more informed search is described in appendix A, where the database instance is used in a more essential manner to actually generate potential key candidates.

None of the a-priori algorithms seem to solve Mannila's problem in the general case.

4.2 Further Directions

As indicated in the previous section, several problems with the a-priori approach to key finding motivated a shift to more informed searching algorithms of the type presented in appendix A. The partition tree algorithms are the subject of ongoing thought and research. We see three avenues of further research on Mannila's problem:

1. In some respects, Mannila's key-finding problem seems poorly motivated. It would be interesting to find some further areas where a solution to Mannila's problem could apply in an integral way.
2. The level-wise hybrid with communication algorithm has some interesting theoretical issues. Namely, developing a deeper understanding of the tradeoff between the communication overhead and the savings gained through pruning. However, the effort required is likely to be quite large. We shall not pursue this avenue further. Mainly because the advances made with the partitioning algorithms of appendix A seem a more promising direction of research.
3. The partition tree algorithms (appendix A) seem to be a reasonably efficient solution to Mannila's problem in the case of Bernoulli databases. Further study is required to determine whether the partition tree algorithms actually solve Mannila's problem for this class of databases.
4. Mannila's key-finding problem for general databases has been linked to a problem not yet known to be in P but also not yet known to be NP-complete. It seems that the general case is significantly harder than the Bernoulli case, since no such linkage exists for the case of Bernoulli databases. Another avenue of further study would be to investigate the differences between the general and Bernoulli cases of Mannila's problem. Such a study would likely be closely tied with 2 (above).

In summary, studying the a-priori algorithms has been fruitful in two ways: first, it has given us some insight into Mannila's key-finding problems and the difficulties with the general case, and second it has led to alternate algorithms which may be very efficient in the Bernoulli case. We look forward to pursuing some (or all) of the avenues listed above.

A Partition Trees

A.1 Basic Partition Tree Algorithm

What follows is a brief introduction to our work on *partition trees*, or *P-trees*. The idea behind the P-tree algorithm is to use the database instance r in a more integral fashion to guide the search through the powerset lattice. At each stage in the algorithm, we partition the database instance (using the same partitioning idea as in quicksort) according to an attribute. Note that the P-tree algorithms are specialized to the case of Bernoulli databases, and will not work in the general case in their current form. Note that we also assume there are no repeated tuples in the database instance r.

We need to introduce some notation. Given a set of tuples of r, $s = \{t_1, ..., t_k\} \subseteq r$, the *difference set* of s, differSet(s) is the set of attributes that distinguish at least one pair of tuples. That is:

$$\text{differSet}(s) = \{A | \exists t_1, t_2 \in s \text{ such that } t_1[A] \neq t_2[A]\}$$

Note that attributes *not* in differSet(s) cannot contribute to minimal keys for s.

We define two operations on sets of keys. Let K_1 and K_2 be sets of keys (i.e. sets of sets of attributes) and define:

$$K_1 \odot K_2 = \{X | X \in K_1 \cup K_2 \text{ and } \nexists Y \in K_1 \cup K_2 \text{ such that } Y \subsetneq X\}$$
$$K_1 \otimes K_2 = \{X \cup Y | X \in K_1 \text{ and } Y \in K_2 \text{ and } \nexists X' \in K_1, Y' \in K_2 \text{ such that } X' \cup Y' \subsetneq X \cup Y\}$$

The operation "$\odot$" is essentially a minimalizing union, returning the keys in K_1 and K_2 that do not properly contain any other key in the union. The operation "$\otimes$" is also minimalizing. It is an "inner union", in that the union takes place on the keys from K_1 and K_2 (as opposed to K_1 and K_2 themselves).

Given these operations, the basic P-tree algorithm is given in figure 9. Note that the search pattern is similar to the depth-first, bottom-up a-priori algorithm presented in §3.2. The difference here is that we are partitioning the database instance at each stage, using the database instance to constrain the search.

Note that in the average case of a randomized Bernoulli database, at each stage in the recursive call the database will partition pretty much evenly into a left and right half. Thus the height of the P-tree in this case is bounded by $log_2(m)$. The number of nodes in the tree may still be exponential. The following section indicates some optimizations and observations that are aimed at reducing (or further characterizing) the branching factor in P-trees.

Function PTREE(Att, s)
Input: Att, the current attribute considered
s, a database partition ($s \subseteq r$)
Global variables: $R = \{A_1, ..., A_n\}$ a relation schema;
$r = \{t_1, ..., t_m\}$ an instance of R
Output: K, the set of minimal keys of r

Base Cases:
if ($|s| > 2^{n-Att}$) // not enough attributes left
return $\emptyset$;
if ($s = \{t_1, t_2\}$)
return differSet(s);
if ($s = \{t_1\}$)
return $\cup_j \{A_j\}$ where $A_j > Att$

Recursive Case:
res = $\emptyset$;
for $A_j > Att$ such that $A_j \in$ differSet(s) **do**
left_keys = PTREE(A_j, $\sigma_{A_j=1}(s)$);
right_keys = PTREE(A_j, $\sigma_{A_j=0}(s)$);
res = res $\odot$ (left_keys $\otimes$ right_keys)

Figure 9: Basic P-tree algorithm

A.2 Optimizations

A.2.1 Communication between Left and Right Partitions

In the unoptimized P-tree algorithm above, the left and right partitions of s are both fully searched. One optimization is to use the keys returned from the left partition to guide the search for keys in the right partition. In particular, if all keys returned from each left partition work for each corresponding right partition, then the algorithm never has to branch (and thus is linear in n, m and $|K|$).

In general, the keys returned from the left partition may fail to be keys for the right partition. However, since the keys must work for the whole database, we do not need to search for all the keys of the right partition, we only need to *augment* the keys found in the left partition to be keys for the right partition. A bad case occurs where keys from the left partition are mid-size (and there are a lot of them), but there is only one large key in the right partition. In this case, a lot of effort has been spent

in the left partition needlessly. One direction for further study is thus to determine a better heuristic for choosing one of the left and right partitions (e.g. the largest one), and study the effect on the P-tree branching factor of using the keys from the chosen partition to guide the search in the other partition.

A.2.2 Sampling Partitions for Maximally Overlapping Tuples

Another key optimization can be made if we relax Manilla's restriction that the key-finding algorithm must be sub-quadratic in $|r| = m$. If we allow $O(m^2)$ algorithms, we can incorporate *sampling* into each node of the P-tree. In particular, given a database partition $s = \{t_1, ..., t_k\}$, define

$$\begin{aligned}\text{maxOverlap}(s) = \{ X \subseteq R | \exists t_i, t_j \in s \text{ such that } & t_i[X] = t_j[X] \\ & \text{and for no other pair } t_u, t_v \text{ is it the case that} \\ & t_u[Y] = t_v[Y] \text{ and } |X| < |Y| \}\end{aligned}$$

Thus, maxOverlap(s) contains the subsets of R which some pair of tuples agree on and which are maximal (no pair of tuples agree on a larger set of attributes). Note (1) maxOverlap(s) may contain more than one element, and (2) maxOverlap(s) can be computed in time $O(|s|^2)$.

The idea is then to pick an element, X, of maxOverlap(s) at each stage of the P-tree algorithm and use X to rule out attributes for branching. Note that if an attribute A is in X, then there is some pair A will not partition. Thus we don't bother branching on A. However, A may be useful later, further down in the P-tree, so we must "keep it around" for possible later branching. The solution is to allow the attribute ordering $0 < A_1 < ... < A_n$ to change at each P-tree node. Any attributes that were already branched on (earlier in the P-tree) must remain in that ordering, but "below" the current node (i.e. attributes greater than the current attribute) we are now allowed to modify the ordering to branch only on those attributes in $X \in$ maxOverlap(s) (but keep the other attributes beneath us if we need to branch on them later).

Note the following:

Lemma: If $|s| > 3$, then maxOverlap(s) contains elements of size $n/2$.

To see this, note that in a Bernoulli database we can only choose "0" or "1" for each attribute value. Two tuples can have 0 overlap between them, but as soon as a third is included it must overlap on at least 1/2 the attributes. Thus, using the maxOverlap sets to guide the branching will result in 1/2 the branching of a normal P-tree at each level.

This discussion has been regretably brief, but we look forward to investigating P-trees in more depth and detail this summer.

References

[1] Agrawal, R., T. Imielinski and A. Swami. "Mining Association rules between sets of items in large databases. *SIGMOD 93*, pp. 207-216.

[2] Agrawal, R., and R. Srikant. "Fast algorithms for mining association rules." *VLDB 94*, pp. 487-499.

[3] Albrecht M., M. Altus, B. Buchholz, A. Düsterhöft, K. Schewe, and B. Thalheim. "Die Intelligente Tool Box zum Datenbankenwurf RAD." *Datenbank-Rundbrief*, 13, F.G. 2.5 der GI, Kassel (1994).

[4] Beeri, Dowd, Fagin, Statman "On the Structure of Armstrong Relations for Functional Dependencies", *Journal of the ACM*, Vol 31., No. 1, Jan 1984, 30-46

[5] Brualdi "Introductory Combinatorics", second edition, published by North Holland, 1992

[6] Demetrovics, J. "On the Number of Candidate Keys" *Information Processing Letters 7*, 6, Oct. 1978, 266-269

[7] Demetrovics, G.O.H. Katona, D. Miklos, O. Seleznjev, and B. Thalheim. "The Average Length of Keys and Functional Dependencies in (Random) Databases." *Lecture notes in Computer Science*, vol 893, 1995.

[8] Personal communication with Ed Robertson, April 1999.

[9] Klemettinen, M., H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. "Finding interesting rules from large sets of discovered association rules." *CIKM 94*, pp. 401-408.

[10] Mannila, Heikki and Kari-Jouko Räihä. "Algorithms for Inferring Functional Dependencies from Relations." In *Data and Knowledge Engineering* 12 (1994), pp. 83-99.

[11] Mannila, Heikki. "Methods and Problems in Data Mining.", *Proceedings of International Conference on Database Theory*, January 1997, Afrati, Kolaitis (ed.), Springer-Verlag

[12] Sperner, E. Eine Satz uber Untermensen einer endlichen Menge *Math. Zeit 27*, 1928, 544-548