# Dynamic Data Mining: Exploring Large Rule Spaces by Sampling

**Sergey Brin and Lawrence Page**
Department of Computer Science
Stanford University
{sergey,page}@cs.stanford.edu

February 23, 1998

**Abstract**

A great challenge for data mining techniques is the huge space of potential rules which can be generated. If there are tens of thousands of items, then potential rules involving three items number in the trillions. Traditional data mining techniques rely on downward-closed measures such as support to prune the space of rules. However, in many applications, such pruning techniques either do not sufficiently reduce the space of rules, or they are overly restrictive.

We propose a new solution to this problem, called Dynamic Data Mining (DDM). DDM foregoes the completeness offered by traditional techniques based on downward-closed measures in favor of the ability to drill deep into the space of rules and provide the user with a better view of the structure present in a data set.

Instead of a single determinstic run, DDM runs continuously, exploring more and more of the rule space. Instead of using a downward-closed measure such as support to guide its exploration, DDM uses a user-defined measure called *weight*, which is not restricted to be downward closed. The exploration is guided by a heuristic called the *Heavy Edge Property*.

The system incorporates user feedback by allowing *weight* to be redefined dynamically. We test the system on a particularly difficult data set – the word usage in a large subset of the World Wide Web. We find that Dynamic Data Mining is an effective tool for mining such difficult data sets.

## 1 Introduction

A classical market-basket data set consists of a large database of cash register transactions, which are lists of items. The goal of mining such data is to find association rules,[1] which are statistical relationships between the purchases of different items. However, this model generalizes to many

[1]The classical example is a correlation between the purchase of beer and the purchase of diapers. It is in fact fictitious and we put it in this footnote for readers who have seen it too often.

domains other than actual cash register receipts from stores. It has been applied to collections of text documents, census data, and environmental data. Any data set that has a large database of baskets containing multiple items can fit this model. We use documents as our baskets and words in the documents as our items. The association rules that may be mined from such a data set consist of sets of items, called *itemsets*, and some indication of how they relate. However, even though this model fits many data types, data mining algorithms designed to mine supermarket data do not necessarily work well with all of them.

Traditional market basket mining algorithms exhaustively explore the space of all possible association rules using downward-closed measures such as support[2] for pruning. Recent papers dealing with supermarket or supermarket-like data report tens of thousands of items with an average of 5 to 20 items per basket in a database of millions of transactions [AS96a]. Such data sets generate hundreds of thousands of large itemsets and require several million itemsets to be tested for support. This mining task can be difficult but with recent algorithmic advances it is manageable in reasonable time [AIS93, AS94, Toi96, AS96b, CNFF96, AS96a, ZPOL97, Mue95].

Once the large itemsets are known, a pass is made over them, testing for properties like confidence, interest, and conviction [BMUT97] and rules that meet a number of constraints are produced. The resulting list of rules can then be queried, visualized, and browsed using a variety of user interfaces [KMR$^+$94].

However, when standard market basket data analysis is applied to data sets other than market baskets, producing useful output in a reasonable amount of time is very difficult. For example, consider a data set with tens of millions of items and an average of 200 items per basket. Many items have very high support (80%) and many are highly correlated. In fact there may be over $10^{34}$ itemsets meeting a reasonable support threshold (see Section 5.4). A traditional algorithm could not compute the large itemsets in the lifetime of the universe. Moreover, such a data set may be much longer or possibly an endless stream of transactions such as news feeds.

Even if these computational difficulties can be overcome, there still remains the problem that there is a very large number of interesting rules according to any statistical measure. In fact, the result size may be comparable to the original data. Providing the user a way to look through these results is a challenging data mining and visualization problem in itself.

Yet many data sets are difficult to mine because they have many frequently occurring items, complex relationships between the items, and a large number of items per basket. In this paper we experiment with word usage in documents on the World Wide Web (see Section 4.2 for details about this data set). This data set is fundamentally different from a supermarket data set. Each document has roughly 150 distinct words on average, as compared to roughly 10 items for cash register transactions. We restrict ourselves to a subset of about 24 million documents from the web. This set of documents contains over 14 million distinct words, with tens of thousands of them occurring above a reasonable support threshold. Very many sets of these words are highly correlated and occur often. In comparison, store transactions have far fewer products that are

[2]The frequency with which an itemset occurs in the data.

sold with reasonable frequency, and their interactions are far more limited. Other examples of challenging data sets are census data,[3] medical histories, and ecological data.

The main difficulty with mining data sets such as these is the size of the space of potential rules which is exponential in the total number of items. Restrictions on support and rule complexity help bring the size of this space down to possibly a small polynomial in the number of items, but for data sets with many correlated items, exhaustive search of this space is still infeasible.

## 1.1 Itemsets in a Lattice and Downward-Closed Measures

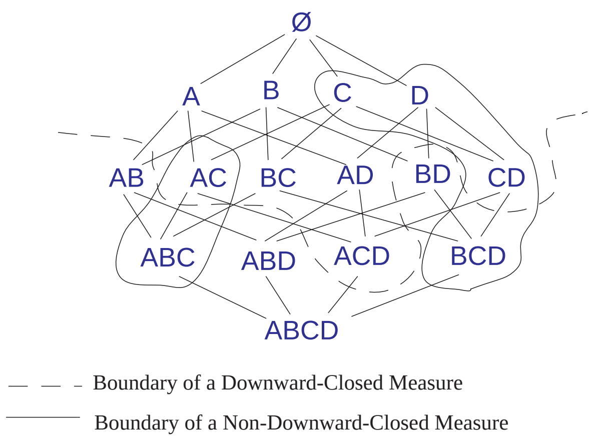
Ø
A
B
C
D
AB
AC
BC
AD
BD
CD
ABC
ABD
ACD
BCD
ABCD
Boundary of a Downward-Closed Measure
Boundary of a Non-Downward-Closed Measure

Figure 1: A Sample Lattice

A rule generated by mining market basket data involves a set of several items[4] called an itemset. The itemset has a place in the lattice of all possible sets of items ordered by the subset relation. The empty itemset is at the top and the set of all items is at the bottom. A measure over this lattice is downward closed if any subset of any itemset has a higher value than the itemset. Therefore, setting a lower bound on a downward-closed measure for itemsets we will consider draws a boundary across the itemset lattice below which we don't need to consider. If the space above that boundary is relatively small (consisting of, say, millions of itemsets), then we can exhaustively search all of it.[5] Figure 1 illustrates an itemset lattice along with the boundaries of both downward-closed and non-downward-closed measures.

The fundamental problem of mining a market basket data set is to find rules which are "interesting." There are various definitions of what interesting means in current data mining tech-

[3]Census data does not have very many distinct attributes in total. However, each basket (person) has many distinct attributes (e.g. female, drives to work, ...), and they are highly correlated, making it a difficult data set to mine (see [BMUT97]).

[4]There are some generalizations such as hierarchical data mining where classes of items can be used.

[5]We must also check all the itemsets immediately below the boundary in order to find the boundary to begin with. This is usually the biggest cost of the data mining operation.

niques. However, they are invariably either measures which are downward closed or are bounded by a downward-closed measure. These measures include support, $\chi^2$ [BMS97],[6] and collective strength [AY].

However, what users in practice find interesting may not be bounded by any downward-closed measure, at least not one that is tight enough to make exhaustive search feasible. Many current definitions of interestingness are motivated by the desire to incorporate some kind of downward closedness.

In this paper, we take a different approach. We forego the deterministic and complete algorithms offered by using downward-closed measures that tightly bound the space of potential rules. Instead, we consider an arbitrary measure of interestingness we call *weight* and rely on a randomized algorithm to explore the space of rules.

### 1.2 Overview

We present a view of itemsets as weighted hyperedges in a hypergraph. In this view, the goal of data mining is to discover the heavy hyperedges in this hypergraph. We do so by incrementally exploring parts of the hypergraph. While the weight function is arbitrary, we observe that in all cases that we have tested, the weight function has a property called the Heavy Edge Property. This property allows us to use certain heuristics to guide the exploration of the hypergraph.

To make the exploring process efficient, we introduce a system we call dynamic data mining, which samples both the hypergraph and the underlying market basket data to find heavy hyperedges quickly. Our tests of this system on the World Wide Web dataset are described in Section 5.

## 2 Itemsets as Weighted Hyperedges in a Hypergraph

**Definition 1 (Weight)** *We define* Weight *to be an arbitrary real function over itemsets. The higher the weight of an itemset, the more interesting it is. An itemset, $s$, is called* heavy *if $Weight(s) \geq \omega$, where $\omega$ is some weight threshhold ($\omega$ is not required to remain constant during the mining process).*

We do not place any restrictions upon the weight function such as downward closedness. Without knowing any more about the weight function, a simple strategy is to sample random itemsets, measure their weight, and report the heaviest itemsets. However, for many weight functions on many data sets there are heuristics that can make mining far more effective.

Consider items as nodes in a hypergraph, with itemsets as the hyperedges. The hyperedges have an associated weight, which is the weight of the corresponding itemset. Figure 2 demonstrates a simple hypergraph with weighted hyperedges. We denote simple edges (2-itemsets) with straight line segments and higher order hyperedges with surrounding curves. Heavier lines and curves denote heavier itemsets.

[6] $\chi^2$ is actually upward closed, but we are looking for the boundary, so it can still be used for pruning.

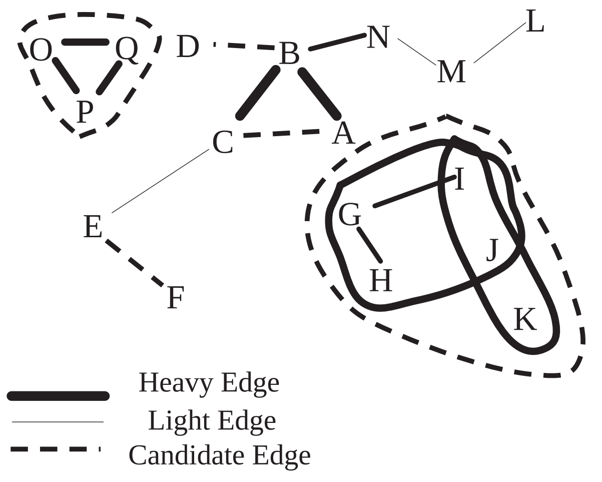

Illustration of the Heavy Edge Property

Figure 2: Example of a Weighted HyperGraph

We consider data mining an exploration of this weighted hypergraph. Our goal is to find the hyperedges with high weight. At any point in time in the mining process, we have sampled some portion of the hypergraph and know the weights of those itemsets. The marked hyperedges in the figure are ones we know the weight of. The remainder are unexplored. The hyper-edges marked with dashed lines are candidates we are considering exploring.

Given a partially explored weighted hypergraph, our goal is to select good candidates to test their weight. In classical data mining, this is a well defined process which takes advantage of downward-closedness. In our case, we rely on a heuristic called the Heavy Edge Property.

## 2.1 The Heavy Edge Property

The heavy edge property is an assumption we make about our data and weight function. We have verified that it holds in our tests and suspect that it holds in most data mining applications to some extent. However, it is an assumption and the techniques we describe in this paper depend on it.

**Property 1 (Heavy Edge Property – HEP)** *Hyper-edges that are heavy are likely to have heavy neighbors.*

First we must explain what we mean by neighboring hyper-edges. Certainly, immediate supersets of hyper-edges are their neighbors. Assuming the HEP holds, if an itemset has high Weight, we expect that if we add one item to it, it will have a better than random chance of having high weight. It will have an even better chance if *all* immediate subsets of the newly formed itemset have high weight. For example, suppose that HEP holds for the weighted hypergraph in Figure 2. Since $OP$, $PQ$, and $OQ$ all are heavy, then it is likely that $OPQ$ is heavy.

This use of the HEP is similar to the use of downward closure of support to find large itemsets. However, we go beyond this notion of neighbor. We also consider hyper-edges which share all but one of their items to be neighbors. We call these siblings. For example, since $AB$ and $BC$ are heavy, it is likely that $AC$ is heavy. More surprisingly, since $B$ already participates in a number of heavy edges, it is likely that $BD$ is heavier than a random edge. We have verified this phenomenon to hold in our data. Finally, $GHIJK$ contains very heavy interconnection and therefore is a good candidate. By comparison, $EF$ is a completely random candidate and is not likely to be heavy.

The correctness of the HEP is a function of the Weight function and of the data set. It is merely a property that interesting data sets and Weight functions are likely to have. In our tests, the HEP has held on the data sets we have tested (see Section 5). We demonstrate how to use the heavy edge property in the following section.

## 2.2 Patterns for Using the Heavy Edge Property

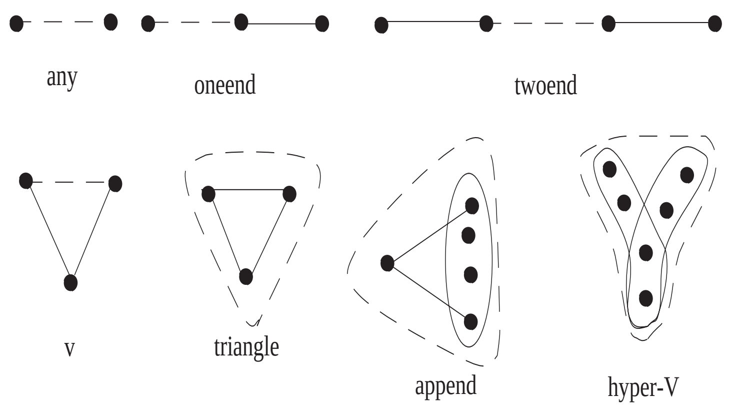

Figure 3: Patterns for Generating Candidate Itemsets

Suppose that the HEP holds for a particular data set and weight function. This still leaves the problem of exactly how to generate candidate itemsets which likely have high weight based on a set of known itemsets with weights. For this we use a number of patterns for generating candidates. The effectiveness of every particular pattern is dependent upon the data set and weight function.

Figure 3 shows the seven patterns we tested. For each pattern, the solid lines represent known heavy edges and the dashed line represents the generated candidate hyper-edge. The *any* pattern generates completely random edges and is used to seed the initial set of hyperedges before anything else is known and it adds necessary randomness throughout the mining process. It is the only pattern which does not use the Heavy Edge Property. The *oneend*, *twoend*, and *v* patterns produce edges (2-itemsets) based on hyper-edges which have already found to be heavy. *Oneend* requires one of the two items to belong to a known heavy edge while *twoend* requires that both are known.

The $v$ pattern is the "transitive" pattern and it generates the third edge of a triangle if two heavy edges already exist..

The *triangle*, *append*, and *hyper-V* create or grow higher order hyper-edges. *Triangle* creates a hyper-edge of degree 3 from three heavy degree 2 edges which are its subsets. *Append* simply adds one item to a heavy hyperedge if the item is already linked to at least two items in the hyperedge. Finally, *hyper-v* is a generalization of $v$ and it merges two heavy hyperedges that share at least two items.

One can imagine far more sophisticated ways of generating candidates. However, they would have to depend on the data and the weight function. We have taken the approach of having several patterns and testing the effectiveness of each pattern for generating heavy candidates in our particular data set. We measure this effectiveness of each pattern in Section 5.1.

## 3 Dynamic Data Mining (DDM)

The HEP provides a mechanism through which we can mine difficult data sets but it still leaves the issue of efficiency. To make an entire pass over the data for every set of candidates is prohibitively expensive. To solve this problem we introduce Dynamic Data Mining. It includes several major architectural changes to the data mining process. Instead of making mining a finite, closed-ended process which produces a well defined, complete set of itemsets, we make it a continuous process, generating continually improving sets of itemsets (see next section). Furthermore, the process takes advantage of intermediate counts to form estimates of itemsets' occurrence so there is no need to wait until the end of a pass to estimate an itemset's weight. The weight function guides the exploration of the hypergraph by taking advatnage of the Heavy Edge Property. Since the weight function can be modified on the fly at runtime, this provides a nice mechanism for user feedback with quick response.

### 3.1 Architecture

The traditional architecture for data mining consists of three major components – an engine which finds large itemsets (Itemset Counting Engine – ICE), a rule generator, and a user interface for viewing rules. In the traditional architecture, the data is the input to the ICE. The ICE sends its output to the rule generator which in turn sends its output to the rule viewer. All further interaction is between the user and the rule viewer (in some cases the rule generator may be rerun using the same set of large itemsets).

Our dynamic data mining architecture has several major differences from the traditional architecture and consists of three major components:

**DICE** Instead of an ICE, we have a Dynamic Itemset Counting Engine (DICE) which is described in detail in section 3.3. The DICE is a simple fast engine for counting itemsets. It operates on a continuous feed of data and supports three operations:

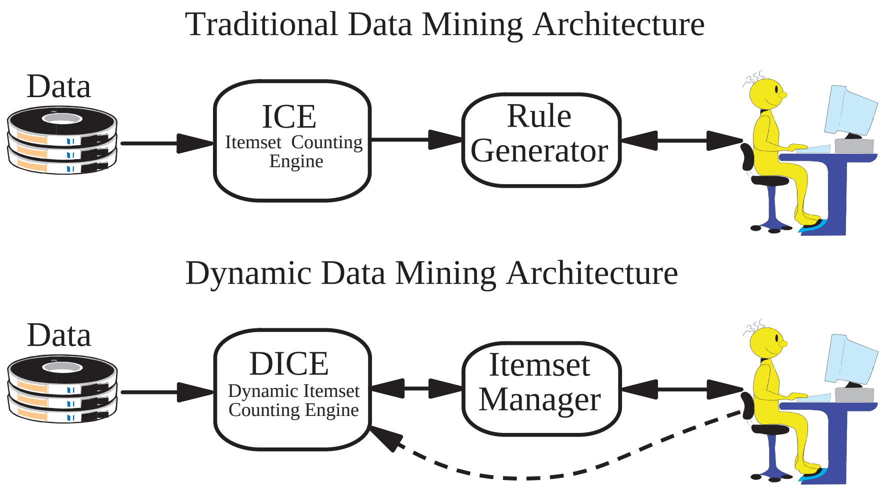
Traditional Data Mining Architecture
Data
ICE
Itemset Counting Engine
Rule Generator
Dynamic Data Mining Architecture
Data
DICE
Dynamic Itemset Counting Engine
Itemset Manager

Figure 4: Architectural Changes

- **AddItemset** – Mark where in the stream we started counting this itemset and begin counting its occurrences.
- **DeleteItemset** – Stop counting an itemset. This call is important because there is a finite amount of memory for counting itemsets. This is a significant constraint. Because of the wide nature of the data, there are incredibly many itemsets that could be counted.
- **GetItemsets** – Get a list of all the itemsets that are being counted along with their current counts.

**Itemset Manager** The DICE is managed by the Itemset Manager (IM). It periodically gets updates from the DICE using GetItemsets and decides which itemsets should be added or removed based on the counts it gets back. The IM also interacts with the user through a web-based user interface. It produces rules for the itemsets with highest weight and ships them to the user interface. It also accepts modifications to working parameters from the UI. As a result the user directly affects functioning of the Itemset Manager and hence the DICE. The results of a modification from the UI are seen at the next update (varies from one to five minutes).

**User Interface** The web-based user interface allows the user to adjust the parameters of the Itemset Manager and to give preference to certain itemsets over others. It provides quick feedback of the effects of the user's changes while the system is running.

## 3.2 The Itemset Manager

The Itemset Manager must receive itemsets with counts from the DICE and perform several operations: output interesting rules to the user, remove itemsets it decides are not useful, and add new itemsets it considers promising. The *Weight* function may change over time through user feedback.

The Itemset Manager runs the following loop continually:

1. DICE.Run($k$) *run the DICE for k transactions*
2. Itemsets = DICE.GetItemsets() *retrieve the results*
3. ComputeWeights(Itemsets) *compute the weights of all the itemsets*
4. PurgeItemsets(Itemsets) *Delete itemsets with low weight*
5. SelectNew(Itemsets) *Add new itemsets to the DICE.*
6. repeat

The functions which are critical to the behavior of the Itemset Manager are the SelectNew function, and the PurgeItemsets function. We describe these functions in the following sections.

### 3.2.1 The SelectNew Function: Using the Heavy Edge Property

The SelectNew function generates candidate hyperedges for testing. It makes use of the Heavy Edge Property via the patterns in Section 2.2. Currently, it randomly generates an equal number of instances from each pattern though sometimes a pattern may generate fewer instances because it is not possible to apply it (for example, we can only use the *any* pattern at the very start). A future enhancement may allow the Itemset Manager to learn which patterns are effective for a particular data set and to favor them.

### 3.2.2 The PurgeItemsets Function

Since the DICE is constrained by memory in terms of the number of itemsets it can count, it is necessary to remove itemsets to make room for additions. Therefore, the Itemset Manager deletes certain itemsets at each point in its main loop. The algorithm for this is very simple – all itemsets which are not heavy are deleted. The weight threshhold, $\omega$ varies so that the top $n$ heaviest edges are considered heavy.[7] Note that this is a nontrivial decision. An itemset which is not heavy may become heavy as its counts are updated or as the Weight function changes. Therefore, a reasonable alternative strategy may be to give near-heavy edges a probation period to become heavy. We experimented with this probation strategy. However, we found it did not work as well as the simple strategy because by testing a near-heavy edge for a longer time, we lose opportunities to test new edges. Furthermore, as we find heavier and heavier edges, the threshold to be in the top $n$ heavy

[7]Note: currently there are actually separate threshholds for 2-itemsets and higher order itemsets so that the top $n$ 2-itemsets and the top $m$ higher order itemsets are heavy.

edges moves up. Therefore, it is unlikely that a borderline edge would be heavy for long and make a significant contribution to the mining process.

An itemset are also pruned away if the system has discovered a superset of the itemset with higher weight.

## 3.3 DICE – the Dynamic Itemset Counting Engine

In [BMUT97], Brin, Tsur, Ullman, and Motwani present an algorithm for counting large itemsets called DIC – Dynamic Itemset Counting. A traditional large itemset counting algorithm proceeds level-wise; it makes one pass over the data for each size of itemsets it is counting. After it counts all the 1-itemsets, it knows which 2-itemsets to count and counts those on the next pass. After it counts the 2-itemsets it knows which 3-itemsets to count and counts those on the next pass and so on.

Instead of going through the data one pass at a time, the DIC algorithm goes through the data one chunk at a time. For the first chunk, it counts only the 1-itemsets. Then it estimates which 2-itemsets to count and starts counting those in the second chunk. Then it estimates which 3-itemsets to count and starts counting those in the third chunk and so on. Once it gets through the first pass, it stops counting the 1-itemsets and it rewinds to the start of the data. Then it goes through the first chunk again and finishes counting the two-itemsets. Then it finishes counting the 3-itemsets, and so on[8].

The DICE system takes advantage of the DIC algorithm's ability to start and stop counting any itemset at any time. Here are the key aspects of the DICE:

- The DICE keeps track of its current position in the data, denoted by *curbasketnum*. If the DICE, wraps around the data, the curbasketnum is *not* reset; it simply keeps getting incremented.
- For every itemset, it is counting, the DICE keeps track of the current *count* of that itemset and the value of curbasketnum when it started counting it, denoted by *since*. If *curbasketnum* - *since* ever reaches the total number of baskets, we mark the itemset done and stop counting it. The support of an itemset may be calculated as *count/(curbasketnum-since)*.
- For efficient counting, the itemsets are kept in a hash tree [AS94].
- The AddItemset function adds a new itemset to the hash tree and setting its count to 0 and its since value to curbasketnum.
- The DeleteItemset function removes the itemset from the hash tree, making sure to clear out all of its memory so new itemsets can be added.

[8]This is somewhat of a simplification of the DIC algorithm. Interested readers should take a look at [BMUT97].

- We assume that the Itemset Manager does not know about all the items that may be in the data ahead of time. Therefore, the DICE adds counters for any new items it may run across. An important point is that the since value must be set to 0 instead of curbasketnum.

These properties allow the DICE to provide a fast and simple interface to control and access itemset counts. The Itemset Manager can take advantage this efficient mechanism to provide quick response to user feedback and to the counts it is receiving from the DICE.

## 3.4 The Weight Function

For our tests, we tried several different weight functions. Let $P(S)$ be the support of an itemset $S$. Then we tried the following weight functions.

$$\begin{aligned}
\text{LogInterest}(S) &= \frac{\log(\frac{P(S)}{\prod_{i \in S} P(\{i\})})}{|S|} \\
\text{SetInterest}(S) &= \max_{i \in S} \frac{P(S)}{P(\{i\})P(S - \{i\})} \\
\text{SetConviction}(S) &= \max_{i \in S} \frac{P(S - \{i\})(1 - P(\{i\}))}{P(S - \{i\}) - P(S)}
\end{aligned}$$

The Set functions require that we know the support of all immediate subsets of an itemset before we can calculate the weight. Otherwise we can only find a lower bound based on the subsets we do know. For this reason, and because the LogInterest is well normalized based on the number of items, we chose to experiment the most with LogInterest. In essence, the LogInterest function is the number of occurrences of an itemset divided by the expectation of that number under the independence assumption. This ratio is then converted to a logscale and normalized based on the number of items.

We also applied a support threshhold to our itemsets. While this may seem to contradict the main point of this paper, the threshhold was only to make sure that the itemsets were indeed interesting. We set this threshhold at 0.0001. At this threshhold, there were at least $10^{34}$ itemsets which satisfied it (see Section 5.4).

## 4 Implementation

We implemented the Dynamic Data Mining system in several components. The Dynamic Itemset Counting Engine (DICE) is implemented in C++, since it is performance critical. The Itemset Manager is considerably more complex but less performance critical so it is implemented in Python. The DICE is wrapped into a python module using SWIG [Bea96] so the Itemset Manager can access its functions and variables.

### 4.1 The User Interface

For the user interface, the Itemset Manager runs as an HTTP server. This way many users can interact with the Itemset Manager at the same time, making collaborative data mining possible. When the server is accessed it produces a web page which consists of three parts. On top are various statistics and parameter settings. Below that is a text input box which contains metarules for guiding the mining process. The user can adjust parameter settings and item weights via a simple language in the text input box.

### 4.2 The Web Repository

The web pages used for mining are stored compressed in a repository which spans 27 disks on three computers. In our system, a separate machine reads the repository in a randomized order, uncompresses and parses the web pages, converts words into unique four-byte integers and feeds them as market baskets to the data mining system. To date we have collected 24 million web pages. They contain over 14 million distinct words with an average of 150 distinct words per page.

## 5 Experiments

In order to test our Dynamic Data Mining system, we ran a number of experiments. We tested the heavy edge property and its usefulness in mining. We also ran a number of tests dynamic data mining. These experiments are described in the following sections.

### 5.1 Testing the Heavy Edge Property

We tested the effectiveness of our patterns in generating candidates that are heavy. We did this by running the DDM system for a short while (40 iterations of the Itemset Manager) and seeing how each pattern had fared in producing itemsets with different weights. Recall that the Itemset Manager produces candidates from each of the patterns equally when possible (the more complex patterns have more complex preconditions so it is not always possible to generate them). The resulting distributions are plotted in Figure 5. These include two types of itemsets – itemsets that were generated a number of iterations ago and have survived numerous weight checks and a few itemsets that have recently been generated and are as of yet untested. Itemsets which don't meet the support requirement and itemsets which did not meet the weight threshhold in the previous iteration are not included here.

Notice that while the heaviest itemsets generated by the *any* pattern are close to the heaviest itemsets, the *any* pattern generated far fewer heavy itemsets overall. The reason we use the *any* pattern is because it provides a starting point for the other patterns initially and it continues to provide itemset diversity as the mining process continues.

The other patterns which produce 2-itemsets behave as one may expect with the *v* pattern producing the most heavy edges followed by *twoend* and then *oneend*. The patterns that produced

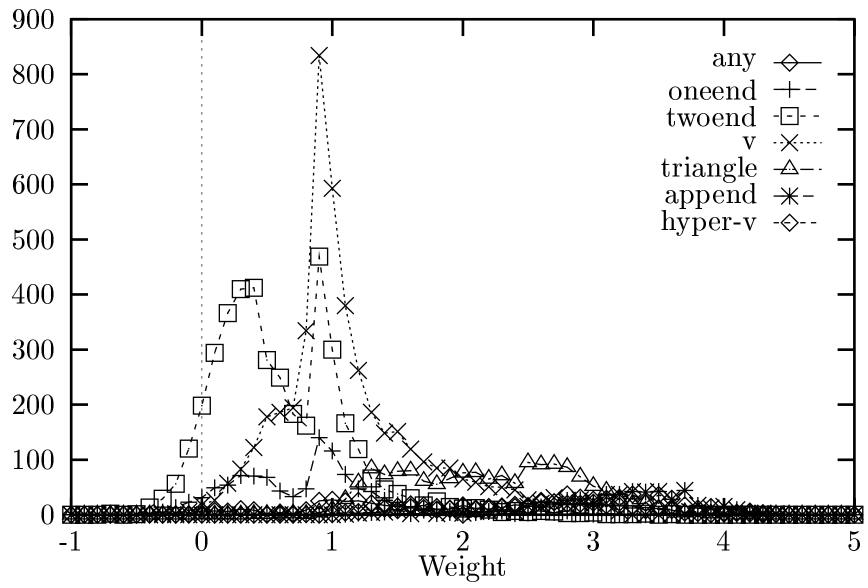

Figure 5: Effectiveness of Different Patterns at Generating High Weight Itemsets

higher order itemsets produced fewer heavy itemsets but they consistently produced the heaviest of edges.

All these results indicate that there is a strong relationship between the weight of an itemset and the weights of its neighbors. This is what one would expect and this verifies that the Heavy Edge Property holds on this data set.

## 5.2 Usefulness of the Heavy Edge Property

In addition to showing that the heavy edge property holds, it is necessary to show that it actually helped the data mining process. There is a major tradeoff involved in using the HEP for selecting new itemsets to mine. Using the HEP too strongly and soon can prevent the system from ever "seeing" a very interesting portion of the hypergraph. This happens because the search is narrowed by using the HEP property. Therefore, some degree of randomness must be incorporated into the mining process. This is the reason for including the *any* pattern.

To test the usefulness of the Heavy Edge Property we attempt to mine using only the *any* pattern (purely randomly) versus using all the patterns and we produce a plot of the weights 5000 heaviest itemsets at every iteration. These are shown in Figures 6 and 7. Notice that the purely random technique finds some rather heavy edges early on. However, its progress quickly diminishes in that increasingly heavier edges are not found much past 30 iterations. Furthermore, the random technique remains dense at the bottom and very light on top all the way through the mining process. By comparison, the HEP based mining starts a little slow but it continues to grow the weight of the itemsets it finds until it is well above the random technique.

Furthermore note that the apparently small difference between the two plots are actually quite

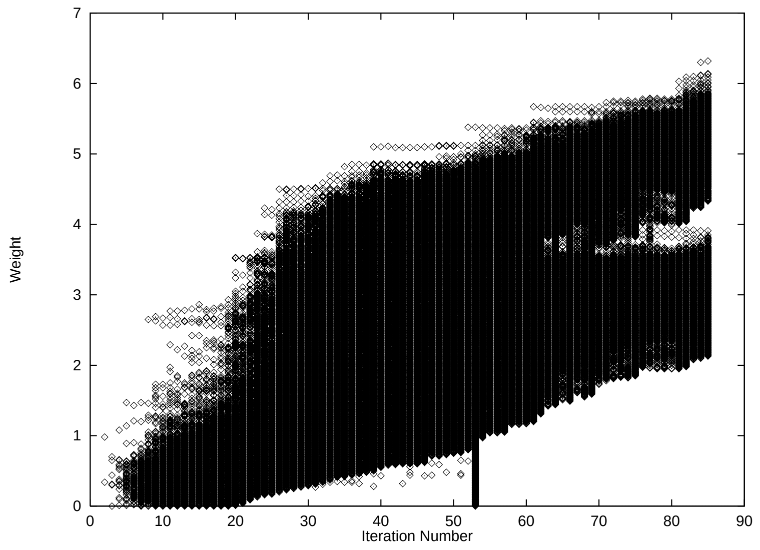

Figure 6: Scatter Plot of Top Weights Using HEP

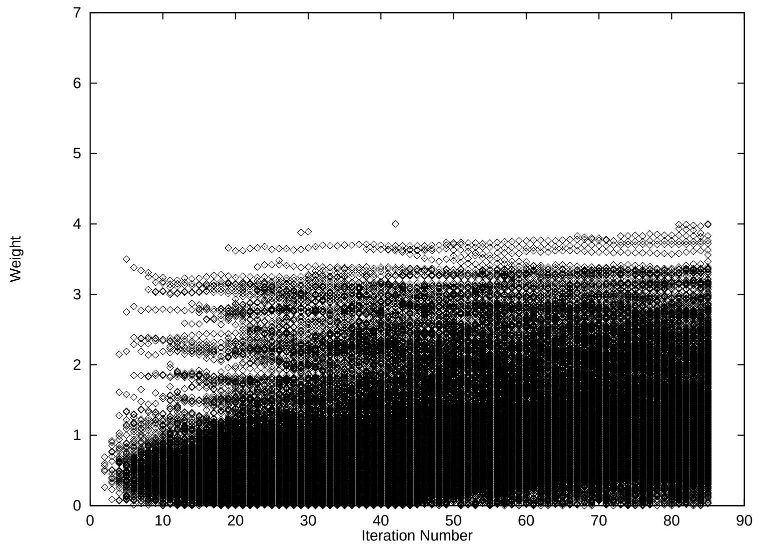

Figure 7: Scatter Plot of Top Weights Using HEP

substantial due to the log factor in the weight function.

## 5.3 Mining Results

Of course the ultimate test of a data mining technique is the quality of the results. In this section, we present results from various runs of the mining algorithm at different times.

| weight | itemset | support |
|---|---|---|
| 2.64 | photoshop, powermac | 0.00118 |
| 2.58 | rb, rebound | 0.00036 |
| 2.46 | emirates, pakistan | 0.00045 |
| 2.36 | pied, dans | 0.00022 |
| 2.17 | powermac, monitors | 0.00077 |
| 2.08 | media, powermac, technology, star | 0.00077 |
| 1.82 | sponsor, unlimited, try | 0.00140 |
| 1.63 | mike, robert, michael | 0.00340 |
| 1.61 | robert, michael, nancy | 0.00107 |
| 1.58 | media, powermac, technology | 0.00080 |
| 1.56 | media, mb, technology, star | 0.00077 |
| 1.38 | buy, america, product, try | 0.00120 |
| 1.38 | lt, mt, gt | 0.00087 |
| ... | ... | ... |

Figure 8: Itemsets Generated Early in Mining Process – 85000 Baskets Processed

We present the heaviest itemsets of an early part of a run in Figure 8. There are a variety of somewhat related itemsets. We see several itemsets involving media and powermac and a couple of itemsets with various names. Later in the mining process, the system begins to find stronger correlations and larger itemsets as we see in Figure 9. There are a number of itemsets of country names and an itemset of computer jargon. The remainder of the output is mostly about those two subjects. In Figure 10, we show some interesting, amusing, or characteristic results we have picket out from various runs.

Other itemsets that DDM has discovered have included state names, sex-related terms, and sports teams. Some of these discoveries have been rather mundane but some have been quite interesting. We have discovered a number of spam lists – lists of words that web sites use to artificially increase the number of hits they get from search engine users. One particularly interesting example was a cluster of sex-related terms (which cannot be printed here). What was unusual about it was that several of the words did not appear to be sex-related at all (for example, "jmis"). Some searching revealed the explanation. A certain Mr. Lick had received a number of hits on his home page from sex-related searches. For fun he actually lists all searches that have led to his home page on the page itself, leading to more hits from sex searches and thus creating a positive feedback loop. The page now contains a list of hundreds of distinct sex related terms. This list turns out to be quite useful for sex site administrators who have simply copied it onto their own web pages

| weight | itemset | support |
|---|---|---|
| 5.72 | maldives, zambia, fiji, guadeloupe, estonia, rwanda | 0.00033 |
| 5.67 | maldives, uruguay, fiji, guadeloupe, estonia, rwanda | 0.00038 |
| 5.63 | maldives, uruguay, namibia, fiji, estonia, rwanda | 0.00040 |
| 5.54 | maldives, fiji, isls, estonia, rwanda | 0.00020 |
| 5.47 | maldives, denmark, caledonia, ivory, djibouti, rwanda | 0.00031 |
| 5.46 | maldives, namibia, fiji, denmark, estonia, rwanda | 0.00039 |
| 5.45 | guinea, uruguay, zambia, fiji, estonia, rwanda | 0.00037 |
| 5.43 | officejet, photosmart, simm, proliant, sportster, nec, portable, cyberstores | 0.00119 |
| ... | ... | ... |
| 4.10 | football, cnn, al, fantasy, league, cfl, sportsticker, live, text, crowd, scores, baseball, left, nl, turner | 0.00048 |
| 4.10 | guadeloupe, denmark, estonia, italy | 0.00040 |
| 4.10 | stb, blaster, gateway, nec, speakers | 0.00075 |

Figure 9: Itemsets Generated Later in Mining Process – 330000 Baskets Processed

| weight | itemset | support |
|---|---|---|
| 6.03 | barbuda, morocco, svalbard, timor, barbados, angola, futuna, slovenia, bolivia, dominican, pitcairn, kiribati, mauritania | 0.00020 |
| 6.63 | appname, image, toc2on, imgon, img, toc3off | 0.00040 |
| 7.28 | dissatisfactions, suffer, come, fear, heterosexual, impotence | 0.00032 |

Figure 10: Miscellaneous Interesting Results From Various Runs

9

and even named URL's after the words in that list. In the process, however, a few normal searches which have led people to Mr. Lick's home page have propagated to many sex sites. It is these words which the mining process linked to sex.

## 5.4 Itemsets with Many Items

Some other interesting results of dynamic data mining have been the sizes of the itemsets generated. The largest itemset generated thus far was a set of 115 computer jargon terms: *deskstar, xpert, microsoft, pin, palm, wordperfect, buy, monitor, viewsonic, apple, helps, needs, backup, mmx, thinkpad, mystique, robotics, access, stylus, wide, w, scans, cable, deskpro, screen, millennium, blaster, games, gold, e6, tool, photosmart, processor, motherboard, dell, card, router, miro, brother, plus, wonder, palmpilot, prices, mb, satellite, systems, lexmark, pc, stock, ethernet, color, word, quantum, tx, omnibook, major, utilities, corel, shareware, modems, cannon, gateway, directly, simm, iii, proliant, flatbed, books, phone, desktop, intel, power, sportster, average, nec, updates, memory, netscape, ink, supermicro, weight, pci, stingray, plextor, dimm, bjc, hercules, portable, download, art, pda, different, computing, huge, serial, speakers, optra, diamond, adobe, battery, ibm, kodak,* *fujitsu, tower, audio, floppy, star, paperport, nt, adaptec, cyberstores, visual, controller, presario, ultimate.* There were other itemsets approaching this size. The discovery of such a large itemset is important for a number of reasons. First, to the best of our knowledge, this itemset contains far more items than any rules produced by a conventional data mining technique. If we were mining for it using a traditional levelwise mining technique, it would require us to test roughly $2^{115}$ or more than $10^{34}$ itemsets in order to get to it. Furthermore, it had plenty of support (0.00048 – more than four times our threshhold) and there are likely many other such itemsets so it is likely that a traditional support based algorithm would produce more than $10^{34}$ itemsets which exceed the support threshold.

It is arguable that such itemsets are not very interesting. However, in this case, the itemset represents a legitimate set of words which occur together very often for a good reason. It is quite likely that the user would want to know about such itemsets. Finally, this itemset concisely summarizes the relationships within all of its subsets (though those may involve more intricacies).

It should be noted that in our tests of DDM, it frequently converged to the same set of clusters. However, this was not always the case. There are benefits to both having a reliable system that generally acts the same way or a system that produces new results every time.

## 5.5 User Feedback

We ran a few simple tests applying user feedback. In one test, we started by artificially increasing the weight of the French word oui and artificially decreasing the weight of several computer terms and country names which frequently clutter the input.

| weight | itemset | support |
|---|---|---|
| 3.67 | egrave,sur,et,la,dans,tout | 0.00100 |
| 3.61 | sur,et,leurs,dans | 0.00102 |
| 3.59 | sur,et,un,dans,tout | 0.00110 |
| 3.58 | egrave,sur,la,dans,tout | 0.00085 |
| 3.48 | sur,et,dans,tout | 0.00124 |
| 3.43 | sur,et,la,dans,tout | 0.00127 |
| 3.40 | egrave,et,la,dans,syst | 0.00060 |
| 3.40 | egrave,et,dans,tout | 0.00088 |
| 3.38 | egrave,et,la,dans,tout | 0.00093 |
| 3.36 | egrave,sur,et,la,dans | 0.00260 |
| 3.35 | sur,de,et,dans,tout | 0.00105 |
| 3.35 | sur,dans,tout | 0.00134 |
| 3.32 | egrave,sur,de,et,dans | 0.00260 |
| 3.31 | sur,la,dans,tout | 0.00133 |
| 3.30 | sur,veloppement,dans | 0.00060 |
| 3.29 | egrave,sur,de,et,la,tout | 0.00110 |
| 3.29 | egrave,la,dans,syst | 0.00067 |

Figure 11: Taken From Top 60 Itemsets After User Increased the Weight of the Word "Oui"

After roughly 40 iterations of the Itemset Manager, a number of French itemsets were generated. The itemsets in Figure 11 all appeared within the top sixty itemsets.

### 5.6 Performance

We ran our Dynamic Data Mining system on a dual processor 300MHz Pentium II with 512MB of RAM. The web pages were uncompressed from a repository, parsed from HTML, and converted into market baskets by a Sun Ultra II and were fed to the Dynamic Data Mining system via TCP.

There are two major components to the performance of dynamic data mining. These components are itemset counting in the DICE and the computation done by the Itemset Manager. The speed of counting itemsets in the DICE is determined largely by the number of itemsets it is counting, their frequency of occurrence in the data set, and the average basket width. For our datasets, the DICE runs at roughly 100 baskets (documents) per second though it depends heavily on how many itemsets are being counted. The total number of itemsets is kept under 200,000.

The overhead of the Itemset Manager depends on several parameters. The most important one is the interval between updates from the DICE. For long runs we set this at 10,000 baskets (documents) amounting to 5 to 6 minutes between updates. At this interval size the Itemset Manager had an overhead of 30 to 40 percent. Given that the Itemset Manager is an interpreted program in the Python language, we were quite happy with its performance. In order to keep this overhead small, the DICE does some preprocessing by filtering out itemsets with extremely low counts (counts below 10).

## 6 Related Work

The related work to dynamic data mining falls into three categories: user interfaces for data mining, the use of sampling in data mining, and heuristic search in high dimensional spaces.

There is a considerable amount of work on user interfaces for data mining. Most work has focussed on postprocessing data mining results so the user can query them and be presented different views [KMR$^+$94]. Feldman et al. demonstrate techniques to visualize text mining results in [FKZ97]. Agrawal et al. allow the user to specify interests in advance of the mining process [SVA]. There has also been work allowing the user to specify beliefs and determining how the data deviates from these beliefs [ST96]. However, there has been very little work on interacting with the user throughout the mining process. Wrobel et al. suggest an architecture for user interaction during data mining in the KESO system [WWV$^+$96].

Sampling market basket databases has been explored by [Toi96]. Our use of sampling to dynamically better estimate itemset counts was presented in [BMUT97]. However, the technique of sampling the space of itemsets has not been presented before. Also, there has been work studying itemset lattices as hypergraphs but not in the context of heuristics such as the heavy edge property.

There has also been considerable work regarding the mining of text data. However, our system is designed to be a general market basket mining system and we avoided using text-specific optimizations.

## 7 Future Work

Experimenting with dynamic data mining has brought up a number of questions and future research directions. First, our current system is by no means a complete data mining system. It is a research tool to test a number of ideas. As it matures it will become more complete and will allow us to study the ideas we mention here.

A current problem is that one or several major clusters of itemsets can dominate the heavy itemsets. We need to find new ways to ensure diversity heavy itemsets. This includes the possibility of identifying clusters and limiting their size.

Dynamic Data Mining gives us the opportunity to experiment with complex weighting functions. In particular we would like to incorporate measures of causality into the weight function. Finally, we would like to extend the techniques of dynamic data mining to other kinds of data such as time series data.

## 8 Conclusions

We have found that dynamic data mining adds several significant enhancements to the process of data mining. Most importantly, it makes it possible to mine data sets which would be impossible otherwise. The key to this process is the use of sampling of both the data and the space of itemsets, the latter being the more important of the two. By forfeiting completeness we are able to drill down much farther into the rule space, generating rules with as many as 115 items.

The key to sampling the space of itemsets is the Heavy Edge Property (HEP), which hypothesizes that interesting itemsets are likely to be near other interesting itemsets. This makes it much easier to find itemsets with high weight. However, it has the side-effect of finding clusters of items which is sometimes desirable and sometimes not. Furthermore, it is important to mix use of the HEP with randomness (for example, via the *any* pattern) to prevent the mining process from settling in local optima. Of course, in order to be useful the HEP must hold on the data set being mined. We have found that the HEP holds very strongly on word usage in web pages and believe it to be true on a wide variety of data sets. We intend to test it on other data sets in the near future.

Furthermore, dynamic data mining allows for quick user feedback. Instead of waiting for a long data mining job to run, the user can get results as they happen from the system and adjust the mining based on those results. When mining a space with many complex and highly correlated interactions, this feedback is very important. Also, the user interface which is implemented as a web server, allows for collaborative data mining. Any user at any location may adjust parameters or weightings of items as the system is running.

Dynamic data mining has several other nice properties. First, it can work on continuous data streams. This means that it is possible to mine data from live data sources or to mine data which one cannot afford to store on disk. For example, it is possible to mine the entire World Wide Web without storing a single web page (Note that it is still necessary to store some large data structures for crawling such as url queues). Another use of this property is mining where quick reaction to new relationships is important such as data feeds from stock and currency exchanges.

In summary, dynamic data mining is a very effective tool for data exploration, particularly when it is not feasible to completely explore the space of rules. It provides results quickly independent of the size of the data set and refines them as it progresses.

## References

[AIS93] R. Agrawal, T. Imilienski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 207–216, May 1993.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference, Santiago, Chile*, 1994.

[AS96a] R. Agrawal and J. Shafer. Parallel Mining of Association Rules: Design Implementation and Experience. Technical report, IBM Research Division, Almaden, California, February 1996.

[AS96b] R. Agrawal and J. C. Shafer. Parallel mining of association rules. *Ieee Trans. On Knowledge And Data Engineering*, 8:962–969, 1996.

[AY] Charu Aggarwal and Phillip Yu. A new framework for itemset generation. In *PODS 1998, to appear.*

[Bea96] David M. Beazley. SWIG: An easy to use tool for integrating scripting languages with C and C++. In USENIX Association, editor, *4th Annual Tcl/Tk Workshop '96, July 10–13, 1996. Monterey, CA*, pages 129–139, Berkeley, CA, USA, July 1996. USENIX.

[BMS97] Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond market baskets: Generalizing association rules to correlations. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(2):265, 1997.

[BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(2):255, 1997.

[CNFF96] D. W. Cheung, V. T. Ng, A. W. Fu, and Y. J. Fu. Efficient mining of association rules in distributed databases. *Ieee Trans. On Knowledge And Data Engineering*, 8:911–922, December 1996.

[FKZ97] Ronen Feldman, Willi Klösgen, and Amir Zilberstein. Visualization techniques to explore data mining results for document collections. In David Heckerman, Heikki

Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, page 16. AAAI Press, 1997.

[KMR$^+$94] Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen, and A. Inkeri Verkamo. Finding interesting rules from large sets of discovered association rules. In Nabil R. Adam, Bharat K. Bhargava, and Yelena Yesha, editors, *Third International Conference on Information and Knowledge Management (CIKM'94)*, pages 401–407. ACM Press, November 1994.

[Mue95] Andreas Mueller. Fast sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, Dept. of Computer Science, Univ. of Maryland, College Park, MD, August 1995.

[ST96] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *Ieee Trans. On Knowledge And Data Engineering*, 8:970–974, 1996.

[SVA] Ramakrishnan Srikant, Quoc Vu, and Rakesh Agrawal. Mining association rules with item constraints. page 67.

[Toi96] H. Toivonen. Sampling large databases for association rules. *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, 1996.

[WWV$^+$96] Stefan Wrobel, Dietrich Wettschereck, A. Inkeri Verkamo, Arno Siebes, Heikki Mannila, Fred Kwakkel, and Willi Klösgen. User interactivity in very large scale data mining. In W. Dilger, M. Schlosser, J. Zeidler, and A. Ittner, editors, *Proc. FGML-96 (Annual Meeting of the GI Special Interest Group Machine Learning)*, pages 125–130, 09111 Chemnitz, August 1996. TU Chemnitz-Zwickau. Computer Science Technical Report No. CSR-96-06.

[ZPOL97] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, page 283. AAAI Press, 1997.