PROBABILISTIC DIAGNOSIS OF HOT SPOTS

Kenneth Salem
Daniel Barbara
Richard J. Lipton

CS-TR-328-91

June 1991

Probabilistic Diagnosis of Hot Spots [1]

Kenneth Salem

Computer Science Department
University of Maryland
College Park
Maryland 20742–3255

Daniel Barbará
Richard J. Lipton
Computer Science Department
Princeton University
35 Olden Street
Princeton, NJ 08544–2087

Abstract

Commonly, a few objects in a database account for a large share of all database accesses. These objects are called hot spots. The ability to determine which objects are hot spots opens the door to a variety of performance improvements. Data reorganization, migration, and replication techniques can take advantage of knowledge of hot spots to improve performance at low cost. In this paper we present some techniques that can be used to identify those objects in the database that account for more than a specified percentage of database accesses. Identification is accomplished by analyzing a string of database references and collecting statistics. Depending on the length of the reference string and the amount of space available for the analysis, each technique will have a non–zero probability of false diagnosis, i.e., mistaking "cold" items for hot spots and vice versa. We compare the techniques analytically and show the tradeoffs among time, space and the probability of false diagnoses.

---

[1] This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and by the Office of Naval Research under Contracts Nos. N00014–85–C–0456 and N00014–85–K–0465, and by the National Science Foundation under Cooperative Agreement Nos. DCR–8420948 and CCR–8908898. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

1. Introduction

Access to items in a database is generally not random. Most commonly, a small set of data items absorbs a disproportionate share of all data accesses. These items are commonly called hot spots.

The ability to identify hot spots opens the door to a variety of performance improvements. Caching, data migration, prefetching, and replication are examples of performance improvement techniques that can take advantage of hot spot information. For instance, replicating data is a well-known technique for reducing access times in a distributed system. The entire database could be replicated, but replication is costly. Clearly, most of the benefits of replication could be had for a fraction of the cost if only hot spots were replicated.

As another example [Sa91], consider prefetching and other types of anticipatory data movement. If it is known that two files (or other data), F_1 and F_2, tend to be used together, then it may be beneficial to move F_2 near F_1 when F_1 is requested and used. How can we determine which files tend to be used together? Determining these relationships can be thought of as looking for hot spots in the set of all *pairs* of files. Thus, when F_2 is used shortly after F_1, we can treat this as a "reference" to item (F_1,F_2) in the "database" of pairs of files. After some time, the set of file pairs can be examined (perhaps using one of the techniques presented in this paper) to determine which pairs of files are related.

When the subject of hot spots comes up, buffer management techniques such as least-recently-used (LRU) and working sets come to mind immediately. These techniques are well-known, and many have been in wide use for years. For this reason, it is important to distinguish between them and the techniques presented in this paper. To facilitate comparison, we have included an LRU-like algorithm (Algorithm D) among those studied in this paper.

Most buffer management techniques select data based on recency of use. In contrast, the algorithms presented here make selections based on frequency of use. Although these concepts are related, they are clearly not the same. For example, consider a database such that 80% of all references are made to 20% of the data, and suppose that a buffer with a capacity of 100 blocks of data is used. Under reasonable conditions, we would expect that roughly 20 of the 100 blocks in the cache would be occupied by data from the "cool" (less frequently referenced) potion of the database. Such data are poor choices if frequency of use is the selection criterion.

A second difference is that the algorithms presented in this paper attempt to detect *absolute* hotspots. For example, the goal may be to determine which blocks (if any) receive at least 25%

of all requests. The absolute threshold, in this case 25%, is a parameter of the algorithm. Techniques such as LRU do not use such a parameter. For example, suppose that there are 1000 blocks of data, each equally likely to be requested. The LRU buffer manager described earlier will select one hundred of these blocks (and place them in the buffer). However, using a 25% threshold, the techniques presented here should report that no hot spot exists.

These comparisons are not to suggest that the hotspot algorithms presented here are "better" that LRU or other buffer management techniques. Clearly, a buffer manager should keep its buffer filled, even if there are no apparent hotspots. Furthermore, frequency of reference alone is not always the best criterion for data selection. (The existence of temporal locality in data references has been well-documented.) Often, however, algorithms like those presented in this paper will be well-suited to a particular task. One situation in which this is true is when hot spots can be taken advantage of only at a cost. For example, replicating a frequently referenced file consumes storage space, processing time, and bandwidth. The file should only be replicated if the potential gains more than offset these costs. Thus, only files that are referenced frequently enough should be replicated. Because they detect data whose "temperature" is above a given threshold, the techniques presented here would be useful for detecting files that are hot enough so that replication will be beneficial.

A number of studies have pointed out the existence of hot spots of various durations in file systems and databases [Ou85], [St88], [Wi89]. Some other attempts to deal with hot spots to increase concurrency and improve the performance of database systems can be found in [Ga85], [GaKi85], [On86] and [Re82]. In [Ba87], a stochastic model of data access is developed under the basic assumption that there exists a fraction of the items that receive most of the accesses, and that the distribution of accesses both within and outside of this set of items is skewed.

In this paper we present several techniques to identify, or diagnose, hot spots in a database. All of them are probabilistic in the sense that they will classify the items as hot or cold and exhibit a non-zero probability of false diagnoses. We analyze each technique to identify the tradeoffs of time and space involved in maintaining a low probability of false diagnosis. In Section 2 we present each of the techniques. In Section 3 presents analyses of the techniques to determine how likely they are to diagnose without error. In Sections 4 and 5 we compare the techniques, including a numerical comparison based on analyses.

2. The Problem and Some Solutions

In this section, we formulate the hot spot diagnosis problem mathematically and describe several algorithms to solve it. We assume that the *database* consists of a set of N items. Requests for database objects (references) are modeled as an independent random process. We assume that these references are arbitrarily but identically distributed, as will as independent. The probability that a reference will be to item i will be denoted p_i. Often, we call this probability the *weight* of the item.

Given a threshold $0 < T < 1$, and a positive constant ϵ (where $0 < T-\epsilon$ and $T+\epsilon < 1$), we classify each item into one of three sets:

$$C = \{i \text{ such that } p_i < T - \epsilon\}$$

is the set of "cold" items, i.e., those which should be appear with a frequency less than the threshold minus a small ϵ. Conversely, the set of "hot" items is denoted by

$$H = \{i \text{ such that } p_i \geq T + \epsilon\}$$

A third set of items, called U, is composed of those items with weight between $T - \epsilon$ and $T + \epsilon$.

The hot spot problem can be stated as follows: given a reference string of length S, generated as described above, the threshold T, and the constant ϵ, report a subset of the database that includes the set H but does not intersect the set C. In other words, a hot spot algorithm should report all of the hot items and none of the cold ones. Items "near" the threshold may be reported or not.

We have investigated four classes of algorithms for solving this problem. All of the solutions make use of *counters*, i.e., variables initialized to zero which can be incremented. In each of the algorithms, one or more database items are assigned to one or more counters. A counter is used to keep track of the number of times that items assigned to it appear (are referenced) in a certain portion of the reference string. We will often refer to the weight of a counter, meaning the sum of the weights of the items assigned to it.

The algorithms have a common component, which we term "processing the reference string". A string of size s is processed as follows. For each reference in the string, counter(s) to which the referenced item has been assigned are incremented. If a referenced item is not assigned to any counter, that reference is ignored.

After processing, an algorithm's counters can be classified as either "hot" or "cold". Let k be the value of the counter after a reference string of length s has been processed. The counter is

considered to be hot if

$$\frac{k}{\varepsilon} \geq T$$

otherwise it is cold. Each of the algorithms, except the last, use this notion of hot and cold counters.

In the following sections, we describe the four algorithms we have considered.

2.1. Algorithm A

This simple algorithm iteratively tests M items at a time until all N items in the database have been tested. During each iteration, M previously unassigned database items are assigned, one to each of M counters. A portion of the reference string is then processed. If any of the counters are hot after processing, the items assigned to those counters are included in the solution. The counters are then reset (cleared) and the process is repeated using the next portion of the reference string, with new items assigned to the counters.

Iteration continues until each database item has had the opportunity to be assigned to a counter and tested. Since M items are tested during each iteration, M/N iterations are required. If the reference string S references long, SM/N references will be used during each iteration.

2.2. Algorithm B

Algorithm B makes use of L groups of counters, with M counters in each group. For each group of counters, the database is randomly partioned into M sets of (approximately) equal size. The items in each set are all assigned to one of the M counters in the group. The random partitioning of database items is done independently for each group of counters.

After the assignments have been made, the reference string is processed. Each reference in the string will cause L counters to be incremented, since each database item is assigned to a counter in each of the L counter groups.

Following processing, L solution sets are formed, one for each of the counter groups. The final solution consists of the intersection of these solution sets, i.e., if an item is reported in the final solution, it must have been in the solution set of each of the L groups. The solution set for each group consists of all items assigned to hot counters in that group. Thus, items reported in the final solution are those that were assigned to hot counters in all of the groups.

2.3. Algorithm C

This algorithm makes use of a single group of M counters. As in algorithm B, the database items are partitioned into M groups of approximately the same size, and each group is associated with one of the counters. In this case, however, any assignment will work. (Random assignment is not necessary). For example, the first N/M items can be associated with the first counter, etc.

A portion (in Section 3 we determine the size of this portion) of the reference string is processed. Afterwards, items are classified into two groups depending on whether their assigned counters were hot or cold. The group of items that were assigned to cold counters is discarded. The remaining (hot) items are then re–partitioned M ways. Items are reassigned to the M counters based on the new partition. The next portion of the reference string is then processed to determine a new hot group. This iteration continues until no hot counter has more than one item assigned to it (or until there are no hot counters). The solution reported by the algorithm consists of the items assigned (individually) to hot counters at the end of the final iteration.

Clearly, if this algorithm is to terminate, the number of items in the hot group should be reduced after each iteration. The number of counters that can be hot after any iteration is no more than $1/T$. For example, if $T = 0.6$, no more than one counter can have received more than sixty percent of the references. This means that the number of cold counters may be as small a $M - 1/T$. Termination can be guaranteed if there is at least one cold counter after every iteration, so that the number of items in the hot group will be reduced. Thus, we will assume that $M - 1/T \geq 1$, i.e., that $M \geq (T+1)/T$.

2.4. Algorithm D

This algorithm uses a set of counters, each with an associated *strength*. In this algorithm, the number of counters in the set varies as the reference string is processed. Initially, the set is empty.

For each reference in the reference string, the following actions are taken. Suppose that the reference under consideration is to item i. If i is assigned to a counter, that counter's value is incremented (as usual) and the strength of the counter it is set to a constant value G. Otherwise, a new counter is created, and i is assigned to it. The value of the new counter is one and its weight is set to G. In addition, after each reference the strengths of all of the other counters in the set are decremented. If a counter's strength reaches zero, the counter is eliminated from the set.

This procedure ensures that an item remains assigned to a counter in the counter set unless G consecutive references occur without a reference to that item. If this occurs, the strength of the item's assigned counter will have decayed to zero, and the counter will have been deleted from the set. Thus, the algorithm "forgets" about items that are not referenced regularly enough.

After processing is completed, the solution consists of all items i that are assigned to a counter whose value (count) is at least C. The purpose of the constant C, known as the counter threshold, is to ensure that only items that remain assigned to a counter consistently are reported in the solution. The counter threshold and the initial strength, G, are parameters of the algorithm.

This algorithm is similar to a several common cache management techniques. It is closest to the working set technique. In fact, if $C = 1$, the solution reported by the algorithm is the working set (as of the end of the reference string) of the process(es) generating the references. It is also similar to LRU cache management, except that under LRU, items are never eliminated from the set until their "slot" is needed for a new item.

3. Analysis

In this section we present an analysis of the algorithms given in Section 2. Our goal is to develop expressions for the probability that an algorithm will not perform correctly, i.e., that a hot or cold item will be misclassified. In several cases we are able to develop exact expressions or good approximations to these probabilities. In other cases we have developed upper bounds on the probability of an incorrect result.

Let us denote the set of items reported by a hot spot diagnosis algorithm by R. An item with a larger weight is at least as likely to be reported (included in R) as one with a smaller weight. (This property is shared by all of the algorithms we have considered.) Thus, a cold item i with $p_i = T - \epsilon$ is at least as likely to be mistakenly reported in the solution as any other cold item. Instead of determining $Prob[\, i \in R \mid i \in C\,]$ for any cold item, our analyses find

$$Prob[\, i \in R \mid p_i = T - \epsilon\,]$$

This number is an upper bound on the probability of any particular other cold item being misclassified by the algorithm. Similarly, we determine

$$Prob[\, i \notin R \mid p_i = T + \epsilon\,]$$

as an upper bound on the probability of any particular hot item being misclassified.

In our analyses, we will make use of the notation shown in Table 1. In addition, there are several algorithm-specific constants described in the previous section. We will introduce them as necessary in each analysis.

symbol	meaning
N	database size (number of items)
T	reference probability threshold
S	sample size (size of the reference string)
μ_N	mean weight of database items
σ^2_N	variance of weight of database items
p_i	weight (reference probability) if item i
ϵ	resolution of the diagnosis

Table 1 – Common Notation

Before we begin, we present a result that we will use frequently in our analyses. Each of the algorithms associate items with counters and then sample the reference stream. Frequently, we will be interested in the probability that a particular counter is "hot" after the sample is completed. We define $P_{samp}(w,s)$ to be the probability that a counter with weight w is hot after a sample of size s. In Appendix A, we show that this probability can be approximated by

$$P_{samp}(w,s) = 1 - \Phi\left(\frac{Ts - ws}{(s(w-w^2))^{1/2}}\right) \tag{3.1}$$

where Φ is the cumulative distribution of a standard normal random variable. The approximation improves as the sample size, s, increases.

3.1. Algorithm A

In algorithm A, items are assigned individually to counters. A cold item with reference probability $T-\epsilon$ will be selected if its assigned counter is hot after sampling. If M counters are available, a total of N/M iterations will have to be performed to check all N items in the database, since only M can be checked during each sample. Since the total sample size is S, each iteration used by Algorithm A can use only S_a references, where

$$S_a = \frac{SM}{N}$$

Using equation 3.1, we can determine the probability that a cold item with reference probability $T-\epsilon$ will be selected when it is tested by the algorithm:

$$Prob[\, i \in R \mid p_i = T - \epsilon] = P_{samp}(T-\epsilon, S_a) = 1 - \Phi\left(\frac{TS_a-(T-\epsilon)S_a}{(S_a(T-\epsilon-(T-\epsilon)^2))^{1/2}}\right) \tag{3.2}$$

Similarly,

$$Prob[\, i \notin R \mid p_i = T + \epsilon] = 1 - P_{samp}(T+\epsilon, S_a) = \Phi\left(\frac{TS_a-(T+\epsilon)S_a}{(S_a(T+\epsilon-(T+\epsilon)^2))^{1/2}}\right) \tag{3.3}$$

3.2. Algorithm B

Recall that algorithm B makes use of L groups of counters with M counters in each group. We will begin by finding a bound on $P_{group}(x)$, the probability that an item with weight x is determined to be hot in a particular group. For a cold item to be selected, it must be selected in all L groups. So,

$$Prob[\, i \in R \mid p_i = T - \epsilon] = \left(P_{group}(T-\epsilon)\right)^L \tag{3.4}$$

Items are assigned randomly to the counters in each group. Define $P_{in}(w,x)$ to be the probability that a counter's weight is w, given that it has had an item with weight x assigned to it. Since we are particularly interested in items of weight $T-\epsilon$, we can write

$$P_{group}(T-\epsilon) = \sum_w P_{in}(w, T-\epsilon) P_{samp}(w, S)$$

In other words, if an item with weight $T-\epsilon$ is to be selected, it must be assigned to a counter with some weight w, and that counter must be hot after sampling is completed. Of course, the counter's weight must be at least $T-\epsilon$, since an item of that weight is associated with the counter. So, we have

$$P_{group}(T-\epsilon) = \sum_{w=T-\epsilon}^{1} P_{in}(w, T-\epsilon) P_{samp}(w, S)$$

Suppose that we choose a set of k points β_i such that

$$T-\epsilon = \beta_1 < \beta_2 < \cdots < \beta_{k-1} < \beta_k = 1$$

We can then use these points to break up the summation above as follows:

$$P_{group}(T-\epsilon) = \sum_{i=1}^{k-1} \sum_{w=\beta_i}^{w<\beta_{i+1}} P_{in}(w,T-\epsilon) P_{samp}(w,S)$$

Taking advantage of the fact that $P_{samp}(w_1,S) \leq P_{samp}(w_2,S)$ if $w_1 \leq w_2$. We can bound P_{group} as follows:

$$P_{group}(T-\epsilon) \leq \sum_{i=1}^{k-1} \left(P_{samp}(\beta_{i+1},S) \sum_{w=\beta_i}^{w<\beta_{i+1}} P_{in}(w,T-\epsilon) \right) \tag{3.5}$$

The more points β_i we choose, the tighter our upper bound will be. For the numerical calculations in the next section, we have used:

$$\beta_2 = T-(\epsilon/2)\,,\ \beta_3 = T\,,\ \beta_4 = T+(\epsilon/2)\,,\ \beta_5 = T+\epsilon$$

The points are clustered around T since the function P_{samp} changes most rapidly near T.

Consider the probability $P_{in}(w,x)$. We assume that an item of weight x has been selected from the database and associated with the counter. Each counter is assigned N/M items total. So, $P_{in}(w,x)$ is the probability that $N/M-1$ items chosen from the rest of the database have weight $w-x$. In Appendix B we derive an approximate expression for $Y(w,n,\mu,\sigma)$. This describes that probability that n items selected from a database of items whose weights have a mean of μ and a variance of σ^2 will have a total weight of no more than w. Then,

$$\sum_{w=a}^{b} P_{in}(w,x) = Y(b-x,\frac{N}{M}-1,\mu^*,\sigma^*) - Y(a-x,\frac{N}{M}-1,\mu^*,\sigma^*)$$

In this expression, μ^* and σ^* are the mean and standard deviation of the database *after an element of weight x has been removed.* (We are assuming that such an item has already been assigned to the counter.) In Appendix C we show that these values can be written in terms of μ_N and σ_N (the mean and variance of the database weights) as follows:

$$\mu^* = \frac{N}{N-1}\mu_N - \frac{x}{N-1}$$

and

$$\sigma^{*2} = \sigma_N^2 + \frac{2x-Nx^2}{N^2}$$

We can now write a complete expression for P_{group}:

$$P_{group}(T-\epsilon) \leq \sum_{i=1}^{k-1} \left(P_{samp}(\beta_{i+1},S) \left(Y(\beta_{i+1}-(T-\epsilon),\frac{N}{M}-1,\mu^*,\sigma^*) - Y(\beta_i-(T-\epsilon),\frac{N}{M}-1,\mu^*,\sigma^*) \right) \right)$$

This upper bound on P_{group} gives us an upper bound on $Prob[\, i \in R \mid p_i = T - \epsilon \,]$, as desired, when it is substituted into equation 3.4.

Using a similar argument, we can derive an expression for the probability that an item with weight $T+\epsilon$ is not reported in the solution. This occurs if the item is not selected in any group.

$$Prob[\, i \notin R \mid p_i = T + \epsilon \,] = 1 - P_{group}(T+\epsilon)^L \tag{3.6}$$

As before, we can write

$$P_{group}(T+\epsilon) = \sum_{w=T+\epsilon}^{w=1} P_{in}(w,T+\epsilon) P_{samp}(w,S)$$

As before, we could break this summation into an arbitrary number of shorter summations to improve our accuracy. However, this is probably unnecessary (unless ϵ is very small). Since the lowest value of P_{samp} occurs when $w = T+\epsilon$, we can place a simple lower bound on $P_{group}(T+\epsilon)$ using

$$P_{group}(T+\epsilon) \geq P_{samp}(T+\epsilon,S) \sum_{w=T+\epsilon}^{w=1} P_{in}(w,T+\epsilon)$$

Since the summation in this expression evaluates to one, our final expression becomes

$$P_{group}(T+\epsilon) \geq P_{samp}(T+\epsilon,S) \tag{3.7}$$

When substituted into equation 3.6, this lower bound on P_{group} gives the desired upper bound on $Prob[\, i \notin R \mid p_i = T + \epsilon \,]$.

3.3. Algorithm C

After each iteration in algorithm C, only items associated with hot counters need be considered during subsequent iterations. Thus, the number of items under consideration is reduced, after each iteration, by the the ratio of hot counters to M. This ratio, which we will term $1/\alpha$, is at most

$$\frac{1}{\alpha} \leq \frac{\frac{1}{T}}{M} = \frac{1}{TM}$$

Thus, the total number of iterations required is no more than $\log_\alpha N$.

Now we can consider the probabilities of false diagnoses of cold and hot items. Consider a cold item with weight $T-\epsilon$ that is accidentally selected. To be selected, the item spends at least one iteration as the only item assigned to a particular counter. The probability that it will be

selected during this iteration is no more than

$$P_{samp}(T-\epsilon, \frac{S}{\log_a N})$$

since the total sample size of S must be distributed over all of the iterations. The probability that the item is actually selected is no more than this, since it must be associated with a hot counter during all iterations. Therefore, we have the upper bound on $Prob[\, i \in R \mid p_i = T-\epsilon \,]$:

$$Prob[\, i \in R \mid p_i = T-\epsilon \,] \leq P_{samp}(T-\epsilon, \frac{S}{\log_a N}) \tag{3.8}$$

Furthermore, unless we are willing to make additional assumptions about the manner in which items are assigned to counters, or about the distribution of the weights of items, this bound is tight. For example, it may happen that the item of weight $T-\epsilon$ is assigned to the same counter as an item of weight $1-(T-\epsilon)$ during all but the last iteration. In this case, the cold item is selected with probability one on all iterations except the last.

To misclassify a hot item as cold, that item must not be assigned to a hot counter during at least one iteration. Since a counter with a hot item assigned to it has a weight of at least $T-\epsilon$, the probability that that counter is hot during a single iteration is at least

$$P_{samp}(T+\epsilon, \frac{S}{\log_a N})$$

Thus we have

$$Prob[\, i \notin R \mid p_i = T+\epsilon \,] \leq 1 - \left(P_{samp}(T+\epsilon, \frac{S}{\log_a N}) \right)^{\log_a N} \tag{3.9}$$

3.4. Algorithm D

For an item to be reported, it must be assigned to a counter with a count of at least C. This will happen if and only if the item has been assigned continuously to the counter since its Cth most recent reference. The probability that this will occur can be conditioned on the event that the Cth most recent reference occurred R references back in the reference stream. We will let $P_r(R,i)$ represent the probability that the Cth most recent reference to an item i occurred R references ago. Also, we will let $P_l(R)$ be the probability that the item has remained on the list since its Cth most recent reference, given that that occurred R references ago. We can then write:

$$Prob[i \text{ is chosen}] = \sum_{R=C}^{S} P_l(R) P_r(R,i) \tag{3.10}$$

The lower limit on the sum is C because $P_r(R,i)$ is zero for $R < C$. The upper limit is because at most S references from the reference stream are considered.

Once assigned to a counter, an item remains assigned unless there are G consecutive references in the stream without a reference to the item. The conditional probability $P_l(R)$ can be computed using the analogy to the well-known problem of distributing indistinguishable balls into distinguishable urns. More precisely, this probability is the probability of having no urn with G or more balls when $R - C$ balls are randomly distributed over C urns. This gives (Appendix D):

$$P_l(R) = \frac{N_G}{\binom{R-1}{R-C}} \tag{3.11}$$

where

$$N_G = \binom{R-1}{R-C} + \sum_{k=1}^{C} (-1)^k \binom{C}{k} \sum_{x=0}^{R-C-kG} \binom{x+k-1}{x} \binom{R-kG-x-k-1}{R-C-kG-x}$$

The probability $P_r(R,i)$ is the probability that one reference to i occurred R references ago, and $C-1$ references to it occurred among the subsequent $R-1$ references. This latter probability has a binomial distribution, so we can express $P_r(R,i)$ as:

$$P_r(R,i) = p_i \binom{R-1}{C-1} p_i^{C-1}(1-p_i)^{R-C} = \binom{R-1}{C-1} p_i^{C}(1-p_i)^{R-C} \tag{3.12}$$

To compute the desired probabilities, we can now use:

$$Prob[\, i \in R \mid p_i = T - \epsilon \,] = \sum_{R=C}^{S} P_l(R) P_r(R, T-\epsilon) \tag{3.13}$$

and

$$Prob[\, i \in R \mid p_i = T + \epsilon \,] = 1 - \sum_{R=C}^{S} P_l(R) P_r(R, T+\epsilon) \tag{3.14}$$

4. Discussion of the Analysis

The meaning of the expressions developed in the previous section can be difficult to discern. In this section, we will attempt to shed some light on the differences among the algorithms by plotting and comparing some of the results of our analyses.

A large number of variables are involved. To keep the presentation as simple as possible, while still pointing out some of the key characteristics of the algorithms, we will focus on the case where the database contains one million items, the reference threshold, T, is 25%. In other words, each algorithm is being asked not to fail to report items which receive more than $(25+\epsilon)\%$ of the references, and never to report items receiving less than $(25-\epsilon)\%$ of the references.

For each algorithm, we have considered the probabilities of false diagnoses (for which we developed equations in the previous section) as a function of the length of the reference string and the number of counters used. The string length is a measure of the speed with which an algorithm arrives at a solution. The number of counters used is a measure of the space overhead of the algorithm. Although the counters are likely to be small relative to the blocks of data, the space overhead is important if the counters are to be maintained in main memory, as may be desirable in some applications.

4.1. Algorithm A

Figures 1 and 2 show the probabilities of false diagnoses for Algorithm A (equations 3.2 and 3.3), as functions of the number of counters used (M) and of the desired resolution (ϵ). For example, the height of the surface at the point (0.05,2000) in Figure 1 gives the probability that an item with weight $T-\epsilon = 0.20-0.05 = 0.2$ will be reported in the solution, given that the algorithm is able to employ 2000 counters simultaneously. The size of the reference string, S, was fixed at ten thousand references.

Clearly, it is possible to achieve very low error rates with Algorithm A. The drawbacks are the large number of counters and the length of the reference string that were used to achieve these low error rates. Indeed, we will see that Algorithms B and C are able to achieve comparable error rates with only a few hundred counters and a much shorter reference string.

Since Algorithm A iterates N/M times, the length of the reference string used should be N/M at the very least, so that at least one reference from the string is available at each iteration. Thus, we should maintain $SM \geq N$. This is unfortunate, since it implies that S and M cannot be varied independently.

Different applications will be interested in different types of hot spots, either longer-term or shorter-term. Longer-term hot spots can be studied by using longer reference strings, i.e., larger values of S. A desirable characteristic of a hot spot algorithm is that for any fixed S, it should be

possible to tradeoff additional space (more counters) for reductions in the probabilities of false diagnoses, and vice versa. If this is the case, then the algorithm will be useful for diagnosing any kind of hot spots, whether long-term or short-term.

Unfortunately, this is clearly not the case for Algorithm A. For example, if only a small about of space is available, the algorithm can not be used to detect short-term hot spots. Since $SM \geq N$ must be maintained, a large value of S will be required. For our example, with $S = 10000$ and a database size of one million, only 20 references are processed during each iteration when the number of counters used is 2000. (One work-around of this problem is for the algorithm to process the entire reference string during *each* iteration, rather than breaking the string into substrings. In an on-line situation, this would require additional space for storing the reference string.)

4.2. Algorithm B

Algorithm B includes two parameters, L and M, which describe the number and size of the counter groups used by the algorithm. The total number of counters used is LM. Figure 3 shows the probability that a cold item will be reported (equation 3.4) as a function of L and M. (The probability of not reporting a hot item is not shown, as it is near zero throughout this range of L and M.) The length of the reference string was fixed at five hundred.

Figure 3 suggests that maintaining $L = M$ is a good idea, at least for small values of L and M. To see this, imagine lines representing constant values of LM (total space) drawn on this surface. The low points of these lines will occur near the main diagonal of the L,M plane.

Figures 4 and 5 are analogous to Figures 2 and 3, and show the probabilities of false diagnoses (equations 3.4 and 3.6) as functions of the resolution (ϵ) and the total number of counters used (LM), assuming $L = M$. A reference string of length 200 was used to create these figures.

The figures indicate that Algorithm B nearly eliminates the probability of false diagnoses when $LM > 200$ and $\epsilon > 0.05$. Thus, the algorithm diagnoses hot spots about as accurately as Algorithm A, but does so using only a fraction of the memory space and reference string size. The reductions in memory space and reference string size (time) are by an order of magnitude, at least for this particular example.

Another advantage of Algorithm B is that any combination of values for S and LM can be chosen. In fact, any two of ϵ, LM, and S can be fixed, and the third can be varied to bring the probabilities of false diagnoses within desired limits. Thus, this algorithm is very flexible when

diagnosing hot spots of any duration. Longer term hot spots can be studied by processing longer reference strings.

4.3. Algorithm C

Figures 6 and 7 show the probabilities of false diagnosis under Algorithm C (equations 3.8 and 3.9), with the length of the reference string fixed at 200. The performance of this algorithm is similar to that of Algorithm B. (Compare Figures 6 and 7 to Figures 4 and 5). Both algorithms make false diagnoses very unlikely (given the default parameters) using about 200 counters.

Algorithm C is iterative, like Algorithm A. Thus, a constraint exists among S, M, and N, requiring a minimum amount of space (M) for any particular value of S. In this case, however, the relationship is logarithmic rather than linear, and thus does not represent a major difficulty. For small values of S, the minimum space required for this algorithm is much less than that for Algorithm A.

4.4. Algorithm D

Algorithm D incorporates two parameters, strength (G) and count threshold (C), which control its behavior. Figures 8 and 9 show the probabilities of false diagnoses (equations 3.13 and 3.14) as functions of G and C. Increasing G increases the probability of a false hot diagnosis (reporting a cold item as hot) but decreases the probability of false cold diagnosis. Decreasing C has a similar, though less dramatic, effect.

It seems clear that the best that can be done in this case is select values of C and G that result in a reasonable compromise between the two types of errors. If we are equally concerned with both types of false diagnosis, the best values of C and G can be obtained by finding the low point of the line formed by the intersection of the surfaces in the two figures. Clearly, the resulting error probabilities will be much higher than those that can be obtained with the other algorithms. Unfortunately, there is no simple way to reduce these probabilities.

Figure 10 illustrates why this situation occurs. The graph plots the probability that an item with a given reference probability (weight) will be reported by the algorithm (equation 3.10). Curves are plotted for three values of C. The strength, G, is constant at $G=3$. Consider a cold item with weight $T-\epsilon$ and a hot item with weight $T+\epsilon$. Clearly, if the probability of selecting the hot item is too low, it can be increased by reducing C. Unfortunately, this also increases the

probability of selecting the cold item! Similar behavior is observed when C is held constant and G is increased. Thus, the algorithms parameters do not provide a mechanism for improving the "resolution" of the algorithm. Improved resolution would correspond to a steeper rise of the curves in Figure 9.

Algorithm D will have difficulty distinguishing among items with similar weights, regardless of the values of C and G. However, it can be useful as a "coarse" filter, for distinguishing between very cold and very hot items. This corresponds to large values of ϵ. To illustrate, Figures 11 and 12 show the probabilities of false diagnosis for two values of G, for the case when $T = 0.5$ and $\epsilon = 0.25$. (These figures represent slices through surfaces such as those in Figures 8 and 9.) In other words, the algorithm is being requested not to fail to report any item with a reference probability greater than 0.75, and never to report any item with reference probability less than 0.25. In for this case, the probabilities of false diagnosis can be reduced to less than 10% (e.g., when $G = 3$ and $C = 5$). Unfortunately, if this level of error is not satisfactory, the only recourse is to increase ϵ.

The first three algorithms exhibited clear relationships among time (the length of the reference string), space, and probabilities of error. This relationship is not as clear for Algorithm D. The number of counters simultaneously in use by the algorithm is never more than G. (G counters will be in use if the most recent G references in the reference string have been unique.) The other important characteristic of the algorithm is that only the last CG references (at most) from the reference string determine which items are reported as hot or cold. Recall from the analysis that an item is selected if and only if it is assigned to a counter continuously from the time of its Cth most recent reference. If the Cth most recent reference is more than CG references old, then the item cannot have remained assigned to a counter, since there would have to have been a gap of at least G between references to the item during that period.

Unfortunately, this implies that the only way to detect long-term hot spots using Algorithm D is to make G large. This can adversely affect the probabilities of false diagnosis, as we have seen, and increases the amount of space used by the technique. Algorithm D is therefore best suited to coarse diagnosis of short-term hot spots.

5. Further Discussion

Several aspects of the hot spot algorithms warrant further discussion. The first concerns the iterative techniques, Algorithms A and C. Each algorithm analyzes a reference string of length S to arrive at its conclusions. The iterative algorithms break the reference string down into several pieces, and then use a different piece during each iteration.

One difficulty with these techniques is that an item that appears hot over the entire reference string may not appear hot during every substring, i.e., it may not appear hot during every iteration. References to it may be clustered in a few of the substrings. Our choice (for reasons of tractability) of an independent reference model "plays down" this possibility in our analyses.

For example, if an item is to be reported as hot by Algorithm C, it must be assigned to a hot counter during all iterations. So, a clustering of references may increase the probability that a truly hot item will fail to be diagnosed. In Algorithm A, a properly diagnosed hot item must have appeared hot during the one iteration that it is assigned to a counter. Since data references in real reference streams are likely to exhibit dependencies, this issue must be considered as a potential strike against iterative algorithms.

A second aspect of these algorithms that should be addressed is their ease of implementation. As might be expected, Algorithms A and D, which exhibit the least desirable properties, are quite simple to implement. Algorithm C is not too much worse, with a small amount of additional complexity due to the reassignment of items to counters at the end of each iteration. Algorithm D complicated somewhat by the need to randomly partition the database among the counters in each counter group. (This must be done independently for each counter group.) The time required for this partition may become an issue for on-line applications of hot spot diagnosis.

6. Summary and Conclusions

We have presented and analyzed four classes of algorithms for the diagnosis of hot spots. Each algorithm attempts to guess, by examining a string of database references, which items from the database are hot, and which are not. The strengths and weaknesses of these techniques can be summarized as follows.

- **Algorithm** A can diagnose hot spots accurately, but may require much more space and time than the other techniques. It is very simple to implement.

- **Algorithm B** uses random collections of database items. Hot spots can be diagnosed accurately, quickly, and without using a great deal of space.
- **Algorithm C** successively refines its diagnosis after each iteration. Like Algorithm B, it is quick, accurate, and uses little space.
- **Algorithm D** is simple to implement but difficult to control. It is best suited to the diagnosis of short–term, very hot hot spots.

We have begun to test these ideas in several applications. One involves detecting correlations between items in reference strings. Specifically, we might like to determine which items are frequently referenced after item i, for all i. A second application involves automatically selecting disk blocks for replication, so that access times can be reduced.

References

[Ba87] Baclawski, K. "A Stochastic Model of Data Access and Communication," Technical Report NU–CCS–87–8, Northeastern University.

[Ga85] Gawlick, D. "Processing of "hot spots" in high performance systems," *Proceedings of COMPCON'85* 1985

[GaKi85] Gawlick, D., and D. Kinkade. "Varieties of concurrency control in IMS/VS Fast Path," *IEEE Bulletin on Data Engineering*, 8(2) pp. 3–10, June, 1985.

[On86] O'Neil, P. "The escrow transactional method," *ACM Transactions on Database Systems*, 11(4) pp. 405–430. December 1986.

[Ou85] Ousterhout, J., et al, "A Trace-Driven Analysis of the UNIX 4.2 BSD File System", Proc. 10th Symposium on Operating Systems Principles, ACM, pp. 15–24, 1985.

[Re82] Reuter. A. "Concurrency on high–traffic data elements," *Proceedings of the first ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pp. 83–92, March 1982.

[Sa91] Salem. K.. "Adaptive Prefetching for Disk Buffers." TR–91–46. CESDIS. Goddard Space Flight Center. Code 930.5. Greenbelt. MD. January. 1991.

[St88] Staelin. C., "File Access Patterns," CS–TR–179–88. Dept. of Computer Science, Princeton University. Princeton. NJ. Sept. 1988.

[Wi89] Wilmot. R.. "File Usage Patternsn from SMF Data: Highly Skewed Usage." Proc. CMG '89. Reno. NV. December. 1989.

Appendix A

We will show here how to approximate $P_{samp}(w,s)$, the probability that a counter with weight w is hot after a sample of length s. Let us define a random variable X as follows:

$$X = \begin{cases} 1 & \text{with probability } w \\ 0 & \text{with probability } 1 - w \end{cases}$$

whose variance is denoted $\sigma_X^2 = w - w^2$. Here, X represents the result of an experiment determining whether the counter is incremented because of an arbitrary reference in the reference string. Let Z_s be sX, the value of the counter after a sample of size s (assuming an initial value of zero). Define

$$Z = \lim_{s \to \infty} \frac{Z_s - sw}{s^{1/2}\sigma_X}$$

By the central limit theorem, Z has the standard normal distribution. So, for large s, we can approximate $P_{samp}(w,s) = Prob[Z_s \geq sT]$ by $Prob[s^{1/2}(w-w^2)Z + sw \geq sT]$, giving

$$P_{samp}(w,s) = 1 - \Phi\left(\frac{sT - sw}{(s(w-w^2)^2)^{1/2}}\right)$$

where Φ is the cumulative standard normal distribution.

Appendix B

We are interested in an experiment in which n weighted items are chosen from a set of (at least n) items whose weights have a mean of μ and variance σ^2. In particular, we would like to find an expression for $Y(w,n,\mu,\sigma)$, the probability that the sum of the weights of the chosen items is w or less.

Since the items are selected without replacement, an exact determination of Y is very difficult. However, if the database is large (relative to n), then selection with replacement will approximate selection without. Intuitively, when the database is large, the probability that an item will be selected more than once (when selection is made with replacement) becomes very small. In the following, we will assume selection with replacement.

Let X represent the weight of a single element selected from the database, and X_n the sum of the weights of n selections. Since the selections are made independently and with replacement, X_n is the sum of n independent random variables each with mean μ and variance σ^2. The central limit theorem indicates that the distribution of X_n approaches normal as n tends to infinity.

Proceeding as in Appendix A, we can approximate $\Upsilon(w,n,\mu,\sigma)$ by $Prob[n^{\frac{1}{2}}\sigma Z + n\mu \leq w]$, where Z has the standard normal distribution. So, under the assumption that n is large, yet small relative to the database size, we have

$$\Upsilon(w,n,\mu,\sigma) = \Phi(\frac{w - n\mu}{n^{\frac{1}{2}}\sigma})$$

which can be determined numerically or from a tabulation of the standard normal distribution.

Appendix C

We are given a set of N weighted items. The sum of the weights is W_{tot} and their variance is σ_N^2. Item i, with weight p_i, is selected from the set. We wish to determine μ^* and σ^{*2}, the mean and variance of the weights of the remaining $N-1$ items in the set.

Clearly, $\mu_N = W_{tot}/N$ and $\mu^* = (W_{tot}-p_i)/N-1$, so we have:

$$\mu^* = \frac{N}{N-1}\mu_N - \frac{p_i}{N-1}$$

The variance is a little trickier. By definition, we have:

$$\sigma^{*2} = \frac{1}{N-1}\sum_{j\neq i} p_j^2 - \mu^{*2}$$

Expanding gives:

$$\sigma^{*2} = \frac{1}{N-1}\sum_{j\neq i} p_j^2 - \left(\frac{W_{tot}-p_i}{N-1}\right)^2 = \frac{N}{N-1}\frac{1}{N}(\sum_j p_j^2 - p_i^2) - \frac{W_{tot}^2-2W_{tot}p_i+p_i^2}{(N-1)^2}$$

Since

$$\sigma_N^2 = \frac{1}{N}\sum_j p_i^2 - \frac{W_{tot}^2}{N^2}$$

our expression becomes

$$\sigma^{*2} = \frac{N}{N-1}(\sigma_N^2 + \frac{W_{tot}^2}{N^2} - \frac{p_i^2}{N}) - \frac{W_{tot}^2-2W_{tot}p_i+(p_i)^2}{(N-1)^2}$$

For $N \gg 1$ and $W_{tot} = 1$, the expression simplifies to

$$\sigma^{*2} = \sigma_N^2 + \frac{2p_i-Np_i^2}{N^2}$$

Appendix D

We want to compute the probability of having no urn with w or more balls when distributing B balls over U urns. Let N_{tot} be the total number of ways of distributing B balls into U urns. Then

$$N_{tot} = \begin{pmatrix} B + U - 1 \\ B \end{pmatrix}$$

Now, let $N_w(k)$ be the number of ways of distributing B balls into U urns in such a way that k urns end up with w or more balls. Let $N_w = N_w(0)$, i.e., the number of ways of distributing the balls so that no urn ends up with w or more balls. According to the principle of inclusion–exclusion, we have:

$$N_w = N_{tot} + \sum_{k=1}^{k^*} (-1)^k N_w(k)$$

where

$$k^* = \min \left[\left\lfloor \frac{B}{w} \right\rfloor, U \right]$$

Furthermore, $N_w(k)$ can be expressed as:

$$N_w(k) = \begin{pmatrix} U \\ k \end{pmatrix} \sum_{x=0}^{B-kw} \begin{pmatrix} x + k - 1 \\ x \end{pmatrix} \begin{pmatrix} B - kw - x + U - k - 1 \\ B - kw - x \end{pmatrix}$$

In the last equation, x represents the number of balls above kw that are placed in the selected urns. The first term in the sum represents the number of ways of placing these x balls in the k selected urns, while the second term represents the number of ways of placing the remaining balls in the remaining urns. The desired probability is then given by N_w / N_{tot}.

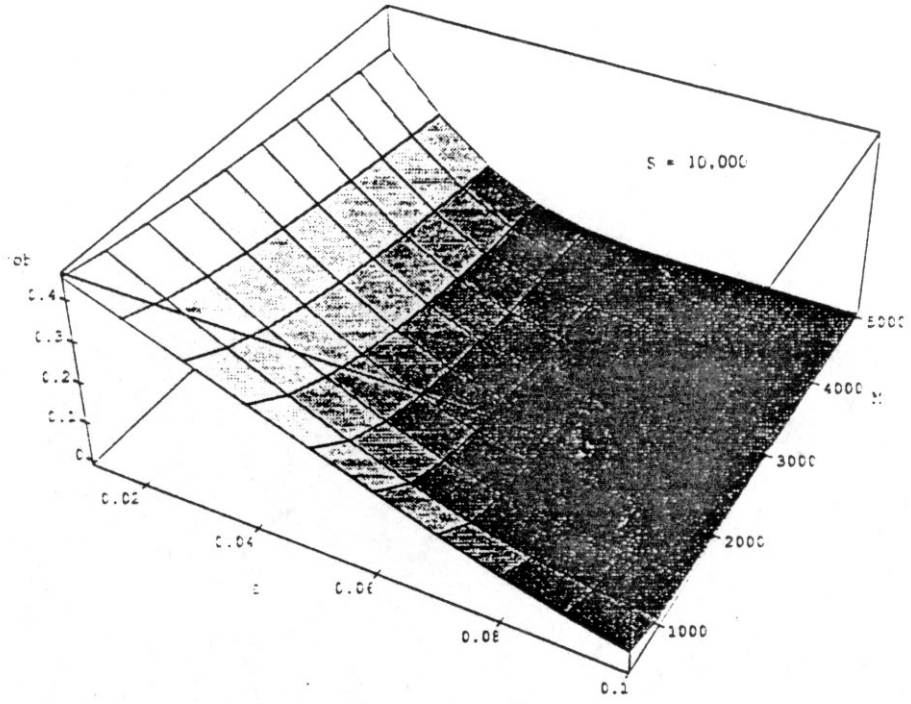

Figure 1 - Algorithm A

Probability of Not Reporting a Hot Item

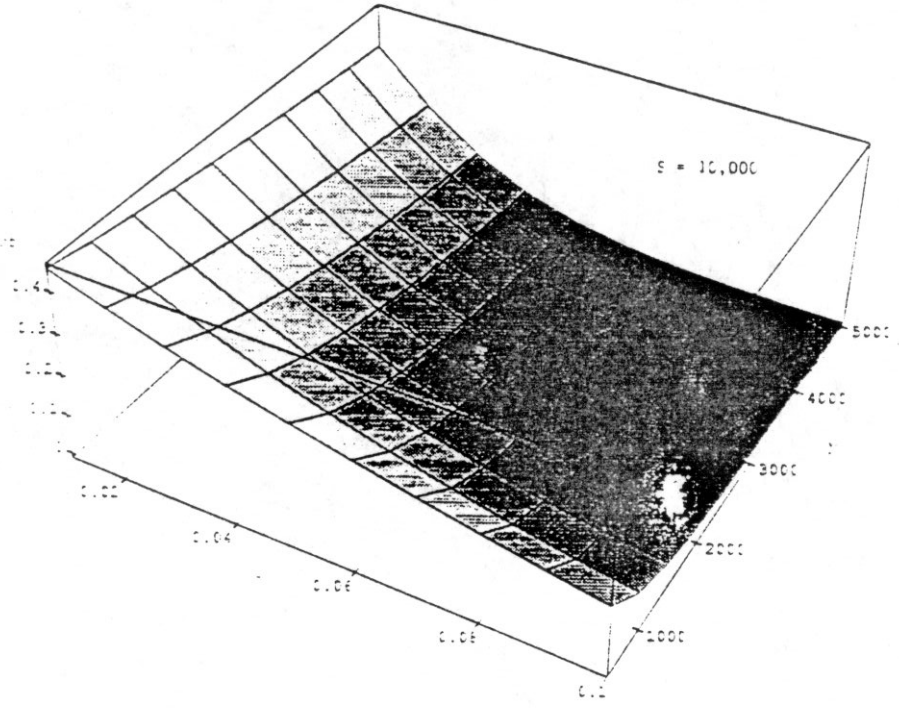

Figure 2 - Algorithm A

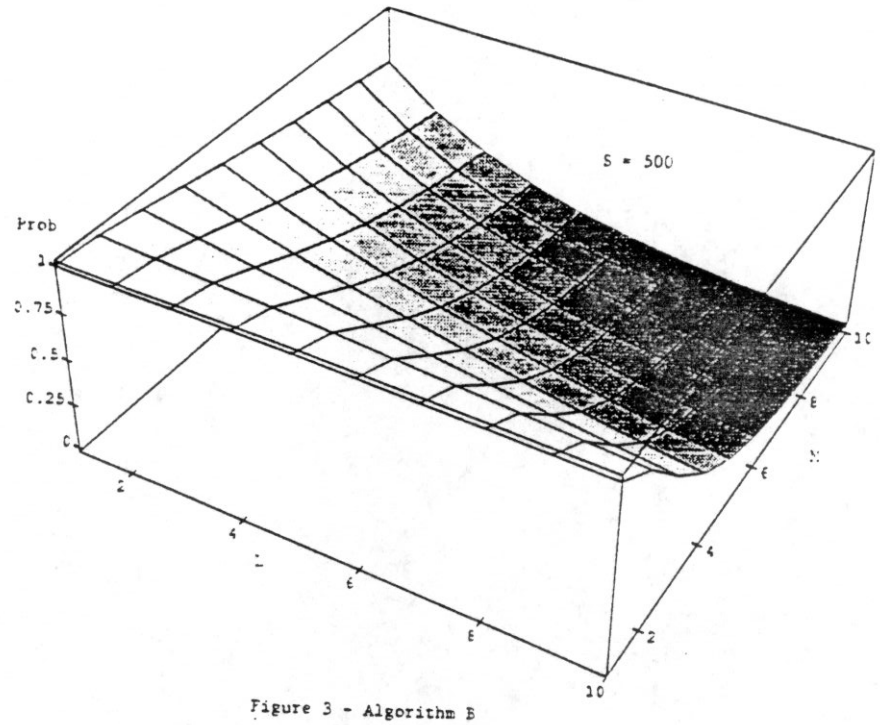

Figure 3 - Algorithm B

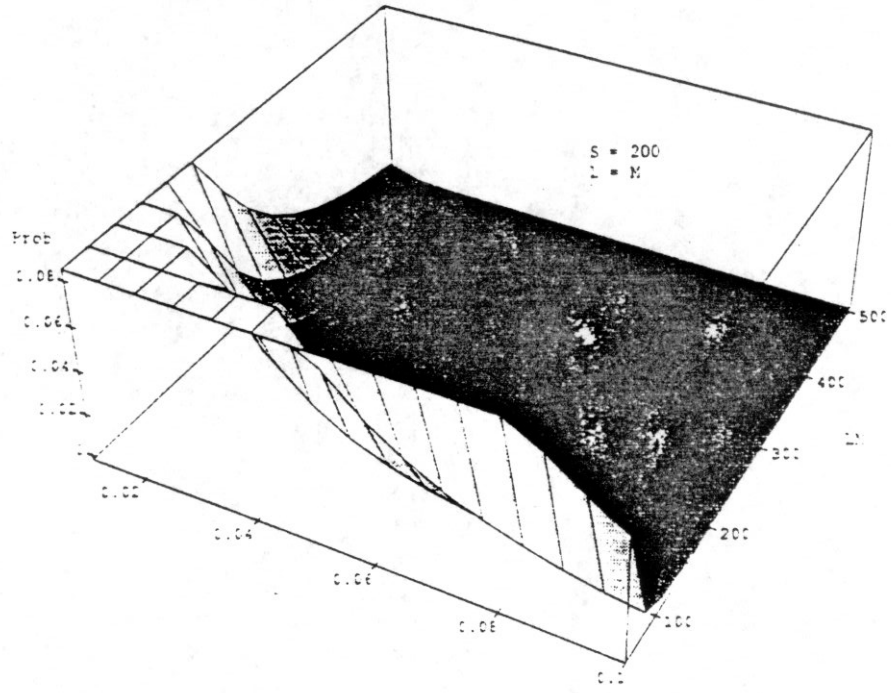

Figure 4 - Algorithm B

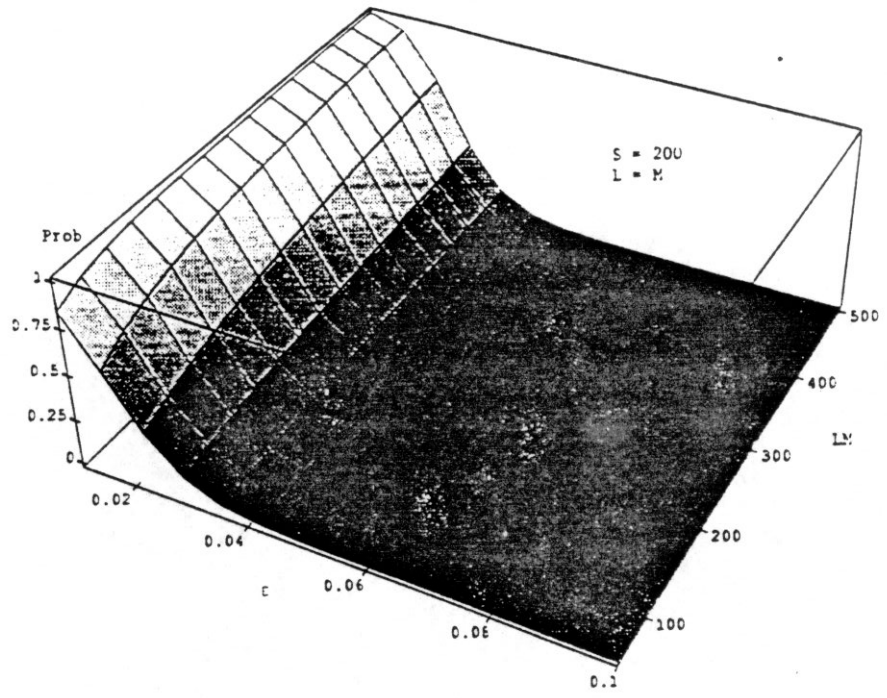

Figure 5 - Algorithm B

Probability of Reporting a Cold Item

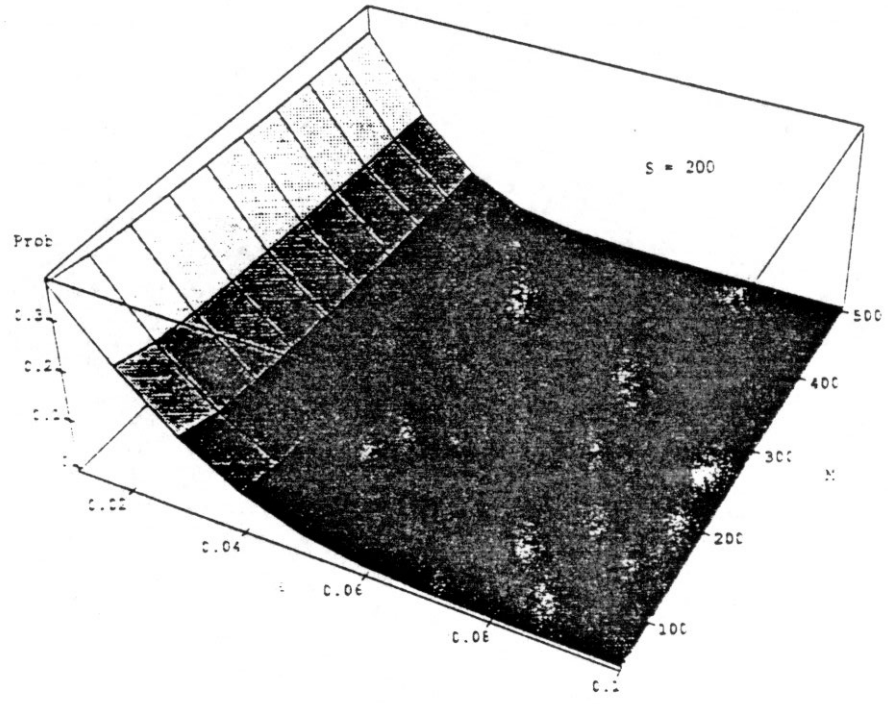

Figure 6 - Algorithm C

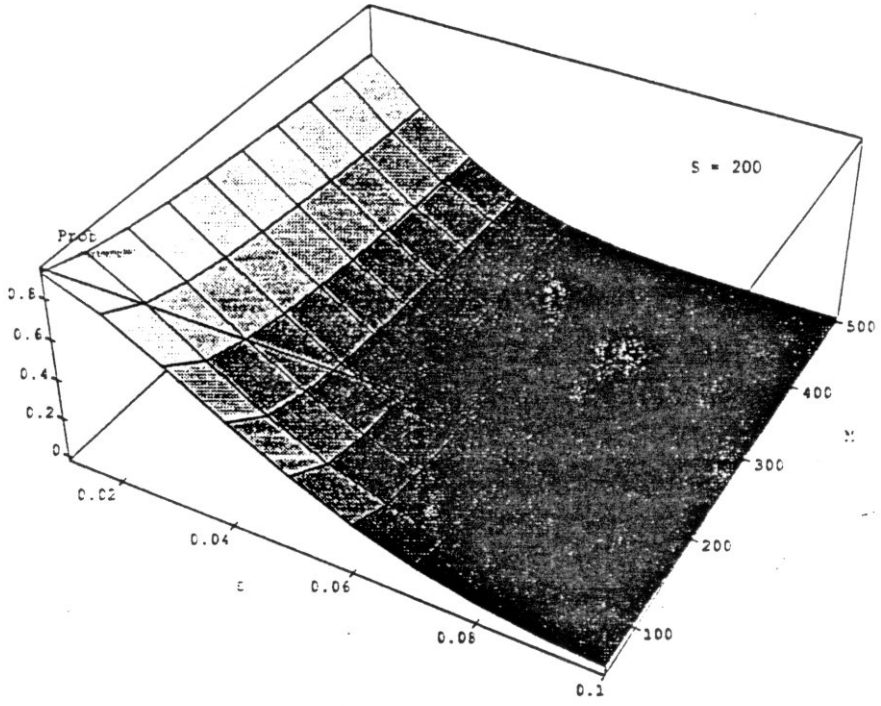

Figure 7 - Algorithm C

Probability of Reporting a Cold Item

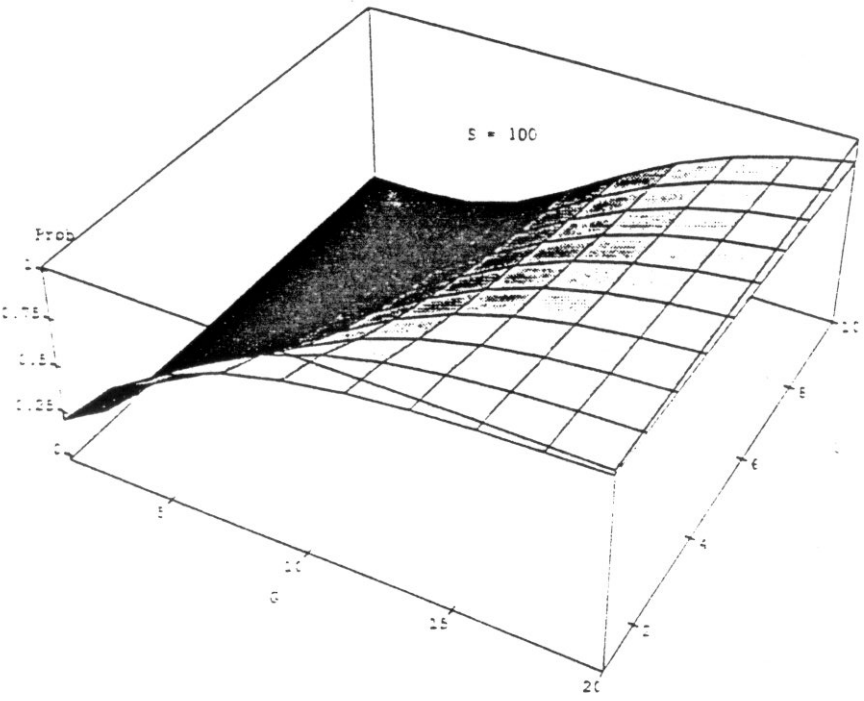

Figure 8 - Algorithm D

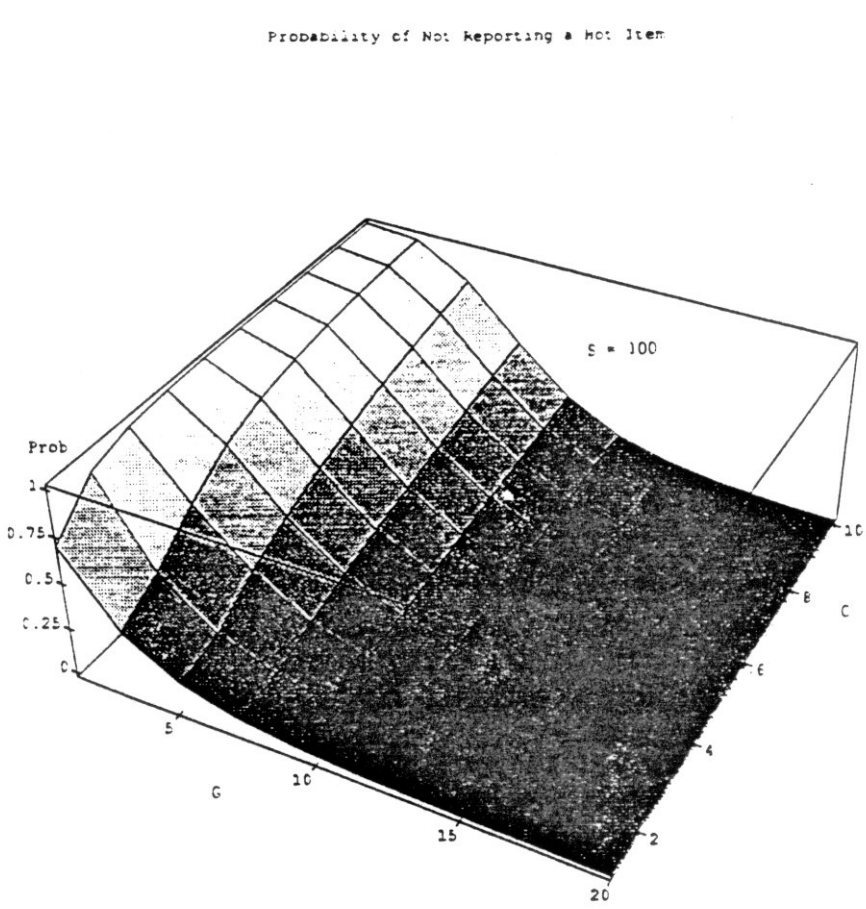

Figure 9 - Algorithm D

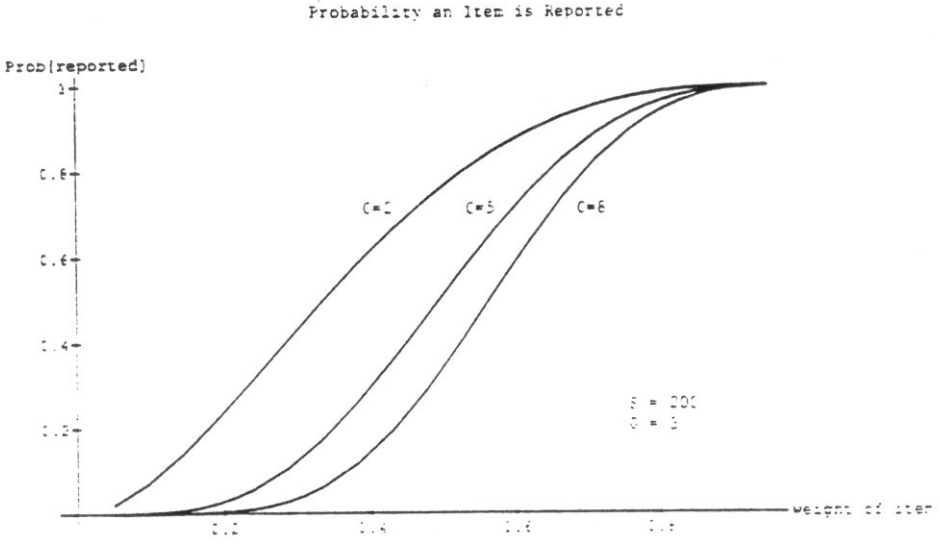

Figure 10 - Algorithm D

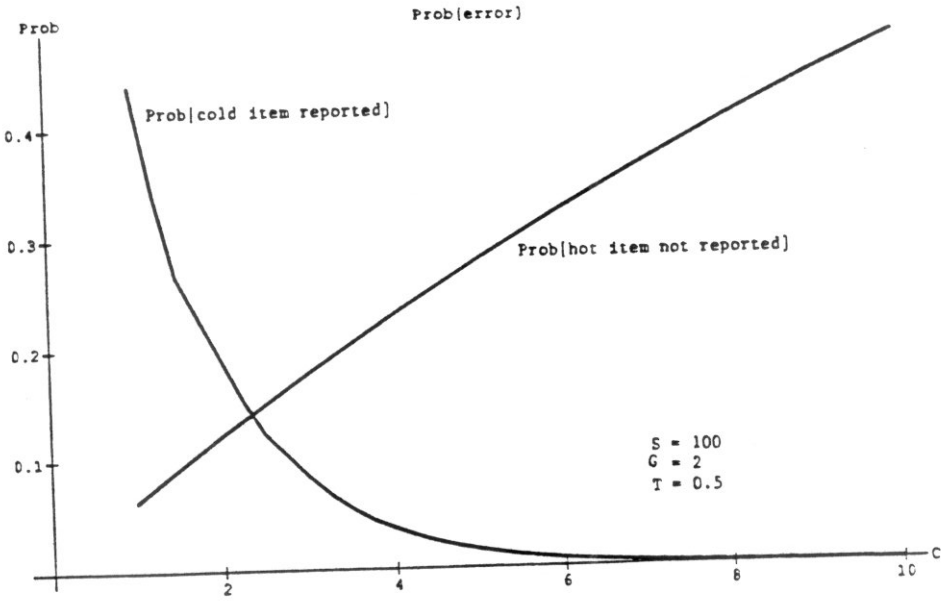

Figure 11 - Algorithm D

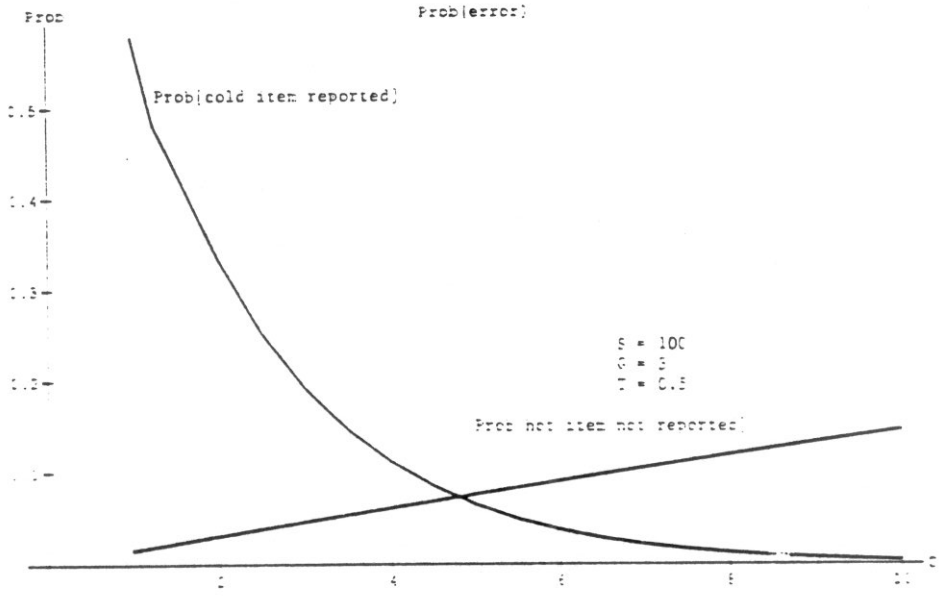

Figure 12 - Algorithm I