2-1972

SOME RESULTS ON THE BEST MATCH PROBLEM

Luther D. Rudolph
Syracuse University

Kishan Mehrotra
Syracuse University, mehrotra@syr.edu

Ralph J. Longobardi

72-1

SOME RESULTS ON THE BEST MATCH PROBLEM

LUTHER D. RUDOLPH

KISHAN G. MEHROTRA

RALPH J. LONGOBARDI

FEBRUARY 1972

SYSTEMS AND INFORMATION SCIENCE

SYRACUSE UNIVERSITY

SOME RESULTS ON THE BEST MATCH PROBLEM

LUTHER D. RUDOLPH

KISHAN G. MEHROTRA

RALPH J. LONGOBARDI

This work was supported by the Rome Air Development Center under Contract F30602-70-C-0190.

SYSTEMS AND INFORMATION SCIENCE

SYRACUSE UNIVERSITY

SYRACUSE, NEW YORK 13210

(315) 476-5541 EXT. 2368

ABSTRACT

The "best-match problem" is concerned with the complexity of finding the best match between a randomly chosen query word and the members of a randomly chosen set of data words. Of principal interest is whether it is possible to significantly reduce the search time required, as compared to exhaustive comparison, by use of memory redundancy (file structure). Minsky and Papert conjecture that "the speed-up values of large memory redundancies is very small, and for large data sets with long word lengths there are no practical alternatives to large searches that inspect large parts of memory". In this report we present two algorithms that do yield significant speed-up, although at the cost of large memory redundancies. (Whether these algorithms constitute counterexamples to the Minsky-Papert conjecture depends on one's interpretation of their term "large memory redundancies".) The algorithms are subjected to statistical analysis and time-memory trade-off curves are given.

ACKNOWLEDGEMENT

The authors wish to acknowledge the contributions to this study by Professor Harry Schwarzlander of the Electrical Engineering Department, Syracuse University.

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

In the program of the "Regional Conference on Phenomena that need Basic Computational Theories" held at Pennsylvania State University in September, 1970, Professor Marvin Minsky of the Massachusetts Institute of Technology wrote the following:

"Most work on the Theory of Computation has been concerned with questions about what can, and what cannot, be computed by various kinds of machines. The results have been mainly of an all-or-none quality; little attention was paid, in the development of the theories of Automata and of Recursive Functions, to the problems of computational effort, or amounts of memory, or other aspects of complexity required to compute things that can clearly be done in principle --- using unlimited time and memory. Even in those few studies of relative amounts of computation, the problems chosen for study have usually been so abstract or combinatorial that we have not often found them helpful for insight into real problems, either in the traditional areas of mathematical algorithms, the newer fields of symbolic mathematical computations, or in our own specialties of automata, learning theories, pattern recognition and other aspects of artificial intelligence.

"In the past few years, however, we have seen steps that may be leading toward more realistic theories. The trouble, as I see it, is that mathematics does not develop in a healthy way, except in the context of very thorough understanding of the fundamental phenomena involved in non-trivial, but very simple, situations. As shown dramatically by the discoveries in the past few years on the complexity of simple arithmetic multiplication, the field of computation has been distinctly backward in respect to asking and answering simple but fundamental questions. But we are on the threshold of acquiring such a stock of elements of basic understanding, I think..... .

"The results ... are still rather fragmentary and anecdotal. Nevertheless, we expect them to lead to some unifications of scattered bits of knowledge, and eventually to systematic theories of computation. Right now, I feel that the most promising directions for work lie in unravelling the prototypes of basic conservation laws -- or laws of exchange -- between intuitively important quantities. The most attractive of these are, in our present stage of thinking, the exchanges between: amounts of memory, amounts of computing hardware, and amounts of time required for computation ... ."

Some interesting and provocative research along these lines has been initiated by Minsky and Seymour Papert. Their findings are described in their excellent book, Perceptrons, an Introduction to Computational Geometry (M.I.T. Press, Cambridge, Mass. 1969). Relevant to this report are the sections on the "exact match problem" and the "best match problem" (pages 205-225) in which they discuss the trade-off between time and memory for two superficially similar computations that arise in information retrieval and pattern classification systems. The investigation described in this report was motivated by Minsky and Papert's work on the exact match and best match problems, and in particular by their conjecture on the gloomy prospects for best matching algorithms.

In Section 2, we establish the framework within which the time-memory trade-off is considered and then describe the exact match and best match problems together with the Minsky-Papert conjecture on best matching algorithms. Then in Sections 3 and 4 we present two algorithms which, under our interpretation, constitute counterexamples to the conjecture. Conclusions and suggestions for further work are given in Section 5.

SECTION 2

THE PROBLEM

In this section, we establish the framework within which the trade-off between time and memory for exact matching and best matching is studied, and then describe the exact match and best match problems. Finally we state and interpret the Minsky-Papert conjecture on best matching algorithms. The material in this section is based on Sections 12.6 and 12.7 of Perceptrons.

2.1 The exact match problem

Suppose that we are given a body of information-- we will call it a data set -- in the form of 2^a binary words each b digits in length; one can think of them as 2^a points chosen at random from a space of 2^b points. Following Minsky and Papert, we will take a = 20, b = 100 (i.e. a data set consisting of roughly a million words of length 100) to be typical of the sorts of data sets under consideration. We will suppose that the data set is to be chosen at random from among all possible sets so that one cannot expect to find much redundant structure within it. The ordered data set requires about $b \cdot 2^a$ bits of binary information for complete description. We will not, however, be interested in the order of the words in the data set. This reduces the amount of information required to store the set to about $(b-a)\cdot 2^a$ bits.

We want a machine that, when given a random b-digit word w, will answer

Question 1 (exact match): <u>Is w in the data set</u>?

and we want to formulate constraints upon how this machine works in such a way that we can separate computational aspects from memory aspects. To this end, we adopt the following scheme.

We will allow our machine a memory of M separate bits, that is, one-digit binary words. We are required to compose in advance, before we see the data set, two algorithms A_{file} and A_{find} that satisfy the following conditions:

1. A_{file} is given the data set. Using this as data, it fills the M bits of memory with information. Neither the data set nor A_{file} are used again, nor is A_{find} allowed to get any information about what A_{file} did, except by inspecting the contents of M.
2. A_{find} is then given a random word, w, and asked to answer Question 1, using the information stored in the memory by A_{file}. We are interested in how many bits A_{find} has to consult in the process.

3. The goal is to optimize the design of A_{file} and A_{find} to minimize the number of memory references in the question-answering computation, averaged over all possible words w.

Let N* (a,b,M) denote the number of bits referenced, averaged over all possible words w, using the best possible A_{file} - A_{find} pair for each value of a,b and M. For given fixed a and b, we would like to be able to plot a curve of N* as a function of M. At our present state of knowledge, however, the best we can hope to do is to find some points that bound this curve and tell us something about its general form.

As one might imagine, it is a very difficult matter to say, for a given value of M, what A_{file} - A_{find} pair is best. However, Minsky and Papert have identified several values of M for which optimal or near-optimal A_{file} - A_{find} pairs can be specified. The two simplest cases are (1) when M is the minimum number of bits required to answer the question, in which case there is no memory redundancy and an exhaustive search is probably required, and (2) when M is large enough to allow the question to be answered by table look-up. Let M_{min} and M_{max} denote the number of bits in the memory at these extremes. It is intuitively clear that the maximum number of bit references N^*_{max} occurs when $M = M_{min}$, the minimum number of bit references N^*_{min} occurs when $M = M_{max}$, and that N^* is a monotonically nonincreasing function of M between these extremes. The region of interest is depicted in Figure 1.

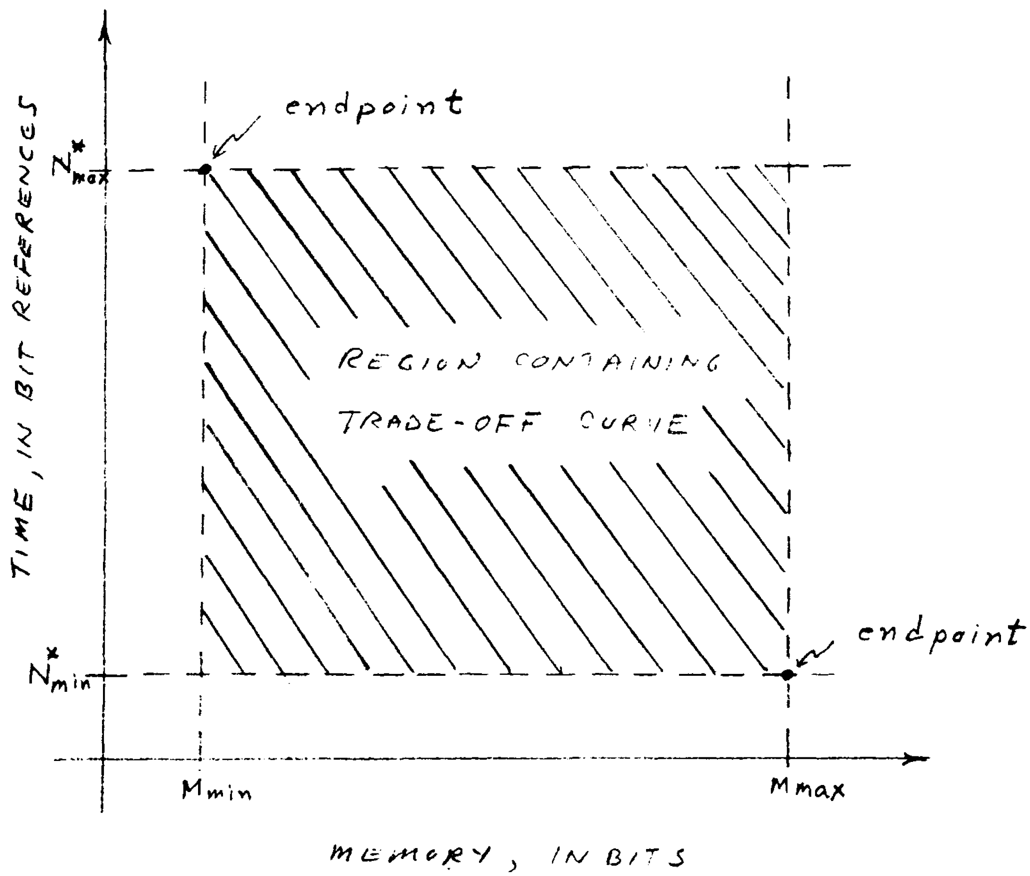


Figure 1. Boundaries and endpoints of the time-memory curve.

The minimum number of bits required to answer Question 1 is roughly $M_{min} = (b-a) \cdot 2^a$ and the corresponding number of bit references is about $N_{max} = 1/2(b-a) \cdot 2^a$. In this case we use just enough memorv to store the unordered data set and A_{find} is an exhaustive search algorithm.

At the other extreme, we have a one-digit word M_w for each possible query word w, where $M_w = 1$ if w is in the data set and $M_w = 0$ otherwise. For a given w, it is necessary only to look up M_w which requires one bit reference. Hence $M_{max} = 2^b$ and $N_{min} = 1$.

In order to determine the general form of the N* vs. M curve between these endpoints, Minskv and Papert identify two other values of M for which verv efficient A_{file} - A_{find} algorithms are known. The first is $M = b \cdot 2^a$. Here the A_{file} algorithm stores the data set in ascending numerical order and A_{find} performs a binarv search to see which half of memory might contain w, then which quartile, etc., i.e. a binary logarithmic sort. The number of bit references in this case is roughlv 1/2 a·b.

The other value is $M = 2b \cdot 2^a$ which is twice the memorv required to store the ordered data set. Here Minsky and Papert choose the A_{file} - A_{find} pair to be a hash coding scheme and show that the number of bit references is roughlv $N = 4$.

These results are summarized in table 1. Although onlv four points that upper bound the time-memorv curve have been identified, the general form of the curve is clearlv that depicted in Figure 2. A verv small amount of memorv redundancv --roughlv a factor of two--reduced the number of bit references from N^*_{max} almost to N^*_{min}.

memorv size M	no. of bit references N	A file-A find
$M_{min}=(b-a)\cdot 2^a$	$N_{max}=1/2(b-a)2^a$	exhaustive search
$M=b\cdot 2^a$	$N=1/2\ b\cdot a$	log sort
$M=2b\cdot 2^a$	$N=4$	hash coding
$M_{max}=2^b$	$N_{min}=1$	table look-up

Table 1. Some points that bound the time-memory curve for exact matching.

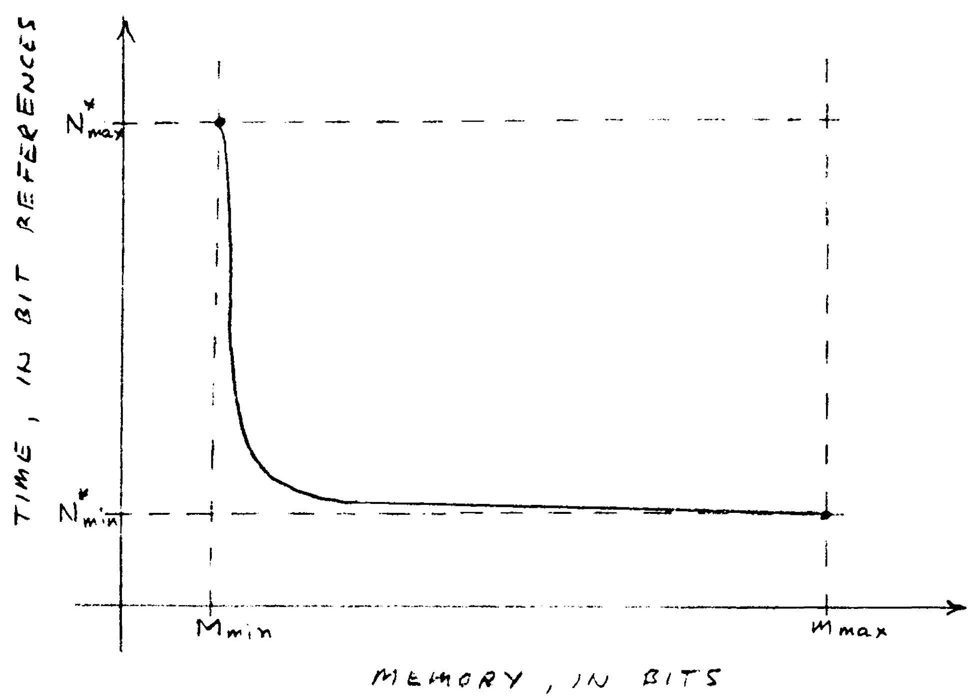


Figure 2. General form of the time-memorv curve for exact matching.

2.2 The best match problem

We next consider

Question 2 (best match): Given w, exhibit the word $\hat{w}$ closest to w in the data set.

The ground rules for A_{file} and A_{find} are the same, and "closeness" is measured by Hamming distance. If $x_1,\ldots,x_b$ and $\hat{x}_1,\ldots,\hat{x}_b$ are the (binary) coordinates of points w and $\hat{w}$, then the Hamming distance is defined to be

$$d(w,\hat{w}) = \sum_{i=1}^{b} |x_i - \hat{x}_i|,$$

i.e. $d(w,\hat{w})$ is the number of positions in which w and $\hat{w}$ disagree. Then Question 2 asks that, given w, we find a $\hat{w}$ in the data set that minimizes $d(w,\hat{w})$.

As in the exact match problem, it is relatively easy to identify the extremes. The minimum amount of memory required to answer Question 2 is again roughly $M_{min} = (b-a)\cdot 2^a$ and the corresponding exhaustive search algorithm presumably requires about $N_{max} = (b-a)\cdot 2^a$ bit references. At the other extreme, we have a b-digit word M_w for each possible query word w, with $M_w = \hat{w}$ where $\hat{w}$ is a word in the data set closest to w. For a given w, it is necessary only to look up M_w and read out $\hat{w}$ which requires b bit references. Hence $M_{max} = b\cdot 2^b$ and $N_{min} = b$. The boundaries and endpoints of the time-memory curve for best matching are the same as for exact matching as depicted in Figure 1. However, here the similarity of the two problems ends. According to Minsky and

Papert, there are no useful results known for $(b-a)\cdot 2^a < M < b\cdot 2^b$. However, it is clear that small amounts of memory redundancy are not going to cause a drastic reduction in the number of bit references required to answer Question 2. An extremely pessimistic view is expressed in

> The Minsky-Papert Conjecture: "Even for the best possible A_{file} - A_{find} pairs, the speed-up value of large memory redundancies is very small, and for large data sets with long word lengths, there are no practical alternatives to large searches that inspect large parts of the memory." (Perceptrons, page 223)

One of the problems faced by anyone who tries to prove or disprove this conjecture is how to interpret the term "large memory redundancies". If we let $M = M_{max}$, then we certainly obtain a large speed-up, so this much redundancy is certainly too large. Rather than try to establish a measure of "largeness" directly, we have chosen to interpret the conjecture in terms of the general form of the time-memory trade-off curve. In particular, we will interpret the conjecture to mean that the time-memory curve is concave on the interval (M_{min}, M_{max}) as illustrated in Figure 3. We apologize to the authors of the conjecture if our interpretation seems unreasonable to them, in the same spirit that they apologized to the readers of Perceptrons for not having a more precise statement of the conjecture.

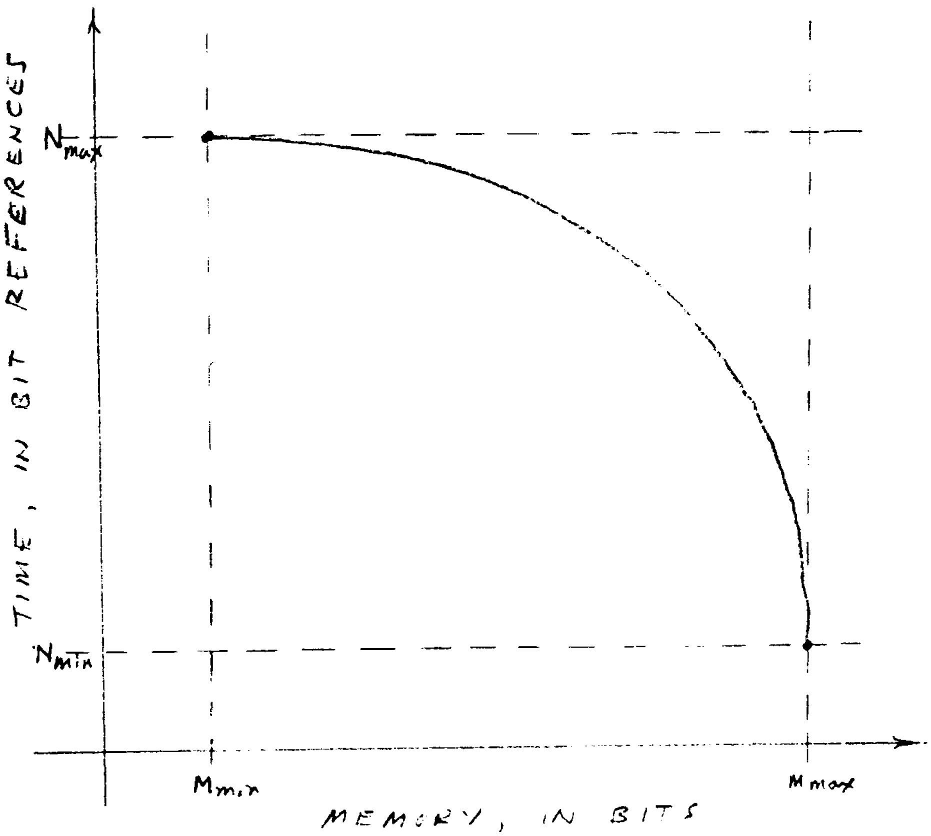


Figure 3. Form of the time-memory curve for best matching based on our interpretation of the Minsky-Papert conjecture.

We will now show, by means of counterexamples, that (our interpretation of) the Minsky-Papert conjecture is false.

SECTION 3

ALGORITHM I

In this section, we present an A_{file} - A_{find} pair, which we refer to as Algorithm I, that achieves a significant reduction, as compared to an exhaustive search, in the number of bit references required to answer Question 2. The amount of memory required is quite large compared to M_{min}, the minimum amount of memory required to answer Question 2, but is also quite small compared to M_{max}, the amount of memory required to answer Question 2 by table look-up. The reader will have to decide for himself whether or not this algorithm constitutes a counterexample to the Minsky-Papert conjecture. Under our interpretation of the conjecture, it does.

3.1 Description

In Section 2.1, it was pointed out that one can think of the data set as 2^a points chosen at random from a space of 2^b points. Distance in this space is measured according to the Hamming metric. A (Hamming) sphere of radius t and center c is the set of all points distance t or less from c. There are $1+b+\binom{b}{2}+\ldots+\binom{b}{t} = \sum_{i=0}^{t} \binom{b}{i}$ such points. Since there are 2^b points in the space, it is conceivable that we could pack $2^b / \sum_{i=0}^{t} \binom{b}{i}$ spheres into the space in such a way that each point is contained in one and only one sphere, i.e. that the spheres fill up the space without overlapping. For certain values of b and t this is possible (e.g. $b = 23$ and $t = 3$), but usually the spheres do not fit together exactly and a perfect packing can only be approximated. Since we are only interested in (i.e. able to obtain) order-of-magnitude results, however, we will pretend that a perfect packing is possible for all values of b and t. So let us assume that the space of 2^b points has been partitioned into $2^b / \sum_{i=0}^{t} \binom{b}{i}$ spheres of radius t, where c_i is the center of the i^{th} sphere. To the i^{th} sphere we assign a memory location L_i. The number of bits at L_i is left unspecified. The partition into spheres and assignment of memory locations are of course done prior to seeing the data set.

We can now give an informal description of the A_{file} algorithm. Let D_i be the distance from c_i to a nearest word in the data set. Then in location L_i store those data words whose distance from c_i is no greater than D_i+2t. After this has been accomplished for $i = 1,2,\ldots,2^b / \sum_{i=0}^{t} \binom{b}{i}$, the data set and A_{file} are never used again.

The A_{find} algorithm operates as follows: Given a query word w, find the i such that w is contained in the i^{th} sphere. Then determine, by exhaustive comparison, which data word at location L_i is closest to w. The resulting data word is $\hat{w}$.

That this A_{file} - A_{find} pair always gives a correct result is guaranteed by the triangle inequality for the Hamming metric. Suppose w is in sphere i, α is a data word closest to c_i and β is a data word closest to w. This situation is shown in Figure 4. By the triangle

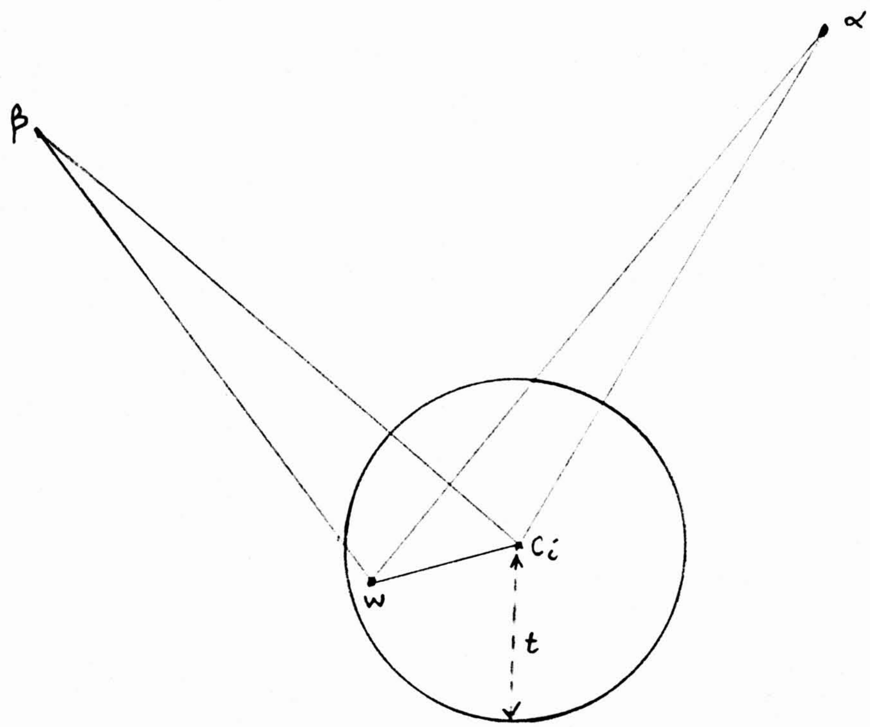


Figure 4. Geometric interpretation of the proof that Algorithm I always produces a data word $\hat{w}$ that is closest to the query word w.

inequality, we have

$$d(\beta,c_i) \leq d(\beta,w) + d(w,c_i)$$

$$d(w,\alpha) \leq d(w,c_i) + d(c_i,\alpha) .$$

Since β is a word closest to w, we also have

$$d(\beta,w) \leq d(w,\alpha) .$$

Combining these inequalities gives

$$d(\beta,c_i) \leq d(c_i,\alpha) + 2d(w,c_i)$$

But $d(c_i,\alpha) = D_i$ and $d(w,c_i) \leq t$.

Hence

$$d(\beta,c_i) \leq D_i + 2t$$

which means that β is one of the data words stored at location L_i by A_{file}.

We remark here that we are attempting to exploit the distribution of distances among points in a high-dimensional space. We know that if we pick an arbitrary word c_i in the space, the distance between c_i and the words in the data set is binomially distributed. For large a and b, this means that 'almost all' of the data words will be close to distance b/2 from c_i. However, the distance D_i to the <u>nearest</u> data word will, on the average, be considerably less. The hope is that the expected number of data words in a sphere of radius D_i+2t centered at c_i (the points A_{file} stores at location L_i) will be small. We will see shortly that this is the case if we choose the radius t to be small enough.

3.2 Analysis

We now give exact and approximate formulas for the expected memory size M and expected number of bit references N as a function of a,b and t, time-memory curves for b=100, and asymptotic results. Note that t is a parameter that traces out a time-memory curve for Algorithm I as it varies over the range $0 \le t \le b$. At $t = 0$, there is a sphere of radius 0 centered at each of the 2^b points in the space, and we need store only one data point at each location L_i. At this extreme, Algorithm I becomes a table look-up algorithm. At $t = b$, there is only one sphere, containing all the data points, and we are forced to compare w with every point in the data set. At this extreme, Algorithm I becomes an exhaustive search.

E(M), the expected size of the memory, and E(N), the expected number of bit references, using Algorithm I are given by

$$E(M) = \frac{b2^a}{\sum_{i=0}^{t} \binom{b}{i}} \sum_{d_0=0}^{b} \left\{ \left[\sum_{x=d_0}^{b} \binom{b}{x} \left(\frac{1}{2}\right)^b \right]^{2^a} - \left[\sum_{x=d_0+1}^{b} \binom{b}{x} \left(\frac{1}{2}\right)^b \right]^{2^a} \right\} \sum_{i=0}^{d_0+2t} \binom{b}{i}$$

$$E(N) = 2^{-b} \sum_{i=0}^{t} \binom{b}{i} \, E(M)$$

These formulas are derived in the appendix, along with approximations which were used for actual computations.

Time-memory curves using Algorithm I for b = 100 and selected values of a are shown in Figures 6 through 10 (at the end of the report).

One characteristic of these curves that is immediately apparent is a sharp drop-off in the expected number of bit references when E(M) exceeds a certain value. It is of interest to see what happens to this threshold as the length of the data word and the size of the data set are increased without bound. In order to fix the relative information storage capacities of the data set and the space from which it is selected, we define the parameter

$$r = \frac{\log_2 \text{(data set size)}}{\log_2 \text{(space size)}} = \frac{a}{b}$$

which we call the <u>density</u> of the data set. For purposes of obtaining asymptotic results, it is also convenient to define a second dimensionless quantity

$$R = \frac{\log_2 [E(M)/M_{min}]}{\log_2 [M_{max}/M_{min}]}$$

which we call the <u>memory redundancy</u>. It is shown in the appendix that, for a given density r, the asymptotic time-memory curve is a step function where the step, or threshold, occurs at a memory redundancy of

$$R_{c_1} = (1-r)^{-1} [1-H\{1/4 - 1/2H^{-1} (1-r)\}].$$

where $H(x) = -x\log_2 x - (1-x)\log_2(1-x)$ is the binary entropy function.

We call R_{c_1} the <u>critical memory redundancy</u> for Algorithm I. It is interesting to note here that the location of the threshold relative to M_{min} and M_{max} is asymptotically only a function of the data set density.

The following question arises quite naturally in a study of this sort. Suppose we don't insist that the answer to the question be correct 100 per cent of the time, but only, say 99 per cent. Does this drastically reduce the time and/or memory required? And in general, how does the computational complexity vary as a function of the allowed probability of error? An obvious way to modify Algorithm I to reduce memory redundancy at the cost of an occasional error is to reduce the number of data points stored at the various locations. This is most easily done by storing at L_i those data points whose distance from c_i is no more than $D_i + k$, where k is a nonnegative integer less than 2t. Because the distances are binomially distributed, we would expect a significant reduction in M at the cost of a very small probability of error when k is slightly smaller than 2t. Unfortunately, this is not easy to verify by analysis. The only case that we considered is the extreme case where we let k=0 and store at L_i only a single data word closest to c_i. In this case

$$M = b2^b \Big/ \sum_{i=0}^{t} \binom{b}{i}$$

$$N = b.$$

The probabilitv of a correct answer to Question 2 when onlv a single data word is stored at each location is shown in Figure 5 for b = 25, a = 5 and various values of t. (See the appendix for details of the analvsis.)

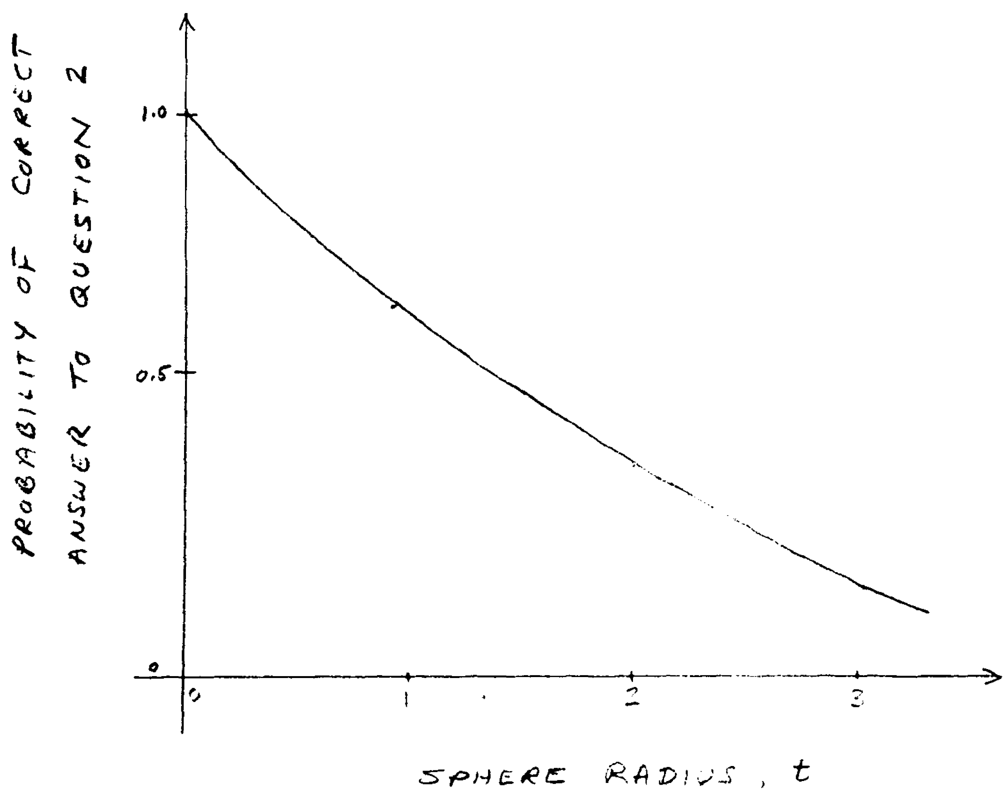


Figure 5. Probability of correct answer vs. sphere radius for modified Algorithm I.

Obviously, storing one data point at each location is not sufficient. It appears that simulation will be required to obtain results for intermediate values of k.

3.3 Implementation

In this report we have ignored, as did Minskv and Papert in their analysis of the exact match problem, the computational complexitv of implementing A_{file} and A_{find}. We believe this is justified on the following grounds. First, the A_{file} part of Algorithm I is an incremental rather than a global algorithm. It examines just one member of the data set at a time, with no control over which it will see next, and without anv subterfuge of storing the data set internally. Second, the A_{find} part of Algorithm I requires a relatively small amount of time and memory overhead to determine L_i for a given query word w and to carry out the search for the data word stored at location L_i that is closest to w. To justifv these assertions, it will be necessarv to consider how Algorithm I might be implemented.

The first problem is to specify the partition of the space of 2^b points into spheres of radius t, or eouiv-alently, to specifv the sphere centers $\{c_i\}$. This sphere-packing problem also occurs in the design of error-correcting codes for the reliable transmission of information through a noisv channel and manv efficient and easily specified codes are known. In the coding context, the sphere centers $\{c_i\}$ are the code words and the set of all centers is called a t-error-correcting code of block length b.

If the code words were chosen in an arbitrary fashion, the odds are that there would be little or no redundant structure, and the code could only be specified by storing all the code words. Fortunately, it happens that very good sphere-packings can be achieved by codes in which the code words form a k-dimensional subspace of the space of k-tuples over the field of two elements. In this case, the code is called a (b,k) linear code over GF(2). An advantage of a linear code is that it can be specified by storing only k linearly independent code words rather than all 2^k code words. A further simplification is obtained by choosing the (b,k) linear code to be cyclic. In this case, the entire code can be specified by storing only one code word. That good sphere-packings can be achieved through the use of cyclic codes is illustrated by the fact that the b=23, t=3 perfect packing can be obtained by using the well known (23,12) triple-error-correcting Golay cyclic code. Hence, specifying the partition of the space of 2^b points into spheres of radius t can be achieved with an insignificant amount of memory overhead.

The second problem is to determine, given w, which sphere w is in, or equivalently, which sphere center c_i is closest to w. This is just the decoding problem for error-correcting codes in which we think of w as a code word plus an error vector and map (decode) w into the nearest code word c_i. If the sphere-packing is perfect, then the query word w falls in one and only one sphere, and nearest-neighbor decoding yields a unique code word c_i, and from c_i a unique location L_i. If the sphere-packing is not perfect, however, and w does not fall within one of the spheres of radius t, the decoding procedure may yield more than one "nearest code word." In this case, it would be necessary to search the contents of more than one location.

While the encoding (specification) of a linear block code is very simple, the decoding process, which is inherently nonlinear regardless of whether or not the code is linear, is in general quite complex. Fortunately, a code with block length on the order of b = 100 is relatively easy to decode, and even for much larger block lengths, certain classes of codes are known that produce relatively good sphere-packings and are easy to decode. Thus, although the decoding of linear block codes is a difficult problem in general, we find that the decoding art has progressed to the point where A_{find} algorithms for data sets of the size considered here could be implemented with relatively modest amounts of time and memory overhead.

In the course of studying the time-memory trade-off in the implementation of the A_{find} part of Algorithm I, and in conjunction with a separate study of the trade-off between decoding time and hardware cost for linear block codes, a new decoding algorithm was found that trades a considerable amount of logical complexity for a small increase in decoding time. This new algorithm is described in a separate report entitled "Decoding by Sequential Code Reduction" by L. D. Rudolph and C. R. P. Hartmann, Systems and Information Science, Syracuse University, 1972.

SECTION 4

ALGORITHM II

In this section, we present the other best-match algorithm studied during the investigation. Algorithm II is quite different from Algorithm I except for the fact that both involve the use of spheres. (We suspect that spheres will play a part in most best-match algorithms.) Given a query word w, there are two fundamental approaches to finding the nearest data word. The first is to compute the distances between w and the data words and then choose a data word that is closest. Algorithm I is a variation of this approach. The second approach is to test w to see if it is a data word; if not, test all words distance one from w; then distance two, etc., until a data word is found. This requires that an exact-match algorithm be used to test each word. Algorithm II is a variation of this second approach.

4.1 Description

The A_{file} part is as follows. Given the data set of 2^a words, store, using the Minsky-Papert hash coding scheme for exact matching, every word in the space of 2^b points that is distance s or less from a data word. Along with each of these words store the corresponding closest data word.

The A_{find} part of Algorithm II, using "hash decoding" and starting at the query word w, performs an ever-expanding search for a word stored in the memory. When it finds one, it reads out the associated data word.

4.2 Analysis

As in the case of Algorithm I, the sphere radius s is a parameter that traces out a time-memory curve for Algorithm II as it varies over the range $0 \leq s \leq b$. At the extreme s = b, Algorithm II becomes a (very inefficient) table look-up procedure.

The following formulas for the memory size and expected number of bit references and the approximations used for actual calculations are derived in Appendix A.

$$M = b2^{a+1} \sum_{i=0}^{s} \binom{b}{i}$$

$$E(N) = 4 \sum_{w=s}^{b} \sum_{i=0}^{w-s} \binom{b}{i} \left\{ \left[\sum_{x=w}^{b} \binom{b}{x} \left(\frac{1}{2}\right)^{b} \right]^{2^a} - \left[\sum_{x=w+1}^{b} \binom{b}{x} \left(\frac{1}{2}\right)^{b} \right]^{2^a} \right\}$$

Time-memory curves using Algorithm II for b=100 and selected values of a are shown in Figures 6 through 10 (at the end of the report).

Comparison of Figures 6 through 10 shows that Algorithm I is best suited for sparse data sets while Algorithm II is best suited for dense data sets. Since data sets in most applications are sparse, our interest in, and analysis of, Algorithm II is rather limited.

SECTION 5

DISCUSSION

The two best-match algorithms described in sections 3 and 4 of this report are admittedly crude. The reader has probably thought of a number of improvements. For instance, in Algorithm I why not iterate the sphere-partition approach, i.e. use some "spheres-within-spheres" scheme, to eliminate the exhaustive search required once L_i has been determined? Or, in Algorithm II, why not conserve memory by storing pointers to words in the data set rather than the data words themselves? We have presented these algorithms in their most primitive forms because the point of the study was to show that there exist ways to achieve a significant speed-up if sufficient memory redundancy is used, not to produce elegant algorithms. At this writing, we have no idea how much memory redundancy is required to achieve a significant speed-up for the best-match problem, but we are convinced that it will be large. Exact-matching and best-matching correspond to error-detection and error-correction respectively, and any coding theorist will attest to the fact that error-correction requires considerably more redundancy than error-detection. The memory redundancies required by the best-match algorithms presented here are very large. There surely exist best-match algorithms that yield the same speed-up for less memory redundancy, but how much less?

Is there an "algorithm-free", Shannon-like critical memory redundancy for a given data set size and data word length above which the number of bit references can be made as close to N^*_{min} as desired by sufficiently complicated A_{file} - A_{find} pairs, and below which it is not possible to do much better than an exhaustive search? This question is of fundamental importance and would provide a natural focus for future research.

In spite of our lack of supporting evidence, we believe that a large decrease in computational complexity can be achieved at the cost of allowing a small probability of error. In real-life applications, the reliability of the data is rarely such that it is reasonable or consistent to require that a question-answering system always give the right answer--assuming that it is possible to define exactly what the "right" answer should be. In our opinion, the reliability-complexity trade-off for such problems as the best-match problem is another important area for future research.

APPENDIX

In this appendix first a result is proved which is applied later on several occasions.

A·1 A Basic Result:

Suppose A_{mxn} is a matrix whose rows are independent random vectors. Elements of each row are mutually independent and take value 0 or 1 each with probability 1/2. Let x be the weight of the i^{th} row i.e. the number of 1's in the i^{th} row. Let $W = \min X_i$. Suppose T is a positive integer and K is the number of rows of A of weight W + T or less. E(Y) denotes the expected value of a random variable Y.

Theorem A·1:

$$E(K) = m2^{-n} \sum_{w=0}^{n} \left\{ \left[\sum_{x=w}^{n} \binom{n}{x} (1/2)^{n} \right]^{m} - \left[\sum_{x=w+1}^{n} \binom{n}{x} (1/2)^{n} \right]^{m} \right\} \sum_{i=0}^{w+T} \binom{n}{i} \qquad (A \cdot 1 \cdot 1)$$

Proof:

Since the elements of a row are statistically independent random Bernoulli variables, each x_i is a binomial variable with parameters n and 1/2. If p(E) denotes probability of the event E, then

$$p(X=x) = \binom{n}{x}(1/2)^n \qquad x = 0, 1, 2, \ldots, n$$

By definition $W = \min_{1 \le i \le m} X_i$. Then for $w=0, 1, 2, \ldots, n$

$$p(W=w) = p\left(\min_{i \le i \le m} X_i = w\right)$$

$$= p\left[\min_{1 \le i \le m} X_i \le w\right] - p\left[\min_{1 \le i \le m} X_i \le w - 1\right]$$

$$= \{1 - p\,[\text{All } X_i\text{'s} > w]\} - \{1 - p(\text{All } X_i\text{'s} > w-1)\}.$$

Using the independence of X_i's, the above expression reduces to

$$p(W=w) = \{p(X_i > w-1)\}^m - \{p(X_i > w)\}^m$$

$$= \left\{\sum_{x=w}^{n} \binom{n}{x}(1/2)^n\right\}^m - \left\{\sum_{x=w+1}^{n} \binom{n}{x}(1/2)^n\right\}^m . \qquad (A\cdot1\cdot2)$$

Next, the probability that a randomly chosen row will have weight $(w+T)$ or less is $\sum_{i=0}^{w+T} \binom{n}{i}(1/2)^n$. Define

$$u_i = \begin{cases} 1 \text{ if the wt. of } i^{th} \text{ row} \le W+T \\ 0 \text{ otherwise} \end{cases} \qquad i = 1,2,\ldots,m.$$

then $K = \sum_{i=1}^{m} U_i$, and $E(K) = E\left[\sum_{i=1}^{m} U_i\right] = \sum_{i=1}^{m} E(U_i) = mE(U_1)$

(A·1·3)

by symmetry. Also

$$E(U_i) = p[U_1=1] = \sum_{w=0}^{n} p \quad [U_1 = 1/W=w] \cdot p(W=w)$$

$$= \sum_{w=0}^{n} \left[\sum_{i=0}^{w+T} \binom{n}{i} (1/2)^n \right] p(W=w).$$

Substituting this value of $E(U_1)$ in (A·1·3) and the value of p(w) given by (A·1·2) the theorem is proved.

Recall that, conventionally $\binom{n}{i} = 0$ for $i > n$. Set

$$a_w = \left\{ \sum_{x=w}^{n} \binom{n}{x} (1/2)^n \right\}^m .$$

Clearly $a_0 = 1$ and $a_{n+1} = 0$. Then

$$m^{-1}2^n \; E(K) = \sum_{w=0}^{n} \{a_w - a_{w+1}\} \sum_{i=0}^{W+T} \binom{n}{i}$$

$$= a_0 \sum_{i=0}^{T} \binom{n}{i} + \sum_{w=1}^{n} a_w \binom{n}{w+T}$$

$$= a_0 \sum_{i=0}^{T} \binom{n}{i} + \sum_{w=1}^{n-T} a_w \binom{n}{w+T}$$

Substituting the value of a_w we obtain

$$m^{-1}2^n \; E(K) = \sum_{i=0}^{T} \binom{n}{i} + \sum_{w=1}^{n-T} \left[\sum_{x=w}^{n} \binom{n}{x} (1/2)^n \right]^m \binom{n}{w+T}$$

$$= \sum_{i=0}^{T} \binom{n}{i} + \sum_{w=1}^{n-T} \left[1 - \sum_{x=0}^{w-1} \binom{n}{x} (1/2)^n \right]^m \binom{n}{w+T}$$

Now we are readv to find an approximation for E(K). Here, basicallv we emplov the following approximation

$$(1 - \frac{x}{m})^m \approx e^{-x} \text{ for large values of } m.$$

To this end we write

$$1 - \sum_{x=0}^{w-1} \binom{n}{x} (1/2)^n = 1 - \frac{m \sum_{x=0}^{w-1} \binom{n}{x} (1/2)^n}{m}$$

and identifv the numerator of the second term on the right by α. Thus

$$m^{-1} 2^n \, E(K) \approx \sum_{i=0}^{T} \binom{n}{i} + \sum_{w=1}^{n-T} \binom{n}{w+T} \exp \{-m \sum_{x=0}^{w-1} \binom{n}{x} (1/2)^n\}. \tag{A·1·4}$$

Another approximation: If n is also large, the binomial probabilitv can be approximated bv a normal distribution, i.e.

$$\sum_{0}^{y} \binom{n}{i} (1/2)^n \approx \Phi\left(\frac{y - n/2}{\sqrt{n/4}}\right)$$

Using the above,

$$E(K) \approx m\left[\Phi \frac{t - n/2}{\sqrt{n/4}} + \sum_{w=1}^{n-t} 2^{-n} \binom{n}{w+t} e^{-m\Phi\left(\frac{w-1-n/2}{\sqrt{n/4}}\right)}\right]$$

In the following discussion we find that the first approximation is more convenient to applv.

A·2 Expected Memory and Bit References for Algorithm I:

We are now ready to obtain the results for the first algorithm. Let S denote the space of 2^b words from which a data set D of 2^a words is chosen at random. S is assumed to be partitioned into $2^b / \sum_{i=0}^{t} \binom{b}{i}$ non-overlapping spheres of Hamming radius t and centers $\{C_i\}$. Let D_i denote the distance from C_i to a nearest data word. Then A_{file} stores at location L_i all data words that are distances $D_i + 2t$ or less from C_i. Our first problem is to find the expected number of data words stored at L_i.

For a given data set D, let K_i denote the number of data words stored at location L_i, i=1, 2,...,m. Because a data set is selected randomly from S, and because of inherent symmetry, all K_i's have identical distributions and therefore identical expectations i.e. $E(K_1) = E(K_2) = \ldots = E(K_m)$.

Without loss of generality we can choose the distinguished sphere center to be C_0, the all-0 b-tuple. The distance between C_0 and any data word is then the Hamming wieght of (number of 1's in) the data word. Let d_0 be the distance C_0 to the nearest data word (i.e. d_0 is the weight of the lowest-weight word in the data set). Then we wish to evaluate the expected number of words in the data set of weight $d_0 + 2t$ or less. However, the result follows immediately by identifying $d_0 \equiv W$, $T \equiv 2t$, $n \equiv b$, $m \equiv 2^a$ and applying Theorem A·1. Thus

E [# of words in the data set of weight $\le d_0 + 2t$]

$$= 2^{-(b-a)} \sum_{d_0=0}^{b} \left\{ \left[\sum_{x=d_0}^{b} \binom{b}{x} (1/2)^b \right]^{2^a} - \left[\sum_{x=d_0+1}^{b} \binom{b}{x} (1/2)^b \right]^{2^a} \right\} \sum_{i=0}^{d_0+2t} \binom{b}{i} \qquad (A\cdot 2\cdot 1)$$

or by (A·1·4)

$$\approx 2^{-(b-a)} \left[\sum_{i=0}^{2t} \binom{b}{i} + \sum_{d_0=1}^{b-2t} \binom{b}{d_0+2t} \exp \left\{ -2^a \sum_{x=0}^{d_0-1} \binom{b}{x} (1/2)^b \right\} \right] . \qquad (A\cdot 2\cdot 2)$$

The total memory given by M = [(number of locations) x(average number of words per location)x(bits per word)] has the following expected value

$$E(M) = \frac{2^b}{\sum_{t=0}^{t} \binom{b}{i}} \cdot b \cdot 2^{-(b-a)} \sum_{d_0=0}^{b} \left\{ \left[\sum_{x=d_0}^{b} \binom{b}{x} \left(\frac{1}{2}\right)^b \right]^{2^a} - \left[\sum_{x=d_0+1}^{b} \binom{b}{x} \left(\frac{1}{2}\right)^b \right]^{2^a} \right\} \sum_{0}^{d_0+2t} \binom{b}{i} \qquad (A\cdot 2\cdot 3)$$

$$\approx \frac{b}{\sum_{i=0}^{t} \binom{b}{i}} \, 2^a \left[\sum_{i=0}^{2t} \binom{b}{i} + \sum_{d_0=1}^{b-2t} \binom{b}{d_0+2t} \exp \left\{ -2^a \sum_{x=0}^{d_0-1} \binom{b}{x} \left(\frac{1}{2}\right)^b \right\} \right] \qquad (A\cdot 2\cdot 4)$$

Further, the expected number of bit references E(N) takes the value

$$E(N) = b \cdot E(K)$$

$$= b2^{-(b-a)} \sum_{d_0}^{b} \left\{ \left[\sum_{x=d_0}^{b} \binom{b}{x} \left(\frac{1}{2}\right)^{b} \right]^{2^a} - \left[\sum_{x=d_0+1}^{b} \binom{b}{x}\left(\frac{1}{2}\right)^{b} \right]^{2^a} \right\} \sum_{0}^{d_0+2t} \binom{b}{i} \qquad (A\cdot 2\cdot 5)$$

$$\approx b2^{-(b-a)} \left[\sum_{i=0}^{2t} \binom{b}{i} + \sum_{d_0=1}^{b-2t} \binom{b}{d_0+2t} \exp\left\{ -2^a \sum_{x=0}^{d_0-1} \binom{b}{x}\left(\frac{1}{2}\right)^{b} \right\} \right] \qquad (A\cdot 2\cdot 6)$$

A·3 Probabilitv of Error for Modified Algorithm I:

In this section we will be interested in the probabilitv of answering question 2 correctlv under slightlv different conditions than described earlier. The sphere packing algorithm as stated above will alwavs find the best match for given search word T in S. Suppose now that in place of a search sphere of "Hamming radius" W + 2t we use a search sphere of radius W, where W, as before, is the random minimum distance of the center from the nearest data word. In other words, at location L_i we store onlv one data word closest to C_i.

An error will be committed if an event of the following nature occurs. See figure FA·1 below. Suppose C is the center of a sphere of radius t and T is a given test word distance L awav from C where $L \leq t$. Assume that

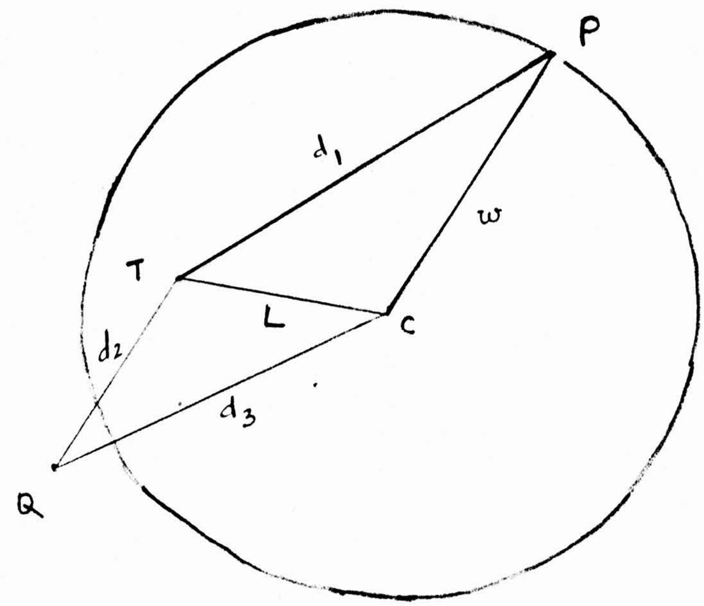


Figure FA·1

the closest data word P is distance w from C. Clearly w is a possible value of the random variable W. We assume that the location corresponding to C in A_{file} will contain only the point P. Suppose P is chosen as a best match for T where P and T are d_1 distance apart. Let Q be another data point which is not in the sphere under consideration and which is at a distance d_2 from T and d_3 from Q, where $d_2 < d_1$. Clearly, we should have chosen Q as best match rather than P. For given W=w and L we will find the probability of this event.

First we will evaluate the probability distribution $p(d_1/L,W)$ of the random variable d_1 which is the distance between the test word T and the nearest data word P when it is given that T is distance L from C. By definition $p(d_1/L,W) = 1/2^b$ (number of points T at a distance d_1 from P/L,w).

To find the number of points, without loss of generality we can assume that the center C is the word (0,0,...,0) and P contains first w ones and remaining (b-w) zeros. Assume that T contains L_1 ones in the first w positions and L_2 ones in the remaining (b-w) positions. Then, L_1 and L_2 satisfy

$$L_1 + L_2 = L$$
$$w - L_1 + L_2 = d_1$$

i.e. $L_2 = (1/2)\ (L + d_1 - w)$ and $L_1 = (1/2)\ (L - d_1 + w)$. Consequently, the number of such points T is

$$\binom{w}{(1/2)\ (L-d_1+w)}\binom{b-w}{(1/2)\ (L+d_1-w)}$$

and $$p(d_1/L,W) = \left(1/2^b\right)\binom{w}{(1/2)(L-d_1+w)}\binom{b-w}{(1/2)(L+d_1-w)} \qquad (A\cdot 3\cdot 1)$$

Next, given d_1, L and w we consider the probability that a point Q, d_3 distance away from C, will be distance

d_2 ($<d_1$) from T. Again without loss of generality, C may be chosen as the all zero word and T as having first L 1's and the remianing (b-L) zeros. Let Q be an arbitrary point such that it has 'a' ones in the first L positions and 'b' ones in the last (b-L) positions. Then

$$a + b = d_3$$
$$w - a + b = d_2$$

or $a = (1/2)\ (d_3 - d_2 + w)$ and $b = (1/2)\ (d_2 + d_3 - w)$.

Thus the totality of such possible points Q is

$$\binom{w}{(1/2)(d_3-d_2+w)}\binom{b-w}{(1/2)(d_2+d_3-w)}$$

However, d_3 can take any value from w to $d_2 + L$, where the upper limit on d_3 is obtained by the triangle inequality and d_2 takes values from D to $d_1 - 1$. Thus, the set of all such points causing an error, denoted by $\mathcal{E}$, contains

$$\sum_{d_2=0}^{d_1-1}\ \sum_{d_3=w}^{d_2+L} \binom{w}{1/2\,(d_3-d_2+w)}\binom{b-w}{1/2\,(d_2+d_3-w)} \qquad \text{points.}$$

Therefore, the probability that a given data point belongs to $\mathcal{E}$, given w and L, is

$$\left(1/2^b\right) \sum_{d_2=0}^{d_1-1}\ \sum_{d_3=w}^{d_2+L} \binom{w}{(1/2)(d_3-d_2+w)}\binom{b-w}{(1/2)(d_2+d_3-w)}$$

Thus, the unconditional probability of a data point causing an error is obtained by multiplying the above probability by the probability of d_1, L and w, and summing over all possible choices of d_1, L and w. This probability, denoted by e say, has the expression

$$e = \left(1/2^b\right) \sum_{w=0}^{b} \sum_{L=0}^{t} \sum_{d_1=0}^{w+L} \sum_{d_2=0}^{d_1-1} \sum_{d_3=w}^{d_2+L} \binom{w}{(1/2)(d_3-d_2+w)} \binom{b-w}{(1/2)(d_2+d_3-w)}$$

$$\left(1/2^b\right) \binom{w}{(1/2)(L-d_1+w)} \binom{b-w}{(1/2)(L+d_1-w)} \cdot p(L)\ p(w)$$

$$(A\cdot 3\cdot 2)$$

where

$$p(L) = \frac{\binom{b}{L}}{\sum_{i=0}^{w} \binom{b}{i}}$$

and p(w) is given by (A·1·2)

The probability that a data point causes an error is e, or does not cause an error is 1-e. Thus probability of no error, which is the same as no data point causes an error, is given by

$$\text{Probability of correct decoding} = (1-e)^{2^{a-1}} \qquad (A\cdot 3\cdot 3)$$

where e is given by (A·3·2).

A·4 The Second Algorithm:

In this algorithm a sphere of radius t is constructed around each data point. Thus the size of the memory, M, equals [(number of sphere)x(two times the number of words per sphere)x(number of bits per word)] or

$$M = b2^{a+1} \sum_{i=0}^{t} \binom{b}{i} \qquad (A\cdot 4\cdot 1)$$

Next we evaluate the expected number of bit references E(N). Assume that the closest data word P is at a given distance w from the test word T. Before a point of the sphere with center P is encountered we will have to compare T with all the points lying within a distance w-t from T.

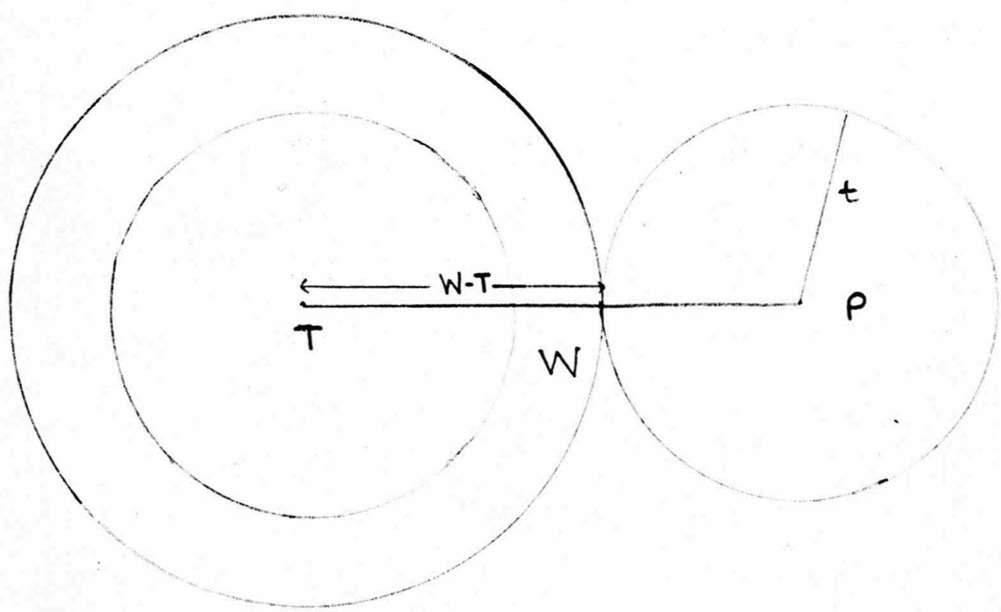


Figure FA·2

There are $\sum_{i=0}^{w-t} \binom{b}{i}$ such points. However, the minimum distance w is a random variable and follows the distribution obtained in section A·1. Thus

$$E(N) = 4 \sum_{w=t}^{b} \sum_{i=0}^{w-t} \binom{b}{i} p(w) \qquad (A\cdot 4\cdot 2)$$

where p(w) given below is obtained from (A·1·2) for m replaced by 2^a and n by b

$$p(w) = \left\{ \sum_{x=w}^{b} \binom{b}{x} (1/2)^b \right\}^{2^a} - \left\{ \sum_{n=w+1}^{b} \binom{b}{x} (1/2)^b \right\}^{2^a} \qquad (A\cdot 4\cdot 3)$$

A·5 Asymptotic Threshold for Algorithm I

In this section we consider the asymptotic behavior of the expected memory as bits per word, b, approaches infinity. We assume that the collection of data words, 2^a, also increases at a rate determined by the relation $a = br$, for some fixed r, $0 \leq r \leq 1$. First we consider the limiting distribution of W, defined in Section 1 for $m = 2^{br}$, $n = b$ as $b \to \infty$. This limiting distribution plays a crucial role in the later developments:

Consider $0<r<1$. From (A·1·2)

$$P(w) = \sum_{x=0}^{w} p(W=x) = 1 - \left\{ 1 - \sum_{0}^{w} \binom{b}{x} 2^{-b} \right\}^{2^{br}} \qquad (A\cdot 5\cdot 1)$$

For any fixed w, as $b \to \infty \left\{ \sum_{x=0}^{b} \binom{b}{x} 2^{-b} \right\}^{2^{br}} \to 0$ implying that $P(w) \to 1$. Thus we confine our attention to the case $w = b\alpha$ for $0<\alpha<1$ and we will be interested in that value of α for which $P(w)$ changes from 0 to 1. Assume that for each b we can find a $\beta = \beta(b)$ such that $\sum_{x=0}^{b\alpha} \binom{b}{x} = 2^{b\beta}$.

Then, from (A·5·1) and the above equality after replacing w by $b\alpha$ and taking the natural logarithm we obtain,

$$\ln[1-P(b\alpha)] = 2^{br} \ln \{1 - 2^{-b(1-\beta)}\}.$$

Expanding the rhs for $0\leq\beta<1$,

$$\lim_{b\to\infty} \ln[1-P(b\alpha)] = \lim_{b\to\infty} 2^{br} \{-2^{-b(1-\beta)} - \frac{2^{-2b(1-\beta)}}{2} - \frac{2^{-3b(1-\beta)}}{3} \ldots\}$$

$$= \lim_{b\to\infty} \{-2^{b[r+\beta-1]} - \frac{1}{2} 2^{b[r+2\beta-2]} - \frac{1}{3} 2^{b[r+3\beta-3]} \ldots\}$$

$$= \begin{cases} -\infty & > \\ -1 & \text{if } \beta = 1-r. \\ 0 & < \end{cases}$$

Thus

$$\lim_{b\to\infty} P(b\alpha) = \begin{cases} 0 & \text{if} \quad \beta < 1-r \\ 1-1/e & \quad\;\; \beta = 1-r \\ 1 & \quad\;\; \beta > 1-r \end{cases}$$

Hence,

<u>Theorem A·5·1</u>: In the limit, the random variable $\left(\frac{W}{b}\right)$ takes value α with probability 1 and all other values in the interval [0,1] with probability 0 where α satisfies the following equation.

$$\sum_{x=0}^{b\alpha} \binom{b}{x} = 2^{(1-r)b} \tag{A·5·2}$$

<u>Remark 1</u>: Let us observe that, in the special case when r=1, 1-r = 0 and therefore $\beta \geq 0$ and

$$\sum_{x=0}^{b\alpha} \binom{b}{x} = 2^{0\cdot b} = 1$$

is satisfied only for $\alpha = 0$, thus implying that W takes value 0 with probability 1 and all other values with probability 0.

At the other extreme r = 0, we have only one point in the data set and the minimum weight in this set is the weight of this one word. Consequently, the distribution will continue to be binomial with increasing value of b, with expected value b/2. Similar argument seems to hold for a very small neighborhood consisting of $0 < r < 1/b$. In what follows, we will restrict r to the range $1/b < r < 1$.

The sum of consecutive binomial coefficients can be approximated bv the entrov function H, which is defined bv the following relation,

$$\sum_{x=0}^{b\alpha} \binom{b}{x} \approx 2^{bH(\alpha)} \qquad (A\cdot5\cdot3)$$

Thus, by (A·5·2)

$$2^{bH(\alpha)} \simeq 2^{b(1-r)}$$

or $\quad H(\alpha) \cong (1-r)$

or $\quad \alpha \cong H^{-1}(1-r) \qquad (A\cdot5\cdot4)$

<u>Remark</u>: Function H^{-1} is not well defined in the full range because H(x) is a 2-1 function. But, in case of the problem under considerations α lies only in the interval (0, 1/2). Thus, in equation (A·5·3) that value of α is chosen which lies in the above interval, giving the uniqueness of α.

By Theorem (A·5·1), an approximation of the type (A·5·3) and using (A·5·4) in (A·2·3), the asymptotic expression for the expected memory for Algorithm I is given by

$$E(M) \approx b\ 2^{b} \cdot 2^{-bH(\frac{t}{b})}\ 2^{-b(1-r)}\ 2^{bH[2\frac{t}{b} + H^{-1}(1-r)]}$$

$$\approx b\ 2^{b[r+H\{2\frac{t}{b} + H^{-1}(1-r)\} - H(\frac{t}{b})]} \qquad (A\cdot5\cdot5)$$

Similarly from (A·2·5)

$$E(N) \approx b\ 2^{b\{H[2\frac{t}{b} + H^{-1}(1-r)] + r-1\}}$$

From the above expression, it can easily be seen that for a fixed b and r the maximum value of E(N) occurs at $t = t_0$ where

$$2^{\frac{t_0}{b}} + H^{-1}(1-r) = \frac{1}{2}$$

or $$t_0 = b(\frac{1}{4} - \frac{1}{2} H^{-1} (1-r)), \qquad (A \cdot 5 \cdot 6)$$

and a sharp decrease is observed in the value at $t = t_0 - 1$.

Using the above result, asymptotically, the expected memory at the threshold point is given by

$$E(M) \approx b2^{b[1+r-H\{1/4 - 1/2\ H^{-1}(1-r)\}]} \qquad (A \cdot 5 \cdot 7)$$

Recall that the minimum and maximum possible memories are respectively given by

$$M_{min} = (b-a)\ 2^a = b(1-r)\ 2^{br}$$

and

$$M_{max} = b2^b$$

Therefore, the "relative logarithmic memory redundancy" is given by

$$R \overset{\text{def}}{=} \lim_{b\to\infty} \frac{\log_2[E(M)/M_{min}]}{\log_2[M_{max}/M_{min}]}$$

$$= \lim_{b\to\infty} \frac{b[1-H\{1/4 - 1/2\ H^{-1}(1-r)]\} - \log_2(1-r)}{b[1-r] - \log_2(1-r)}$$

$$\doteq (1-r)^{-1}[1-H\{1/4 - 1/2\ H^{-1}\ (1-r)\}]$$

(A·5·8)

FIGURE 6

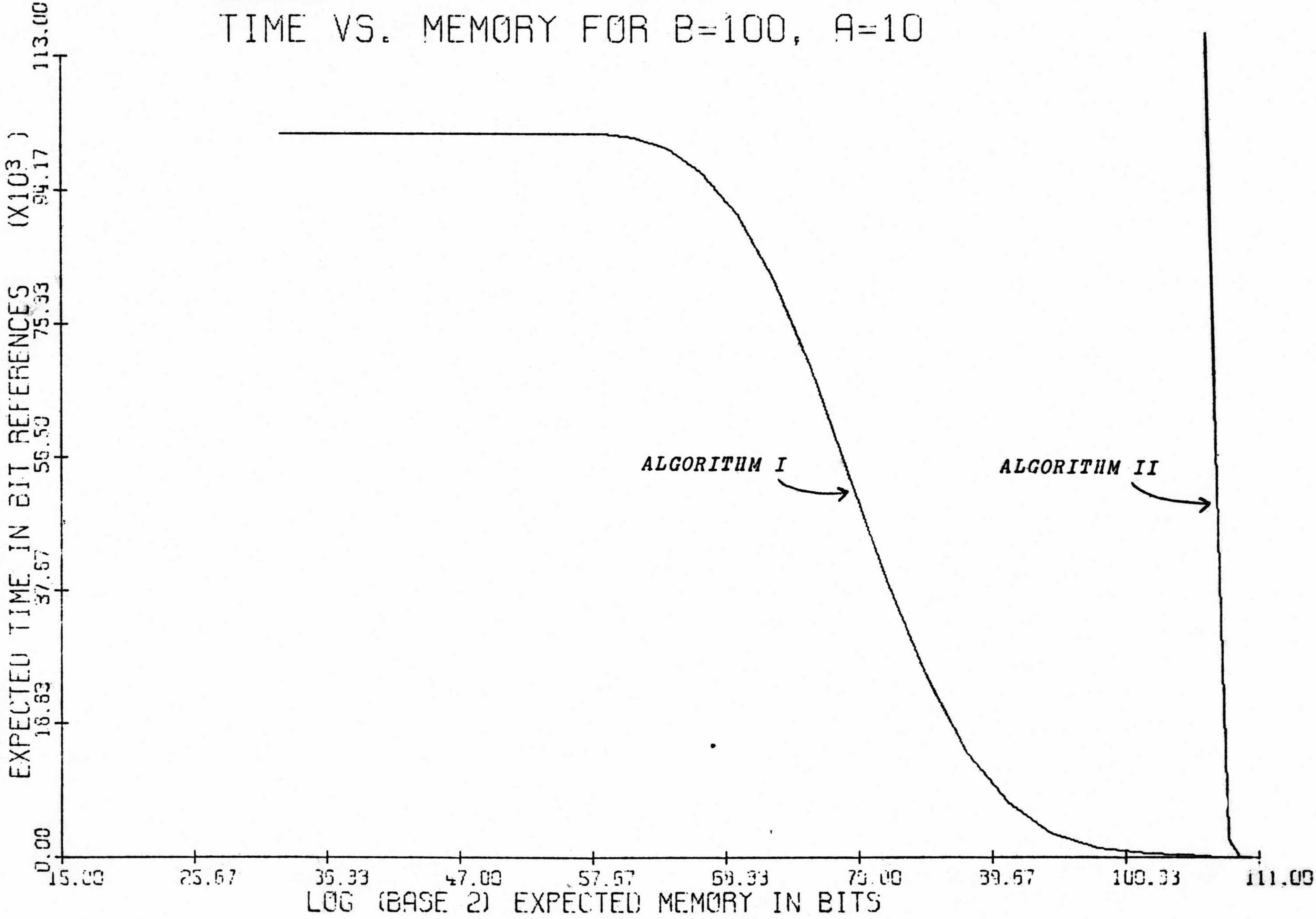

TIME VS. MEMORY FOR B=100, A=10
EXPECTED TIME IN BIT REFERENCES (X10³)
113.00
94.17
75.33
56.50
37.67
18.83
0.00
15.00
25.67
36.33
47.00
57.67
68.33
79.00
89.67
100.33
111.00
LOG (BASE 2) EXPECTED MEMORY IN BITS
ALGORITHM I
ALGORITHM II

FIGURE 7

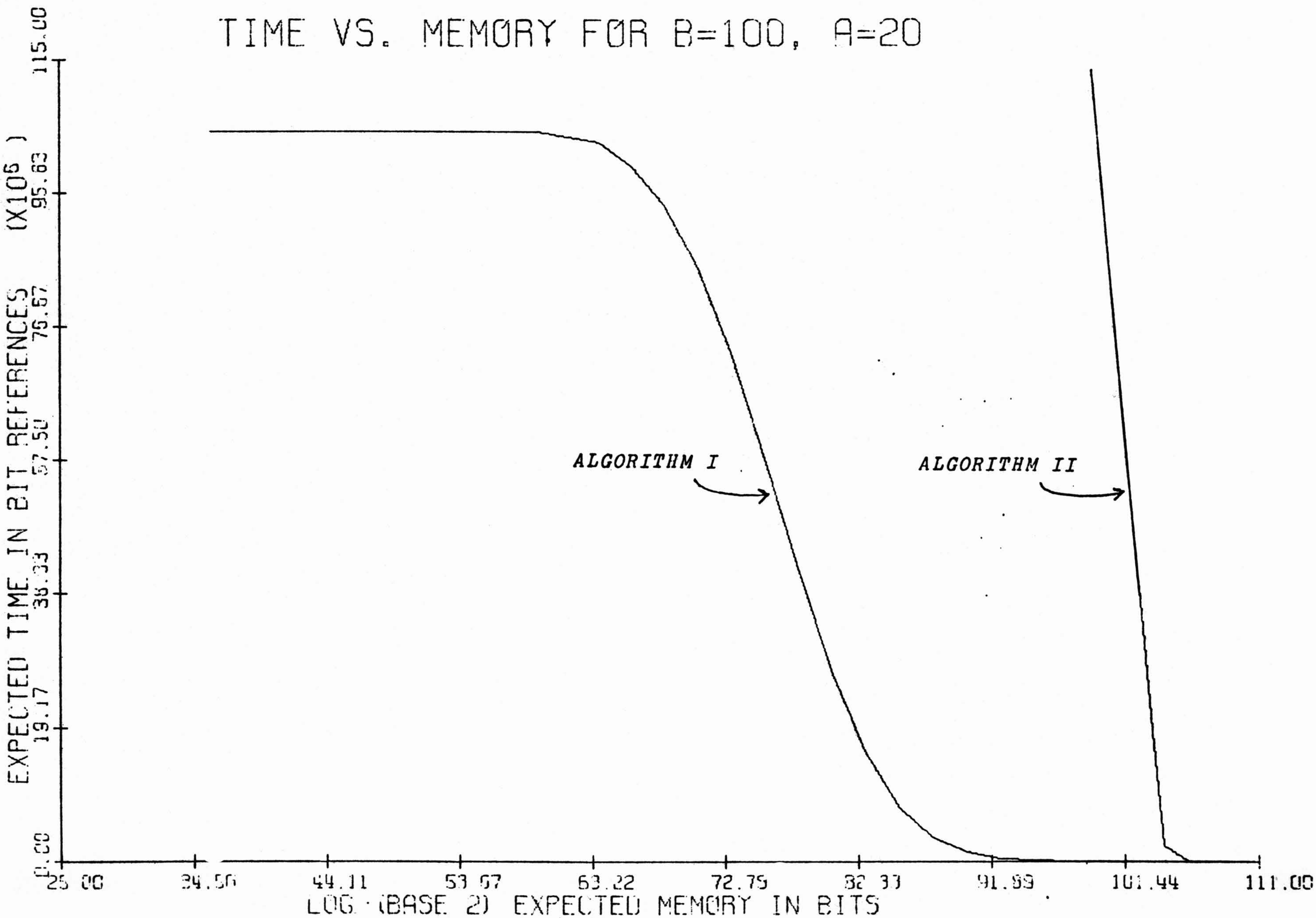
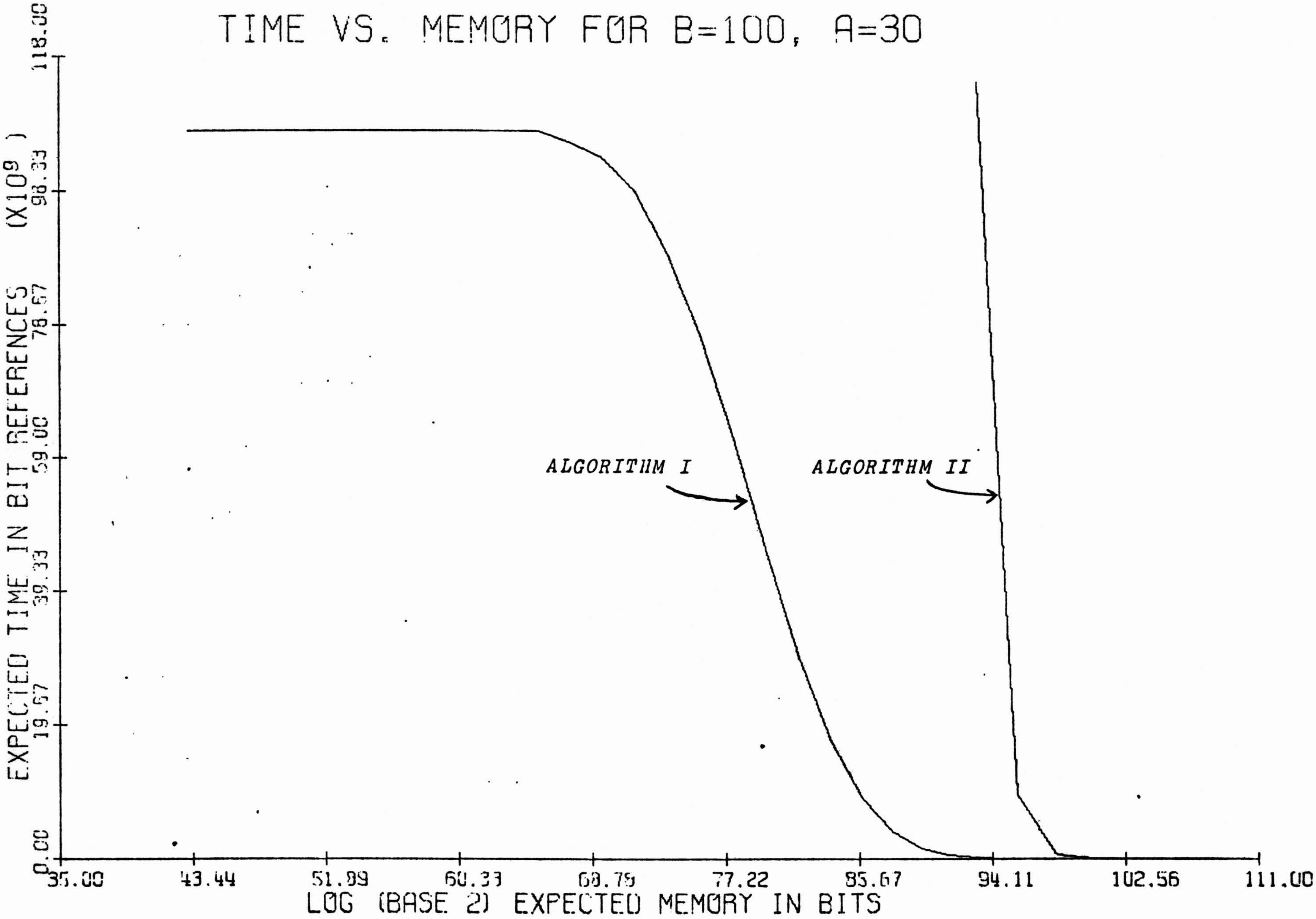
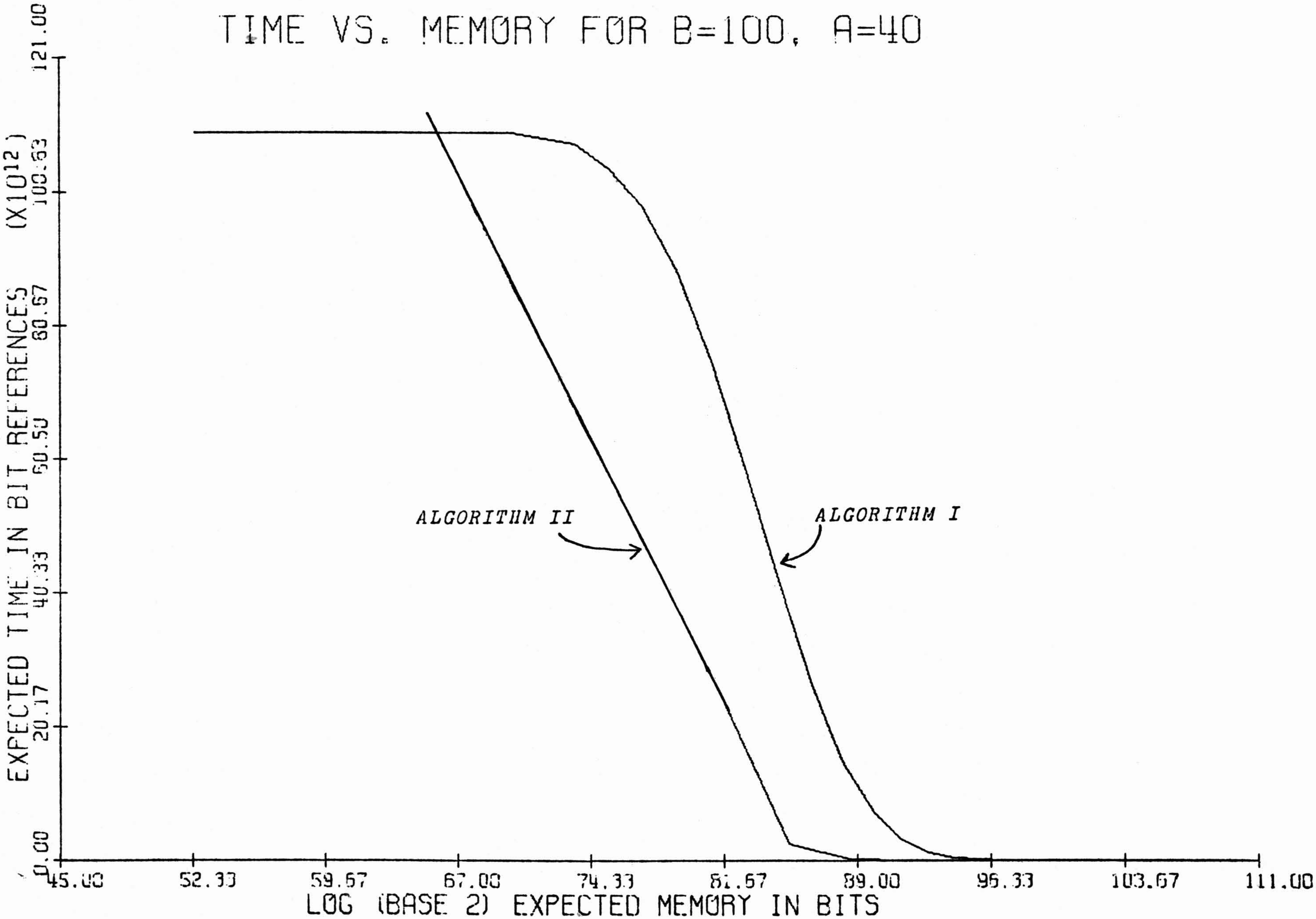

TIME VS. MEMORY FOR B=100, A=20
EXPECTED TIME IN BIT REFERENCES (X10^6)
115.00
95.83
76.67
57.50
38.33
19.17
0.00
25.00
34.56
44.11
53.67
63.22
72.78
82.33
91.89
101.44
111.00
LOG (BASE 2) EXPECTED MEMORY IN BITS
ALGORITHM I
ALGORITHM II

FIGURE 8



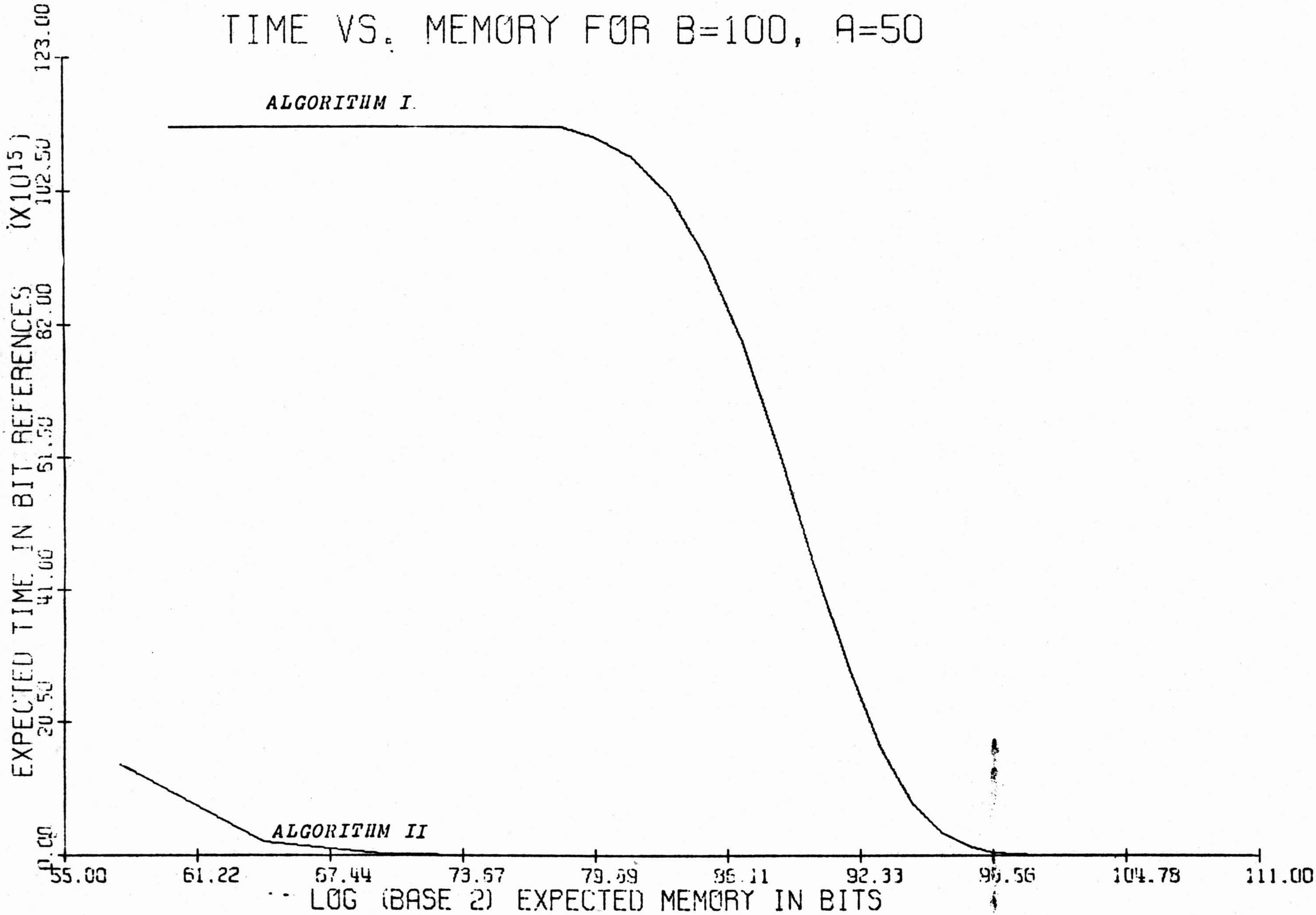
TIME VS. MEMORY FOR B=100, A=30
EXPECTED TIME IN BIT REFERENCES (X10^9)
118.00
98.33
78.67
59.00
39.33
19.67
0.00
ALGORITHM I
ALGORITHM II
35.00
43.44
51.89
60.33
68.78
77.22
85.67
94.11
102.56
111.00
LOG (BASE 2) EXPECTED MEMORY IN BITS

FIGURE 9



TIME VS. MEMORY FOR B=100, A=40
EXPECTED TIME IN BIT REFERENCES (X10^12)
121.00
100.83
80.67
60.50
40.33
20.17
0.00
45.00
52.33
59.67
67.00
74.33
81.67
89.00
96.33
103.67
111.00
LOG (BASE 2) EXPECTED MEMORY IN BITS
ALGORITHM II
ALGORITHM I

FIGURE 10



TIME VS. MEMORY FOR B=100, A=50
ALGORITHM I
ALGORITHM II
EXPECTED TIME IN BIT REFERENCES (X10^15)
123.00
102.50
82.00
41.00
20.50
0.00
55.00
61.22
67.44
73.67
92.33
104.78
111.00
LOG (BASE 2) EXPECTED MEMORY IN BITS