U. Kansas

ASSOCIATIVE RETRIEVAL

by

DYNAMIC TRANSFORMS

UNPUBLISHED PRELIMINARY DATA

GPO PRICE $ _____

CFSTI PRICE(S) $ _____

Hard copy (HC) $1.00

Microfiche (MF) .50

Author:

Dr. Wilburn O. Clark
General Electric Company
Huntsville, Alabama

Presented October 16, 1964, at:

Southeastern Regional Meeting of
Association for Computing Machinery
and Southeastern Simulation Council

Atlanta Americana Motor Hotel
Spring and Carnegie Way
Atlanta, Georgia

[redacted]

ASSOCIATIVE RETRIEVAL BY DYNAMIC TRANSFORMS

In those problems where the number of retrieval operations from memory exceeds the number of storage operations, extra time can be allowed for storage if a sufficient time can be saved by associative retrieval. In this paper a technique is described which uses dynamic transforms to achieve associative retrieval.

The concept of simultaneously searching a set of stored information items for a desired word is generally called either associative retrieval, content addressed, or "key-to-address transform." Several bibliographies are available describing earlier work. To graphically demonstrate the difference in techniques (let us) consider SLIDE 1. Consider the problem of finding that an item is not in the list. To use sequential searching requires N retrieval times. To use binary splitting requires $\log_2 N$. But to use dynamic transforms requires only "a" which is not a function of N.

Various portions of some problems can be formulated in terms of a set of records, each of which may have several identifying items called "keys" associated with it, as illustrated in Figure 1.2.1. To facilitate record manipulation the content of a record is represented by the address to the actual record where the information relevant to a particular application is stored.

The realization of an associative memory organization is either of the special circuitry type usually called "content addressable memory" (CAM) or of the interpretive program type which may be called "algorithmic associative memory" (AAM). Only the AAM type can be realized on a general purpose computer equipped with a standard "random access memory" (RAM).

In AAM systems parallel comparisons with all keys stored is achieved by using the same searching algorithm. The reference key is the input data to a program segment which computes an address. The content of the computed address directly (or indirectly) leads to all of the stored records which are associated with the reference key. Thus, it is possible for one key to refer to multiple records and for one record to be identified by many keys. The algorithm for transforming a key into an address is called the "mapping algorithm."

Let us have the next Slide (Figure 1.2.2). To remove the constraint of physical adjacency implied in dimension arrays each member of the list can contain the address, called a "link", to the next member. A list constructed with its members linked together is called a "linked list."

The top member (i.e., the member with no predecessor) is the "head" of the list and the address stored there is called the "head link." The terminology suggests an analogy between a linked list and a log chain. Assuming one

list member could be stored per memory cell, the memory cells are linked together like a chain. The relationship of the head link to the succeeding cells in the chain is illustrated in Figure 1.2.2. Note that this is like LISP. Although storage is very fast in a simple list system, retrieval requires a maximum of N list step-and-compare operations, where N is the number of keys stored. One unit of retrieval time is taken to be the time needed to extract an element from a RAM, compare it with the reference key, and perform any necessary list operations. By arranging the list of keys in ascending (or descending) order the storage is only slightly complicated but retrieval time is improved with binary splitting to a maximum of $\log_2 N$ retrieval times.

By introducing a transformation function as the central part of an algorithmic associative memory, a more direct realization is possible. Let us assume that the machine representation of each of the N keys is a positive integer chosen from a larger set of positive integers 1 to M, where M is the maximum size of any one key. Then if M is smaller than the maximum available memory working area (IA), a one-to-one linear transformation can be made from the reference key to a memory location in IA which contains a "tag" to identify which member of a simple list contains the key. In practice M is much larger than the tag memory space (IA) into which the keys are mapped.

First a nonlinear squeezing function is applied to the key, generating a new key called (key)', in order to reduce the magnitude as much as possible without causing a loss of information. Next a linear transform of the form.

IA(key=(tag width)*(key)' +(threshold of IA) (2.2.3) is performed, where "tag width" is the number of characters used for a tag in variable word length computers and "threshold of IA" is the lowest address in the IA block.

In fixed word length machines the tag width is 1. Usually the threshold of IA will be the first address above the program area. The significant feature of this transform is the aspect which performs the nonlinear squeezing.

Let us consider the requirements placed on the squeezing function to make it most suited for associative memory applications. Since each of the N keys to be stored associatively comes from a larger class of M integers, the size of each key must in general be reduced to a smaller value (key)' such that no two distinct keys have a common (key)'. This process of reducing the key size unambiguously is called "squeezing", and the set of keys thus uniquely transformed is said to be "mapped." Since the (key)' value is used in the above equation (2.2.3) to compute an address, the maximum size of the tag memory required is proportional to the largest (key'. Thus, minimizing the maximum (key)' conserves memory space.

(continued)

There are many possible functions which will squeeze a set of scattered integers. Typical of these are number system conversions, weighted column transforms, and modulus arithmetic.

In order for the system to be considered associative in the practical sense of the word, the fetching time must be small. This implies that, with a general purpose computer, the calculations necessary for squeezing must be few and simple. The requirement to maintain unambiguity during storage implies that the squeezing function must have at least one parameter to adjust. The number of store operations required to store uniquely all N input keys is taken as the "time" (t) for storage. The ratio of the number of input keys to the tag memory required is defined as the tag efficiency (v). Thus,

$$v = N/IA. \qquad (2.3.1)$$

The ratio of the storage time to the tag efficiency is taken as a rough figure for the cost (c):

$$c = t/v. \qquad (2.3.2)$$

The transform which meets the above requirements and minimizes the cost (c) is considered the best squeezing function.

Of those studied the most promising was the modulus arithmetic transform.

Modulus arithmetic refers to arithmetic operations in number systems manipulating the integer remainder after division. For example, an integer i is said to be congruent to another integer j "modulo" k, denoted by:

$$i = j(\text{mod } k)$$

if the remainder of the division i/k is the same as the remainder of the division j/k.

Certainly the size of a key is reduced if

$$(\text{key})' = \text{key}(\text{mod MM}) \qquad (2.4.2)$$

where MM is less than key. Thus, we have a candidate for a squeezing function. By allowing MM to be a variable, a "dynamic transform function" is available. For identification purposes a mapping algorithm using this transform is referred to as a "variable modulus-transform."

As new keys are added to the list to be associatively stored, the value of the divisor (MM) is changed from one value to another as necessary in order to map all N keys uniquely. A sequence of divisors which will map all possible combinations of N keys out of M positive integers is said to "converge." To illustrate the manner in which the argument of

the variable modulus-transform is changed, an example is given in Figure 2.4.3 which uses a sequence of prime numbers for trial divisors in storing a particular set of 17 keys, where each key is its own record. The sequence of divisors used for this particular example where N=17 and M=99, is shown above a set of columns of residues generated for each input member until an ambiguity resulted disqualifying that trial divisor. Convergence is indicated in Figure 2.4.3 by a solid line tracing the successfully stored keys of the ordered input list. In this example 41 is the first successful divisor and seven trials were required before it was found. Suppose it is desired to retrieve key 47. First (key)' is computed giving 47 mod (41) = 6. From the IA area (not shown) the tag is found to be 16. Thus the sixteenth key is 47.

Evaluation

To determine an optimum between memory required and storage time required a study of divisors was made.

The decision as to what value the divisor should be must be made external to the variable modulus-transform. Once a divisor has been found which will successfully map a given list of N keys, it need not be changed until more keys need to be recorded giving a new value of N. The successful divisor is determined by sequentially trying a set of divisors one at a time until all N keys can be mapped uniquely. In this section criteria are developed for choosing the first divisor so as to minimize the cost of the recording operation.

The results are displayed on the next slide (3.1.18).

Note optimum locus implied for first guess. Now that for second guess?

Similarly the expected space (s_e) is computed by

$$s_e = {}_j\,MM\,{}_j \cdot p(d_j) \qquad (3.2.8)$$

where the same upper limit is determined by the number of trials to get p=1.

The results for several combinations (N,M) are summarized in Figure 3.2.9 as a poly-parameter two-dimensional representation of the expected space, s_e, as a function of the combination (N,M). At each coordinate box located by (N,M) the value of (s_e) is given for each of the sequences tried. Note the increasing trend.

Similarly, the expected cost (c_e) for various combinations (N,M) are displayed in Figure 3.2.10 for each of the sequences tried. As would be expected for any row (i.e., N), the cost increases with increasing M for all ways of choosing sequences; and, similarly for any column (i.e.,M), the cost increases with increasing N. It is significant that no general advantage is obtained from either the "prime way" (i.e., the next divisor is the next prime number) or the "L-way" (i.e., the next divisor is chosen when L is zero or one), where

$$L = M\text{-}MM \cdot K \qquad (3.2.11)$$

as might be expected. For larger values of N skipping every other integer and generating the sequence N+2, N+4, ···M (i.e., skip 2 way), shows a slight advantage over a sequence of successive integers. The "skip 1 way" is a continuous sequence of successors starting with N+1. The difference between the "skip 1 way" and the successors is directly accounted for by the better choice of the first divisor.

(continued)

For worst case considerations (let us) consider the next slide (3.3.1).

Now consider the question of what problem solving capability does such a function give?

Compactly stated, it allows a high level language in Information Retrieval to have a very direct translation. Consequently, a considerable saving in searching time is realized.

To observe some features of the language referred to, let us consider the next slide (4.2.3). Note that hierarchy of operators is allowed. Each source statement is shown with an intermediate output in a Polish string.

The intermediate output is then input to an IR interpreter, as shown in the next slide. The IR system output is shown at the bottom. The output format is the same as the input so that a user can return those statements which will update the system.

Conclusions

The realization of associative memory techniques on conventional digital computers is demonstrated using only a single search cycle which allows for computing an address and binary comparison. A class of dynamic functions which utilize the apriori information available during the recording operation is evaluated. The flexibility of a trade-off between program running time and computer memory is illustrated. A comparison of various associative memory organizations is made by considering applications to the field of information retrieval.

The feasibility of a real-time information processing system is demonstrated. Both the input and output of such a system are assumed to conform to the rules of language specification disclosed. The structure of the syntax allows three types of associative operations: 1. RECORD a given record with a Boolean "OR" expression of associated keys, 2. ERASE a record from association with each one of a Boolean "OR" expression of keys, and 3. RETRIEVE all records associated with a Boolean expression (any logical combination of "AND", "OR", "AND NOT", and parenthetical expressions) of keys.

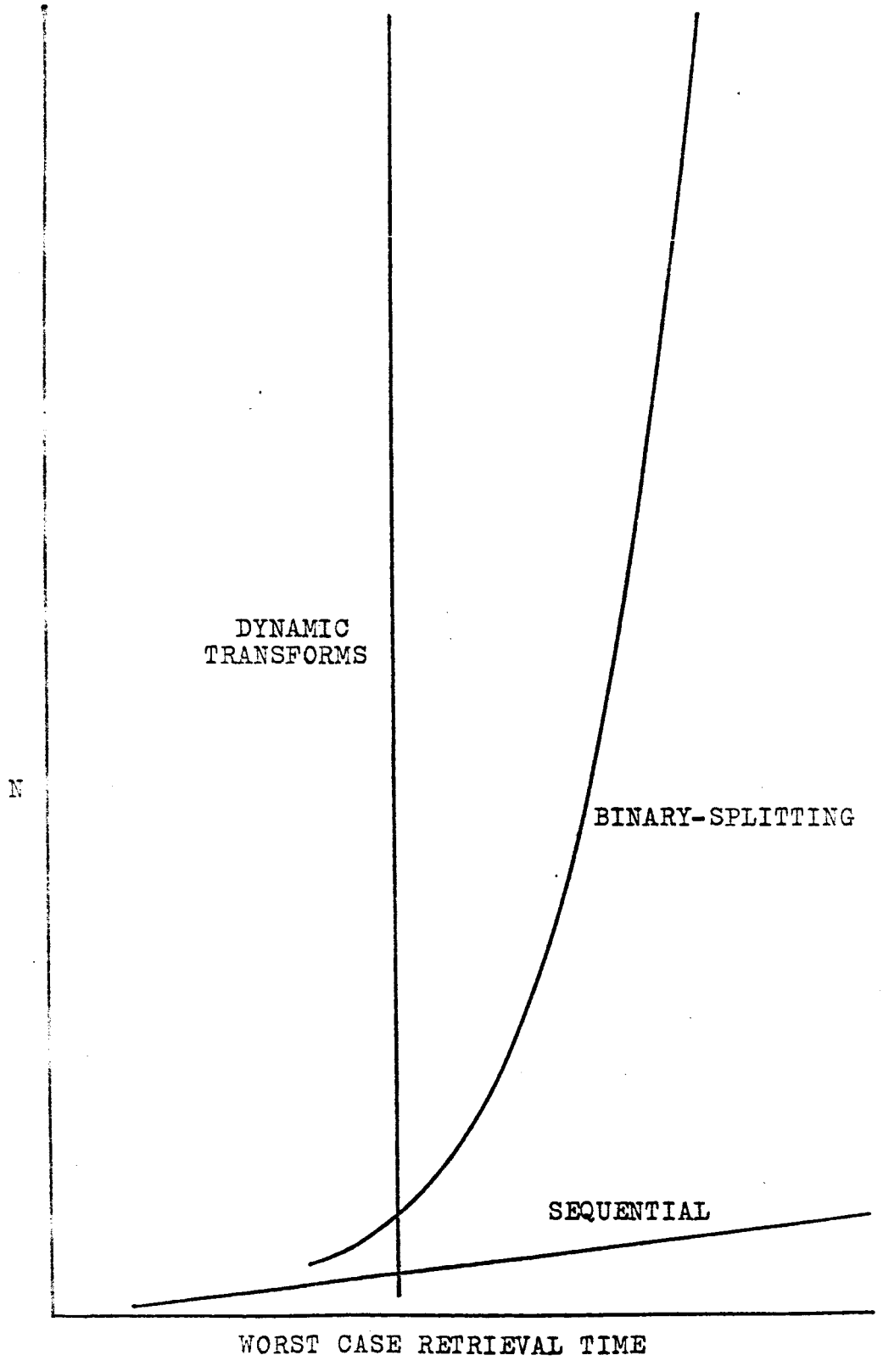
Acknowledgement:

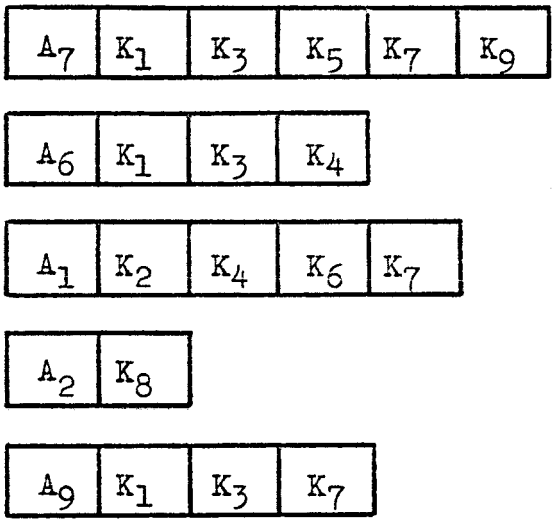
This research was supported by the National Aeronautics and Space Administration, Research Grant NsF-298 to the University of Kansas.

Reference:

"Associative Memory Realization by Dynamic Transformation Functions"

Ph.D. Thesis by Wilburn O. Clark
University of Kansas 1964


DYNAMIC
TRANSFORMS
N
BINARY-SPLITTING
SEQUENTIAL
WORST CASE RETRIEVAL TIME

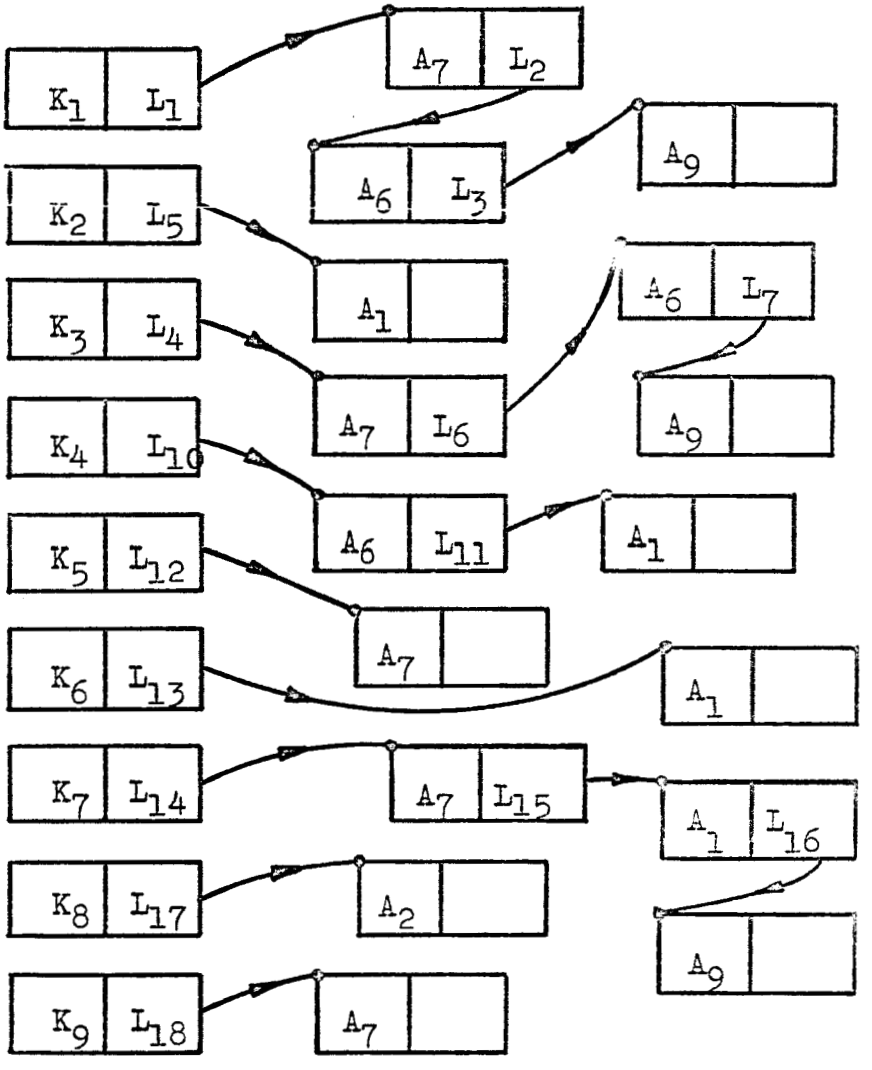


Example of 5 records to be recorded associatively which are designated by subscripted (A's) each followed by the keys (K's) with which it is associated. Each of the 5 groups of adjacent boxes represent collections of information which may be assumed to be available in the application concerned.

Figure 1.2.1

Sequential list of cells containing keys and head links

Distributed cells from available storage with a record and a link to the next cell



The simple list implementation shows reorganization of the example in Figure 1.2.1. Each "L_i" is a link designating the RAM location of the next cell. Diagrammatically a curved arrow points to the next cell in the chain as designated by the link in the preceding cell. Each chain of cells is terminated by a blank label.

Figure 1.2.2

Simple List Implementation

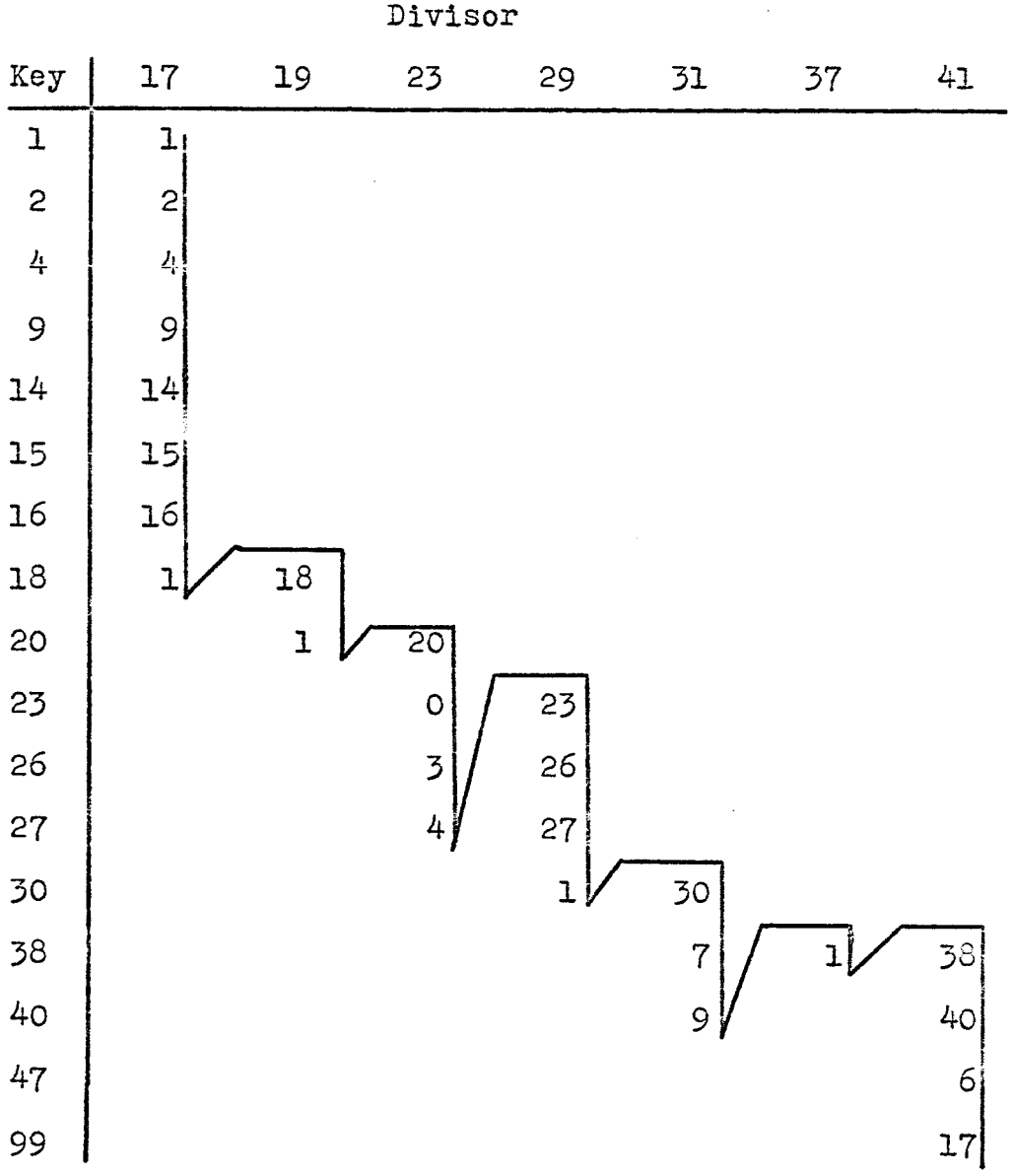
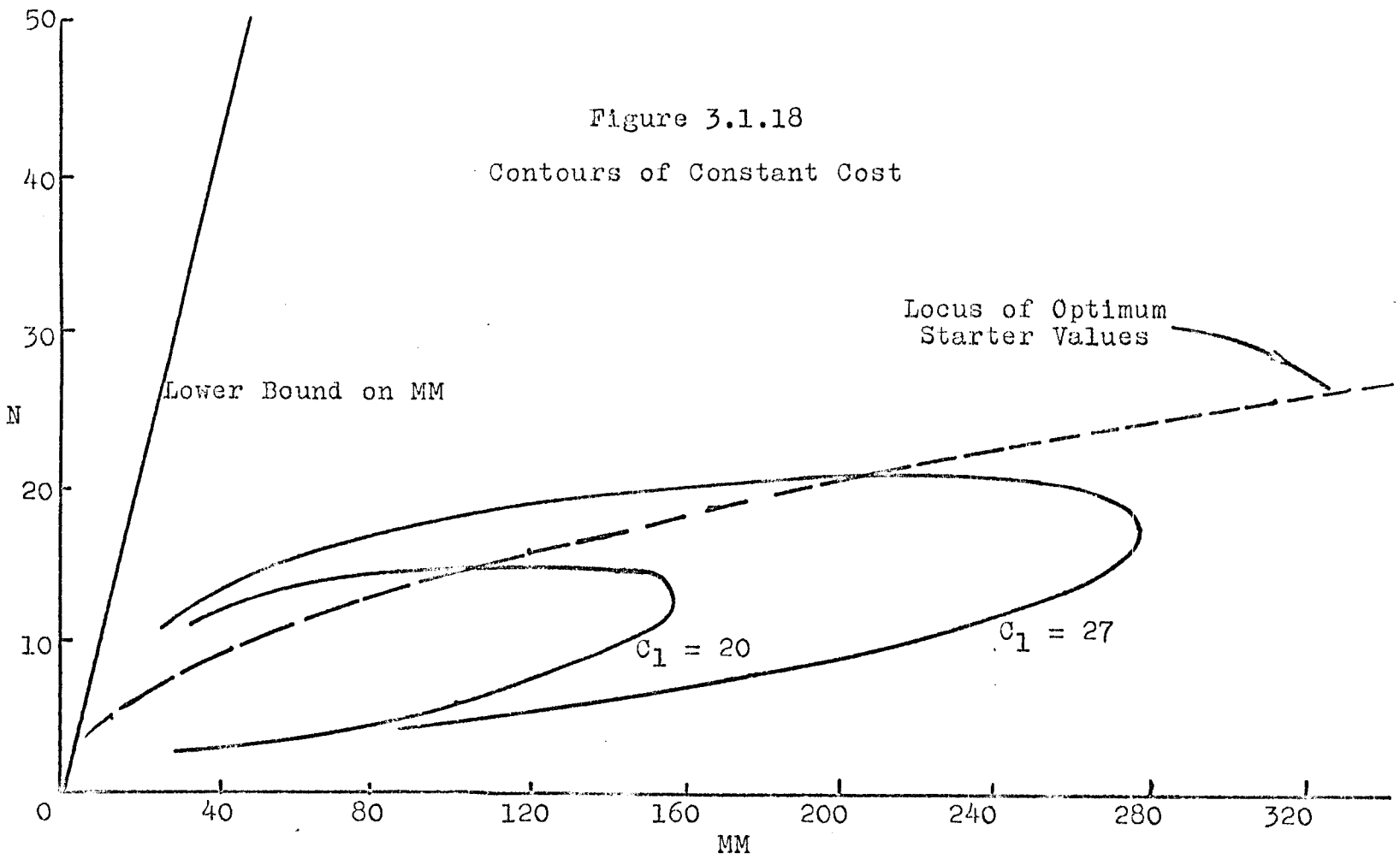


Figure 2.4.3

Prime Divisors Sequence Used

Figure 3.1.18

Contours of Constant Cost



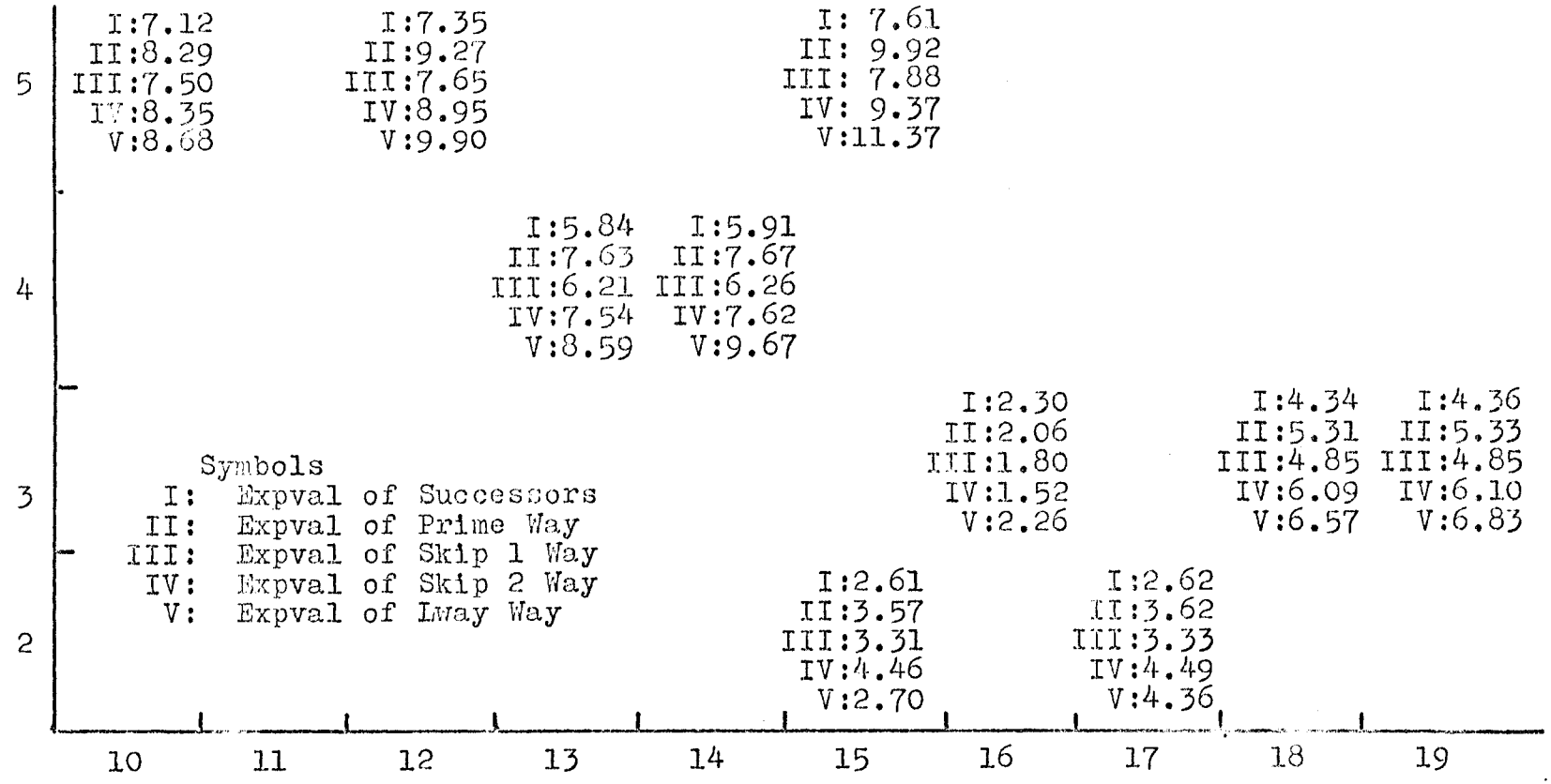


Figure 3.2.9

Expected Value of Memory Required

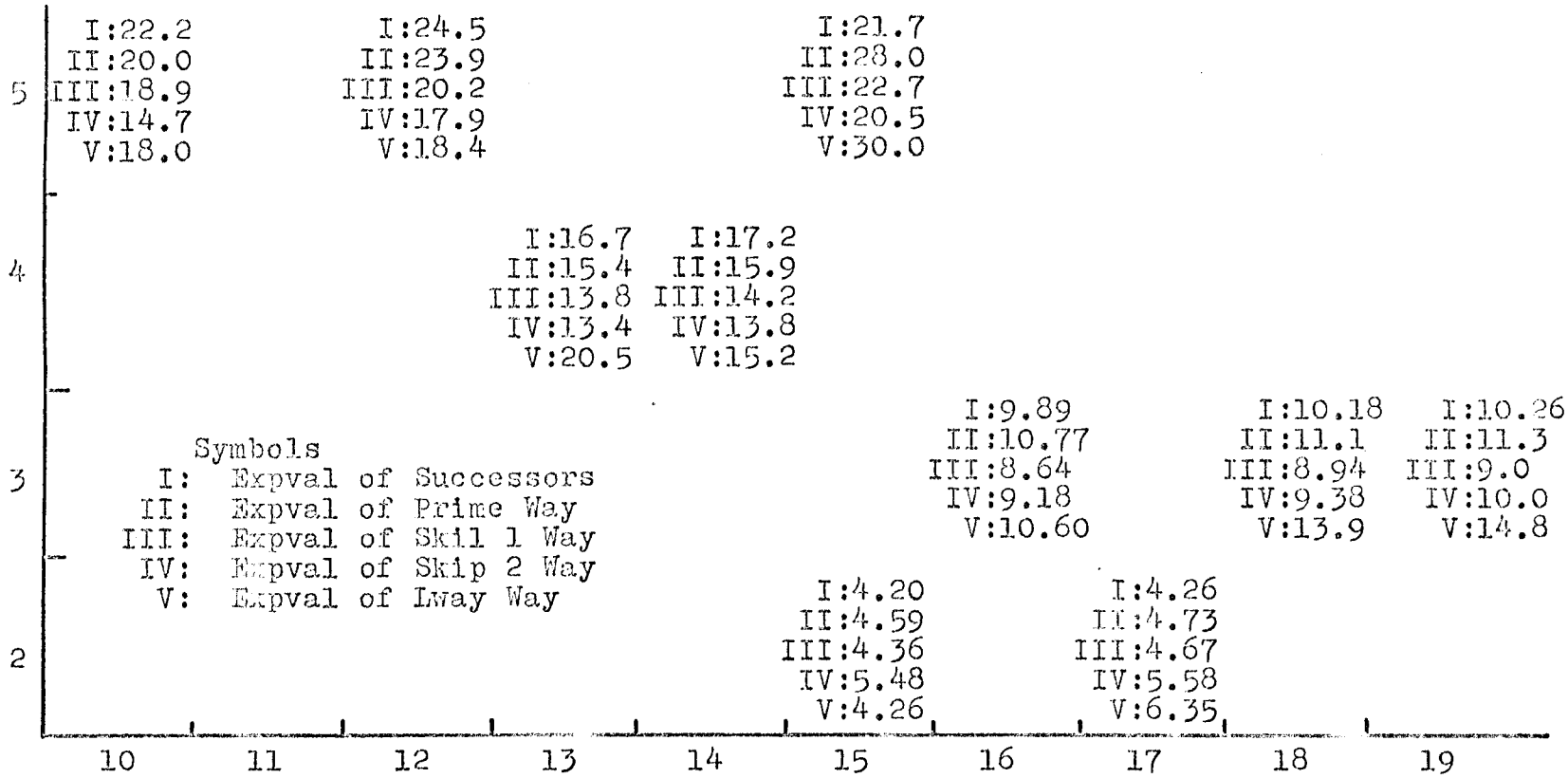


Figure 3.2.10

Expected Value of Cost

Example of deducting the smallest last divisor (ID_1) needed for p(s)=1. Suppose we have the combination (3,19), then the sequence of trial divisors is given below.

Divisor	Key 1	Key 2	Key 3	Key 4	Maximum Conditions	
4	$4a_1+r_1$	$4a_2+r_1$			$a_1,a_2\leq 4$	$r_1\leq 3$
5		$5a_3+r_2$	$5a_4+r_2$		$a_3,a_4\leq 3$	$r_2\leq 4$
6	$6a_5+r_3$		$6a_5+r_3$		$a_5,a_6\leq 3$	$r_3\leq 5$
7	$7a_7+r_4$			$7a_8+r_4$	$a_7,a_8\leq 2$	$r_4\leq 6$
8			$8a_{10}+r_5$	$8a_9+r_5$	$a_9,a_{10}\leq 2$	$r_5\leq 7$
9		$9a_{12}+r_6$		$9a_{11}+r_6$	$a_{11},a_{12}\leq 2$	$r_6\leq 8$
10	$10a_{14}+r_7$	$10a_{13}+r_7$			$a_{13},a_{14}\leq 1$	$r_7\leq 9$

The blank spaces indicate "don't care" relationships.

Constraints:
All numbers below each key must equal it.
Keys must all be distinct and $\leq M$.
All a's and r's must be non-negative.

Figure 3.3.1

Possible Values to be Unmappable

FIGURE 4.2.3
INPUT STATEMENTS WITH POLISH OUTPUT OF TRANSLATOR

```
REC    (REF37,ROBERT,LEDLEY)
 RECO
REF37
OBERT
EDLEY
    ,
REC    (REF37,PROGRAMMING,AND,USING,DIGIT,COMPUTERS)
 RECO
REF37
MMING
    ,
  AND
    ,
USING
    ,
DIGIT
UTERS
    ,
RET    ( BOOK,ROBERT,DIGIT,MEMORY)
 RETR
 BOOK
OBERT
    ,
DIGIT
EMORY
    ,
RET     (A,(ROBERT.    DIGITAL COMPUTERS))
 RETR
    A
OBERT
UTERS
    .
ERA(,MMING)
 ERAS
    ,
MMING
RET    (KEY,ITEM,TERM$DIGIT.DISCRIPTOR)
 RETR
  KEY
 ITEM
 TERM
DIGIT
    S
IPTOR
    .
    ,
```

```
C    INPUT DATA--I.E.--OUTPUT OF TRANSLATOR
 RECO
REF37
OBERT
EDLEY
     ,

 RECO
REF37
MMING
  AND
     ,
USING
     ,
DIGIT
     ,
UTERS
     ,

 RETR
 BOOK
OBERT
DIGIT
     ,
EMORY
     ,
```

```
C    OUTPUT OF SIMULATOR
REC    (REF37, BOOK)
REC    (REF37, BOOK)
REC    (      , BOOK)
```