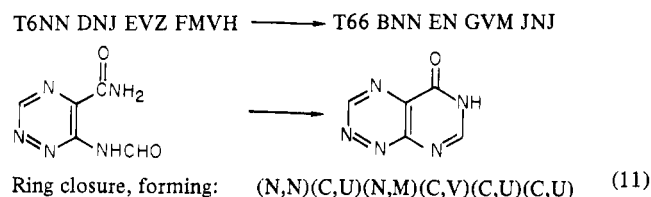
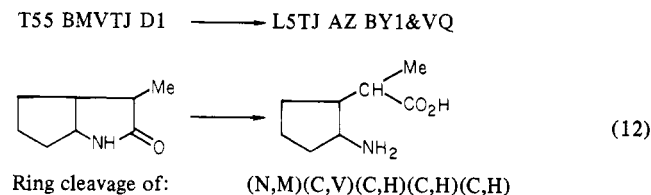


additional changes other than those occurring in the ring systems. This is carried out using the algorithm designed to analyze Type 1 reactions. Therefore, an analysis of the reactions including the acyclic parts of the molecule can be produced as illustrated in eq 11 and 12. These procedures



Substituents
undergoing change: VZ, MVH



Substituents
undergoing change: Z, Y1&VQ

provide analyses for 89% of type 2 reactions or 27% of the total file.

Although the successful percentage of analyses of type 2 reactions is high, the simpler ring changes predominate. Many of the ring system notations are too complex for analysis by

software available to us, but for those not successfully analyzed, an index showing ring size changes alone may be produced.

CONCLUSIONS

It can be seen from this work that it is possible to analyze a substantial proportion of reactions using WLN. The algorithms, which entail manipulation of strings of WLN characters, are all relatively unsophisticated and have been developed in a short time. The programs could be developed to a considerably higher level to deal with a greater percentage of the database.

ACKNOWLEDGMENT

We thank British Library R&D Department for financial support, the Institute for Scientific Information for the data used, I.C.I. Pharmaceuticals Division for software, and G. W. Adamson, E. Hyde, and P. Willett for valuable discussions.

LITERATURE CITED

- (1) J. Valls, and O. Schier, "Chemical Reaction Indexing in Chemical Information Systems", J. E. Ash and E. Hyde, Ed., Ellis Horwood, Chichester, 1975, pp 243-255.
- (2) R. Clinging and M. F. Lynch, "Production of Printed Indexes of Chemical Reactions I. Analysis of Functional Group Interconversions", *J. Chem. Doc.*, **13**, 98-101 (1973).
- (3) R. Clinging and M. F. Lynch, "Production of Printed Indexes of Chemical Reactions. II. Analysis of Reactions Involving Ring Formation, Cleavage, and Interconversion", *J. Chem. Doc.*, **14**, 69-71 (1974).
- (4) Final Report to OSTI on the project "Computer Organisation and Retrieval of Chemical Structure Information", OSTI Report No. 5208, July 1974.
- (5) Final Report to the British Library Research and Development Department on the project, "Development and Assessment of an Automatic System for Analysing Chemical Reactions", British Library R&D Dept. Report No. 5236, July 1975.

An Efficient Design for Chemical Structure Searching. II. The File Organization[†]

LOUIS HODES*

National Institutes of Health, Bethesda, Maryland 20014

ALFRED FELDMAN

Walter Reed Army Institute of Research, Washington, D.C. 20012

Received December 8, 1977

A novel file organization design for substructure search, based on hash coding is described. This organization permits search of only a portion of the file, the portion decreasing with increasing query specificity. The same organization is practical for full structure search, which is routinely used in searching for duplicates when updating the file. Statistics on a sample file of about 20 000 compounds are presented as well as its performance on typical queries.

INTRODUCTION

Part I¹ described a new method for generating molecular fragment screens based on heuristic and information theoretic principles.² It showed how the screens can be weighted and compacted through the use of superimposed coding. This paper continues the account by describing the file organization designed for this system. The delayed publication has awaited the generation of a sample file of over 20 000 compounds in order to present some of the preliminary results.³

Until now there have been two basic methods of designing files to facilitate substructure searching. The fragments or keys can be coded into a bit string so that the computer can

rapidly reject compounds which do not match at least the pattern of the query structure. Alternatively, the fragments or keys can form an index and the corresponding inverted lists of structures can be intersected.

Although inverted file systems are generally considered superior for on-line applications, bit string systems have at least two advantages. The connection tables can be stored with the bit strings for atom-by-atom search of screen hits. Also, the bit strings can be used for full structure search as well as substructure search if one merely substitutes identity match for inclusive match of the query string. This kind of identity search can be very fast by binary search and even faster by hashing.

The advantages of inverted file systems lie in not searching the entire file in responding to a substructure query; only the

[†]Partially presented at the 168th National Meeting of the American Chemical Society, Atlantic City, N.J., Sept. 1974, and the Conference on Computer Graphics, Pattern Recognition, and Data Structure, Los Angeles, Calif., May 1975.

relevant lists need be accessed. These systems are conveniently implemented on disk. The disadvantages are increased storage, loss of full structure search capability, for which a separate search system is generally required, and the fact that connection tables of screen hits must be accessed separately, usually individually, resulting in uncomfortably long elapsed times for some queries.

In his paper,⁴ Lefkowitz deals with some of these issues after a brief tutorial on file structures for information storage and retrieval. The solution of the "dilemma" that he chose is an inverted file system where storage is saved by converting longer lists into bit maps.

He offered an alternative solution which was based on a bit string that was formed by superimposing fragment codes. The design involved a sequential scan of all the bit strings which would presumably be followed by atom-by-atom search of the screen hits as in the inverted file system.

With this background, one easily proceeds to our file organization which is based on a hash code of the superimposed 96-bit string screen described previously.¹ It is a novel design because hash coding has not generally been used for the kind of inclusive matching required in the screening phase of substructure searching.

Hash coding refers to the storage of data at locations derived from the data itself. That is, some transformation or "hash", perhaps simply an abbreviation of the data, produces the address. Generally, more than one item will hash to the same location;⁵ indeed the design will call for dozens of items to be stored at the same location in what is known as a "bucket". The address of the location is called the bucket index.

Each item represents a molecular structure and consists of three parts. First there is the 96-bit superimposed screen code, which is prefixed by the bucket index in the case of indexed sequential file organization. Second, there is a variable-length concise connection table for the structure. Last follows an accession number for retrieval of other data. The connection tables are stored with the screens so that atom-by-atom search on screen hits can be performed immediately and thereby interleaved with bucket accesses.

The organization differs from Lefkowitz's alternate solution since there is no sequential scan of all the screens alone. Hash-coded organization will usually permit scanning only certain portions of the file. This has the additional effect of enriching the proportion of screen hits among compounds in the selected buckets since they are concentrated from the entire file.

Thus, the file organization retains some of the properties of bit string organization, allowing inclusive match for substructure search and exact match for identity search. But by means of a hashed screen code, an accommodation to disk storage has been accomplished in a different way from inverted files. This method allows us to integrate screening and atom-by-atom search more efficiently. Further, it can be shown⁶ that the growth of search time with increased file size is not linear but can approach the square root of the increase in file size. This holds for substructure search, while identity match accesses only one bucket and the time would not necessarily grow with file size. In both sequential bit string and inverted file systems search times grow linearly with file size.

The length of the bucket index determines roughly the number of disk accesses, since at least one disk access will generally be necessary for each bucket scanned. For example, a 12-bit index will result in $2^{12} = 4096$ buckets and require on the order of 4096 disk accesses in the worst case.

It is important to realize here that the query index is computed in the same manner from the query superimposed code as the bucket index is computed from the code of the filed

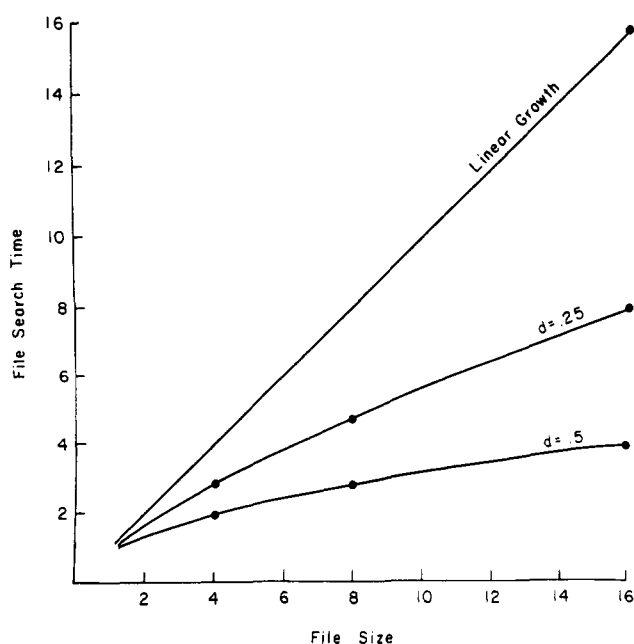


Figure 1. The growth in search time, expressed according to the number of compounds scanned, as the file size grows. Taken from ref 6.

structure. For inclusive matching, only those buckets need be scanned whose indexes have ones where the query index has ones. Thus, each one in the query index cuts in half the number of buckets needed to be scanned. The longer the index is, the greater the possibility of ones and the more saving the search.

The plot in Figure 1 shows the part of the file that must be scanned as a function of file size, the only assumption being that the bucket index grows with file size in such a manner that the average bucket size remains constant. This can be approximated by recomputing the index to be one bit longer for each doubling of the file. Under these conditions the part of the file scanned is related to the ones density of the query index as shown. Thus, the square root relationship holds for highly specific queries where the ones density is 50% or about the same as an average full molecular structure.

At the other extreme, the query is highly nonspecific and there may be no ones at all in the index. In this case the entire file must be scanned. This situation corresponds to a linear growth of search time with file size.

Usually the query will have specificity somewhere between these two hypothetical cases. The example of a query index having 25% ones density is shown in Figure 1.

There is, in this system, an important payoff for making queries more specific and thereby increasing the ones density of the query index. This is in marked contrast to an inverted file system where a more specific query leads to more lists to intersect and, at least in that phase, longer search times.

The discussion will now consider in more detail the means of hashing the screen code to form the bucket index. Then some experience with a sample file will be presented.

COMPUTATION OF THE BUCKET INDEX

The desire to save disk accesses and the desire to restrict the part of the file to be scanned present opposite requirements for the length of the bucket index, as explained earlier. As a compromise, a 12-bit index size was chosen. For a file of 250 000 compounds over 4096 buckets, the average bucket would contain about 60 compounds. Since our sample file showed an average of about 50 words per compound, the average bucket would extend over about 3000 words which is a reasonable core buffer size.

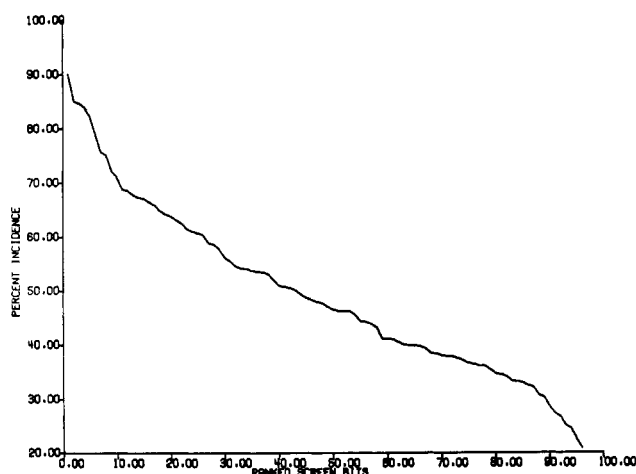


Figure 2. The incidence variation over the bit positions. Replaces the graph A in Figure 10, ref 1.

In generating the hash from the 96-bit screen code, it is helpful to use the characteristics of the screen codes which were generated for the sample file. Then the hash can be used to create the sample file. The bucket size distribution for the sample file will then predict the distribution that can be expected from generating the entire file. This is so because the 96-bit screen code and hence the 12-bit bucket index was highly dependent on relatively high incidence molecular fragments. For fragments of over 0.1% incidence (over 250 compounds) a 10% random sample should yield accurate estimates of incidence in the entire file. Therefore, the 96-bit screen code should have a similar distribution in the sample file as in the entire file. Likewise, the distribution of the bucket indexes should be similar and therefore the bucket sizes.

The critical property of the hash transformation from 96 bits to 12 bits is that of preserving inclusion. This is the requirement that permits substructure search to be performed by inclusive match on bucket indexes as well as on screen codes. The operations that preserve inclusion are sometimes called the monotonic functions. These are abbreviation, splitting the code into parts, and using logical AND or logical OR on the parts, and repeated application of these operations.

The simplest choice of transformation would be an abbreviation, the first 12 bits or any selection of 12 bits. However, this would tend to organize the file according to a relatively small number of structure features. The first plan was to split the code in half, OR the halves, split the result, and AND these halves, yielding a 24-bit string which would then finally be abbreviated to produce the bucket index. In this process, a random structure of ones density 1/2 would yield a bucket index of ones density 9/16.

The main reason for abandoning this method was the evaluation of its prospective performance on weak queries, i.e., queries with low ones density. For example, if a query had a ones density of 0.1, then ORing would almost double it to 0.19, and ANDing would reduce it to 0.036. On the other hand, a simple abbreviation would keep the ones density intact which seems a worthier goal than the unpredictable structural refinements and combinations produced by the more complex hash.

Thus, the first trial generation of the sample file used 12 bit positions, randomly chosen, as the hash transformation. It was soon found that between 5 and 10% of the compounds were located in the all zeros bucket or the all ones bucket. This unbalanced distribution was thought to be more than naturally occurring clustering of chemicals. The cause turned out to be a departure from randomness in setting screen codes.⁷

In other words, an attempt to produce uniformity among the screen bits backfired when it came to getting bucket

| NUMBER OF COMPOUNDS IN BUCKET | NUMBER OF BUCKETS | |
|-------------------------------------|------------------------|------------------------|
| | A) 10,000 COMPOUNDS | B) 20,000 COMPOUNDS |
| 0 | 356 | 31 |
| 1 | 870 | 153 |
| 2 | 1,065 | 370 |
| 3 | 867 | 602 |
| 4 | 530 | 735 |
| 5 | 260 | 718 |
| 6 | 106 | 584 |
| 7 | 37 | 407 |
| 8 | 11.3 | 249 |
| 9 | 3.1 | 135 |
| 10 | .75 | 66 |
| 11 | .17 | 29 |
| 12 | .03 | 12 |
| 13 | | 4.5 |
| 14 | | 1.6 |
| 15 | | .5 |
| 16 | | .15 |
| 17 | | .04 |

Figure 3. The theoretical bucket size distribution for (A) 10000 and (B) 20000 compounds over 4096 buckets assuming 12 statistically independent bits at 50% ones density.

indexes. There were too many relationships among the bit positions. The fragment codes were reassigned on a purely random basis and the screen codes were redone for the sample file. The new distribution of bit incidences is shown in Figure 2. This led to a marked improvement in bucket distribution, but the all ones bucket was still getting between 1 and 2% of the compounds.

Theoretical bucket size distributions were computed under the assumption of a 12-bit index where the bits were statistically independent with 50% probability of being one. Simulations were run with 10000 and 20000 compounds. The results are not perfectly uniform distributions, but as shown in Figure 3. These results seemed to indicate that the distribution of the sample file could be improved by a more rational choice of bit positions for the bucket index.

Instead of the previous random choice of bit positions, those positions were chosen where the ones density was closest to 50%. This led to a bucket size distribution slightly worse than before. Thus, statistical independence became a more important goal than approximating 50% ones density.

In order to find statistically independent bit positions, the 96×96 covariance matrix was computed from the incidences and joint incidences of the screen bits. The sample file that will be briefly discussed was created by minimizing the sum of the pairwise covariances of the bits selected, actually the absolute value of the covariances, by means of an iterative procedure. A more appropriate quantity for optimization has since been discovered. Because of circumstances beyond the control of the authors, an indefinite time may elapse before the more correct procedure can be tested. Nevertheless, it will be presented now and the reader should note that some of the results on the sample file are not well optimized.

There was an error in using the covariance rather than the correlation matrix. This had the effect of emphasizing the high and the low incidence bits which were selected in preference to bits occurring about 50%. When the correlation matrix is used, statistical independence can be maximized by maximizing the determinant of the submatrix of the bits selected. The determinant of a correlation matrix is always less than one except for statistical independence, when it equals one.

A final selection of bits for the bucket index was based on

| NUMBER OF COMPOUNDS IN BUCKET | NUMBER OF BUCKETS | NUMBER OF COMPOUNDS IN BUCKET | NUMBER OF BUCKETS |
|-------------------------------------|----------------------|-------------------------------------|----------------------|
| 0 | 534 | 52-56 | 5 |
| 1 | 651 | 57-61 | 2 |
| 2 | 616 | 62-66 | 4 |
| 3 | 489 | 67-71 | 1 |
| 4 | 399 | 72-76 | 0 |
| 5 | 311 | 77-81 | 1 |
| 6 | 213 | 82-86 | 1 |
| 7 | 173 | | |
| 8 | 123 | 87-134 | 0 |
| 9 | 97 | 135 | 1 |
| 10 | 73 | 136-164 | 0 |
| 11 | 69 | 165 | 1 |
| 12 | 52 | 166-193 | 0 |
| | | 194 | 1 |
| | | 195-214 | 0 |
| | | 215 | 1 |
| 13-15 | 74 | | |
| 16-18 | 49 | | |
| 19-21 | 25 | | |
| 22-24 | 23 | | |
| 25-27 | 13 | | |
| 28-30 | 8 | | |
| 31-33 | 8 | | |
| 34-36 | 10 | | |
| 37-39 | 1 | | |
| 40-42 | 2 | | |
| 43-45 | 2 | | |
| 46-48 | 4 | | |
| 49-51 | 4 | | |

Figure 4. The actual bucket size distribution for the sample file of 20 253 compounds shows many unexpectedly large size buckets when compared with Figure 3b.

the observation that there were many pairs of bits which yielded 50% incidence when either ORed or ANDed. These bits were generally the high incidence bits (for AND) and the low incidence bits (for OR) which were favored with low covariance. At the same time it was noted that such a selection of ANDs and ORs would not necessarily lead to a lowering of ones density for weak queries.⁸ And since it involved 24 bits rather than 12, it was more broadly based for structural refinement.

In this manner the bucket index was determined from the

statistics of a sample file of 23 460 compounds. All pairs of bits whose AND or OR were within 1000 of the 50% value (11 780) were collected. There were 499 such pairs out of 4560 total pairs. The 12 with the lowest covariances were initially chosen. An iterative hill climbing procedure, at each step, replaced a single pair with one which yielded the greatest decrease in covariance summed over all pairs among the 24 bits. After 11 iterations the procedure halted with no further decrease possible by a single replacement. These pairs of bit positions provided the bucket index for the sample file.

THE SAMPLE FILE

The sample of compounds used in this work was organized into a test file. Actually only 20 253 compounds of the 23 460 just mentioned were used since one tape developed a reading problem. The sample file distribution among the buckets is shown in Figure 4. This distribution is quite different from the theoretical distribution shown in Figure 3b for 20 000 compounds. It has a long slowly diminishing tail through the 60 compound per bucket range while theoretically the chance of having a bucket with as many as 20 compounds is vanishingly small. Because of this long tail, the number of empty buckets and the peak or mode value of one compound per bucket are more like the values for 10 000 compounds in Figure 3a than Figure 3b.

There are also four large buckets, each containing more than 100 compounds. These include the all zeros bucket (165 compounds) and the all ones bucket (215 compounds). This latter bucket contains slightly more than 1% of the file.

The reason that the distribution is far from the theoretical one is the existence of relationships among the bit positions. These relationships are due to (1) the fact that most of the original fragment codes from which the screen is composed are represented by more than one bit position and (2) naturally occurring clustering of chemical structures, several having even identical bit patterns. The distribution, therefore, cannot be expected to equal the theoretical one, even by more careful selection of the bucket index as described in the previous section. The achieved distribution can be considered adequate but there still seems to be room for improvement.

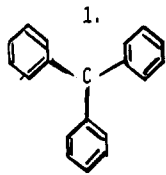
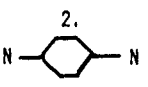
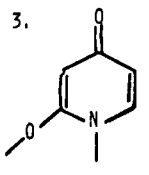
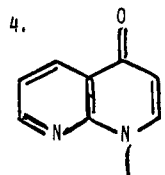
| QUERY STRUCTURE |  |  |  |  |
|----------------------------|---|---|--|---|
| NUMBER OF SCREEN BITS | 18 | 10 | 27 | 48 |
| NUMBER OF INDEX BITS | 2 | 1 | 3 | 8 |
| BUCKETS | 1,024 | 2,048 | 512 | 16 |
| BLOCKS | 970 | 1,835 | 511 | 62 |
| COMPOUNDS SCANNED | 4,986 | 9,325 | 2,996 | 730 |
| SCREENED OUT | 4,769 | 8,684 | 2,858 | 720 |
| PERCENTAGE SCREENOUT | 98.9 | 96.8 | 99.76 | 99.95 |
| ATOM X ATOM HITS/SEARCH | 10/217 | 443/639 | 1/48 | 3/10 |

Figure 5. Some query statistics on the sample file illustrating the capability of the 96-bit screen.

At the time the sample file was being implemented, there was no completely adequate indexed sequential system available at the CDC 3500 installation at WRAIR. Therefore, a new file allocation design was developed for this application. This design was intended for the entire file but has since been displaced by a more suitable indexed sequential system.

Software constraints forced allocation of space to be specified as a number of fixed length blocks. The size of a block must be a multiple of 160 words. In order to provide relatively large blocks without thereby leaving much unused space, not one but five files were allocated. Each successive file had a block size 320 words larger than the block size of the previous file.

The sample file was generated by the same process that would be used to update the file. The first structure to be assigned a given bucket is sent to the file with smallest block size (size 1). When the capacity of a size 1 block is exceeded, the block is released and the contents shifted to the next available block in the size 2 file. The procedure continues until a size 5 block is exceeded. This block is labeled as an overflow block, and the structure is entered in a block of size 1, starting the cycle again. The address of the overflow block is kept as a pointer in the newly occupied block, and in the bucket index lookup table the address is, as usual, the address of the newly occupied block.

In generating the file each entry of a structure required a search for duplicates. This involves a scan for identity match over the 96-bit screen codes of a single bucket followed by an atom-by-atom search on screen hits. For the file of 20 253 compounds there were a total of 329 screen matches, of which 207 were actual duplicates according to atom-by-atom search. Thus, high screenout yielded only 122 unnecessary atom-by-atom searches.

Of the 4096 possible buckets 534 were empty, which agrees with the distribution in Figure 4. Of the remaining 3562 buckets all but 42 were confined to a single block; 31 of the 42 were contained in two blocks, i.e., one overflow block. Seven buckets had two overflow blocks, two buckets had three overflow blocks, one bucket had four overflow blocks, and the all ones bucket had nine overflow blocks. Of course, if the entire file were generated by this method, the blocks would be somewhat larger, with more increments, so the number of overflow blocks would not increase in proportion to the increase in size of the file.

The sample file was extensively tested by performing about 100 substructure queries. These queries generally verified the high screenout of the 96-bit screen as well as the saving in scanning only a fraction of the buckets. Some of the statistics are shown in Figure 5. The number of blocks accessed is also given; it is often less than the number of buckets scanned owing

to the large number of empty buckets. This would not hold for the entire file.

From Figure 5 the reader can notice that most of the screen hits in the first query are false drops. This is due to the relatively paucity of screens for that structure. However, in the second query, fairly low screenout is due mostly to true hits.

REMARKS

The first remark concerns the adaptation of this design to a multiuser multicomputer on-line system such as that being developed by Chemical Abstracts Service for their large file. A straightforward approach would be to designate one of the computers as an executive to receive queries and organize the search. A query for full structure search would get priority over substructure search because it can be accomplished much more quickly. In general, the executive computer would assign a new bucket to a computer when it becomes free and send it just those queries which apply to that bucket. It is not even necessary to have all the buckets accessible to all the computers. Upon hits, the computers may have the capability of replying to the users, or the task of channeling responses can be shared with another executive computer.

The organization of buckets, and therefore responses, by structure may be a serendipitous aspect of this design.

ACKNOWLEDGMENT

Consultation and assistance on several occasions by D. B. Tang are gratefully acknowledged.

REFERENCES AND NOTES

- (1) A. P. Feldman and L. Hodes, "An Efficient Design for Chemical Structure Searching. I. The Screens", *J. Chem. Comput. Sci.*, **15**, 147-152 (1975).
- (2) L. Hodes, "Selection of Descriptors According to Discrimination and Redundancy. Application to Chemical Structure Searching", *J. Chem. Inf. Comput. Sci.*, **16**, 88-93 (1976).
- (3) An additional year's delay was spent in the expectation of better results on the sample file bucket distribution as is described later.
- (4) D. Lefkowitz, "The Large Data Base File Structure Dilemma", *J. Chem. Inf. Comput. Sci.*, **15**, 14-19 (1975).
- (5) Most traditional work on hash coding deals with methods for resolving collisions by relocating items which hash to occupied spaces. This issue becomes meaningless here since provision will be made to store many items at a given location.
- (6) L. Hodes, "A Square Root Algorithm for Inclusive Matching. Application to Chemical Structure Searching", Proceedings of the Conference on Computer Graphics, Pattern Recognition & Data Structure, Los Angeles, Calif., May 1975, pp 375-378.
- (7) Reference 1 describes how bit positions were no longer assigned to fragments when their percentage occupancy exceeded the expected average. This was the only departure from random assignment.
- (8) If the query ones density is p and half the pairs are ORed, the result is ones density $2p - p^2$. The remaining half of the pairs would be ANDed yielding ones density p^2 . Thus the average is $2p/2 = p$ which gets back to the query ones density.