Use of Ring Assemblies in a Ring Perception Algorithm[1a]

W. TODD WIPKE*[1b] and THOMAS M. DYOTT[1c]

Department of Chemistry, Princeton University, Princeton, New Jersey 08540

Received May 31, 1974

The characteristics of several ring-finding algorithms are critically evaluated for memory requirements and execution times. An algorithm called the Welch-Assembly-Gibbs algorithm is presented and shown to yield the fastest ring algorithm in tests on 21 diverse polycyclic structures. This algorithm finds the basis set of rings by Welch's #1 algorithm, then ring assemblies, and then all rings by Gibb's algorithm. Our implementation of Welch's and Gibbs' algorithms uses binary sets and supersets to gain significant improvements in memory requirements and execution times over previous implementations. Flowcharts and Fortran listings are presented.

Recognition of rings within a chemical structure is an important first step in the perception of the chemical nature of the structure, the prediction of its chemical behavior, or the inference of possible syntheses for the structure.[1a,2] The presence of rings within a structure removes degrees of freedom and restricts the available geometries to a small subset of those available to its acyclic counterpart. This restriction may alter the chemical reactivity of functional groups within the molecule through changes in either the strain of the structure or the steric environment of the groups. This fundamental importance of the ring system is reflected in the use of ring systems as a basis for naming and classifying structures and recently by Feldmann and others for substructure searching.[1a]

Ring perception is also important in fields where graphs may represent communication networks, electrical circuits, chains of command, industrial production flow, chemical balances, etc., and because of this importance there has been considerable work directed toward the perception of rings by computer.[2-10]

In contrast to human perception of rings, which is probably a *gestalt* recognition from a graphical representation, computer perception of rings must be defined by an algorithm operating on atoms (nodes) and bonds (edges). The choice of an algorithm depends highly on the precise information desired. One may wish to find all rings or only the smallest rings. Lists of the atoms and bonds in order around the rings may be desired, or unordered sets of the atoms and bonds in each ring may be sufficient. Ideally the ring perception algorithm should be simple and fast, and the speed should be as independent as possible of the size, number, and degree of interconnectivity of the rings. This paper reviews the characteristics of several algorithms and then describes a new algorithm which is faster than any previously reported.

WALKING ALGORITHMS

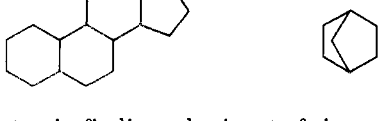
The earliest algorithm, and perhaps the simplest conceptually, used to find rings is the "walking" algorithm.[2,6] The algorithm starts from any atom on the graph and "walks" about on the structure, keeping track of its path and noting any branch points along the way. When the path crosses itself a ring has been found. The path is then shortened and the walk continued from the last branch point. The algorithm is finished when all choices have been taken at all of the branch points.

This method has the advantage that ordered lists of the atoms and bonds around the rings may be obtained directly from the path. The disadvantages are that each ring is found at least twice and generally many times, and the time required for the algorithm increases greatly as the size and degree of interconnectivity of the rings increase. An improvement generally made is to remove from the structure those atoms and bonds not involved in the ring system. This is done by recursively pruning from the graph all primary atoms, until only the ring assemblies,[11] and the atoms and bonds interconnecting them, are left. Choosing the most highly substituted atom as the walk's starting point also improves efficiency by reducing both the number of redundant rings found and the average path length.

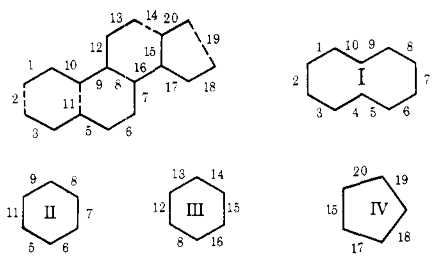
In contrast to a later algorithm,[10] one cannot restrict the length of the walks in an effort to improve the speed, even if one is willing to restrict the size of the rings found. Finding rings remote from the starting point requires a long path regardless of the size of the remote ring.

GRAPH THEORETIC APPROACHES

The development of graph theory led to a different type of algorithm due to the infusion of more mathematical techniques. Matrix manipulation techniques were applied to graphs, represented as adjacency or incidence matrices,[12] to enable one to essentially walk many paths simultaneously. The algorithms developed using these techniques generally consist of at least two sections. The first obtains *a fundamental set* of rings (or *basis set* of rings).[3-5,8] A basis set of rings is a set containing the minimum number of rings which contain every ring atom and bond in the structure. The number of rings in a basis set is the Frerejacque number, that is, the number of cuts required to reduce the structure to an acyclic one (no. of bonds − no. of atoms + 1). For the structures shown below, the basis set of rings contain four rings for the steroid nucleus and two rings for norbornane. Basis sets for a structure, however, are generally non-unique. The second part of these algorithms finds the rings of interest, using the basis set of rings. The rings of interest may be all of the rings, or only those that possess certain characteristics.



The first step in finding a basis set of rings generally involves growing a *spanning tree*. A spanning tree is an acyclic connected graph which contains all of the atoms of the original graph, but, in the case of a cyclic graph, does not contain all of the bonds. Those bonds not contained in the spanning tree are termed *ring closure bonds.* Conceptually a spanning tree can be formed by growing out to the attachments of a starting atom, then growing out from these attachments, and so on, not allowing any rings to be closed, until all of the atoms have been covered. A ring in a basis set can be obtained by proceeding from the ends of a ring closure bond toward the root of the tree until a common atom is found. In this way each ring closure bond, that is, each cut that must be made to reduce the structure to an acyclic one, contributes one ring to the basis set. An example of a spanning tree and the basis set of rings derived from it is shown below.



Note that the rings in the basis set need not be the smallest rings that could form a basis set. The basis set obtained from a spanning tree has the characteristic that every ring in the set contains one bond that is contained in no other ring in the basis set, namely, its ring closure bond.

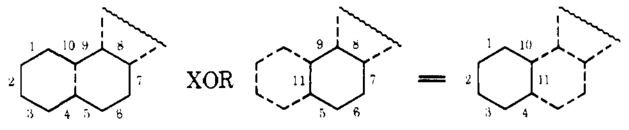
The spanning tree can be grown and the basis set of rings can be collected efficiently using matrix techniques on binary adjacency[4,5] or incidence[3] matrices. It has also been pointed out[7] that the FROM list in the Morgan name[13] of a chemical structure is a spanning tree, and therefore a set of basis rings can be generated rapidly from the Morgan name.[14]

Essentially two different types of algorithms have been used to find the rings of interest from the basis set of rings, or equivalently the set of ring closure bonds. The first involves taking the logical *exclusive or*[15] (XOR) of the bonds of every combination of one to n of the n basis rings, checking each for a new ring. If the bonds are represented as bits in a binary set, the XOR process is very fast since most computers have an XOR machine instruction. The XOR process is illustrated below. The four basis rings shown above can be represented as binary sets where, a 1 in the nth position indicates that the nth bond is included in the ring, as follows:

```
RING I   11111111110000000000
RING II  00001111101000000000
RING III 00000001000111110000
RING IV  00000000000000101111
```

Another ring may be generated by taking the XOR of the bond sets of RING I and RING II:

XOR I, II, 11110000011000000000

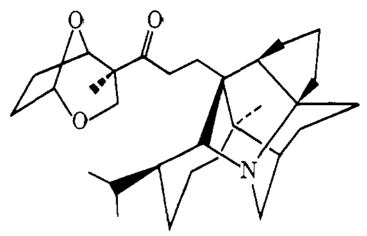


The process of obtaining all rings by taking the XOR of the bond sets of the basis rings has been formulated as an algorithm by Gibbs[9] and can be programmed in two different manners.[8b] The first calculates, stores, and checks for uniqueness the XOR of every combination of m of the basis rings, and then extends it to $m + 1$ of the basis rings by taking the XOR of the $m + 1$ basis ring with all of the previous combinations, checking and storing them as well. The second method does not store the XOR sets unless they actually contain a new ring, and therefore must recalculate some of the intermediate XOR sets many times. With the first method the storage required is $2^{N-1} - 1$ XOR bond sets, if efficient use is made of binary sets of the XOR sets,[16] where N is the number of rings in the basis set. The execution time is roughly proportional to 2^N, since $2^N - N - 1$ XOR sets are generated and examined for rings. With the second method the need for temporary storage is largely eliminated, but the execution time required is roughly proportional to $2^N(N/2 - 1) + 1$.[8b] Gibbs' algorithm has these additional characteristics: it finds all rings and its execution time is dependent on the interconnectivity of the rings and highly dependent on the number of rings in the basis set.

A second type of algorithm, similar to the walking type described earlier, involves growing a tree out from the atoms on each end of the ring closure bond until all rings involving that ring closure bond have been found.[10] It can be shown that in general it is faster to search from two ends at once than from one end. The storage and the amount of execution time required are roughly proportional to $2(b - 1)^{r/2}$ and $2N(b - 1)^{r/2}$, where b is the average branching at the atoms in the graph, r is the maximum ring size searched for, and N is the number of rings in the basis set. This type of algorithm is poorly designed for the task of finding all rings, which would require growing trees over the entire structure. However, in contrast to the walking algorithms, it is possible and beneficial to restrict to maximum ring size allowed, since one can then limit the size to which the trees are grown and still be guaranteed of finding all rings of that size or smaller. If, for example, we defined as "chemically interesting" all rings with six or fewer atoms,[10] the $2N(b - 1)^{r/2}$ term would not be exceedingly large.

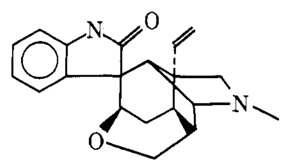
USE OF RING ASSEMBLIES

The speed of Gibbs' algorithm is highly dependent on the number of rings in the basis set, since the logical exclusive or is taken for all combinations of one to n basis rings for a structure with n basis rings. Clearly, in the case of Codaphniphylline shown below, combination of the basis rings in one assembly of rings[11] with any combination of basis rings in the other assembly will not result in any rings that are not obtained simply by a combination of the basis rings within each assembly. Interassembly combination of basis rings is unnecessary. Therefore, by perceiving the assemblies of rings the speed of Gibbs' algorithm (the faster of the two implementations described above) is proportional to



$$\sum_{i=1}^{j} 2^{n_i}$$

instead of 2^N for a structure with j ring assemblies and n_i basis rings in the ith assembly, where $N = \Sigma_{i=1,j} n_i$. At the same time the storage required drops from 2^{N-1} to $2^{n_{\max}-1}$, where $n_{\max}$ is the number of basis rings in the largest assembly. In the same manner the *spiro atom* in Gelsemine, shown below, *divides the ring system into two parts,* each of which can be separately processed by Gibbs' algorithm.

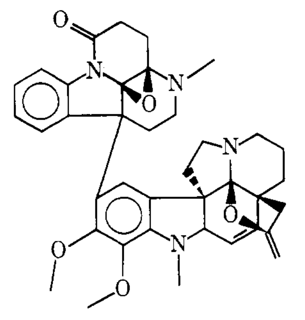


For Codaphniphylline the execution time is reduced from roughly 128 iterations to 32, and the storage required drops from 128 XOR bond sets to 32. While not all complex polycyclic structures have more than one ring assembly, ring assemblies of more than eight basis rings are extremely complex and are rarely encountered in organic chemistry.[17]

Ring assemblies can be found by collecting the basis rings which have bonds in common. A basis ring is selected to be part of the first assembly, and other basis rings that have bonds in common with that assembly (initially just the basis ring chosen) are added to the assembly, until only those basis rings that have no bonds in common with the assembly are left. One of the remaining basis rings is then chosen to begin a new assembly, and the process is repeated until all of the basis rings have been assigned to an assembly. The division of the basis rings into assemblies is very fast since it can be done by means of logical operations on the binary bond sets used to represent the basis rings. The following algorithm finds ring assemblies in a structure which contains the set of basis rings $R = \{r_i\}$, where r_i is the set of bonds comprising the ith basis ring:

```
1. $n \leftarrow 0$
2. If $R = \phi$ then go to step 7
3. Take any $r_i \in R$; $n \leftarrow n + 1$; $A_n \leftarrow \{r_i\}$; $B \leftarrow r_i$;
   $R \leftarrow R - \{r_i\}$
4. COMPLETE ← TRUE
5. For all $r_j \in R$, if $r_j \cap B \neq \phi$, then begin
   COMPLETE ← FALSE; $A_n \leftarrow A_n \cup \{r_j\}$;
   $R \leftarrow R - \{r_j\}$; $B \leftarrow B \cup r_j$ end
6. If COMPLETE then go to step 2; else go to step 4
7. There are $n$ ring assemblies $A_1, A_2, \ldots, A_n$.
```

The increased speed is very dramatic for a structure with more than one ring assembly and a large number of basis rings. Haplophytine, shown below, has two ring assemblies



containing five and six basis rings. By perceiving the ring assemblies the time required for the Gibbs' part of the ring algorithm fell 10.4 sec to 0.08 sec (a factor of over 100). The 0.08 sec includes the time required to perceive the ring assemblies and to determine which basis rings belong to each. At the same time the temporary storage required was reduced from 2048 to 64 XOR bond sets. The improvements in speed and storage requirements can be even greater for structures with more basis rings and/or more ring assemblies.

Table I contains the relative execution times for the structures shown in Figure 1. The time improvement factors listed are the ratio of the times required for the original Gibbs' algorithm and the assembly-Gibbs' algorithm. The results listed indicate that implementation of the assembly concept largely eliminates the very high dependence of the execution time and the storage requirements on the number of rings in the basis set, thereby introducing a degree of linearity into the time and storage requirements

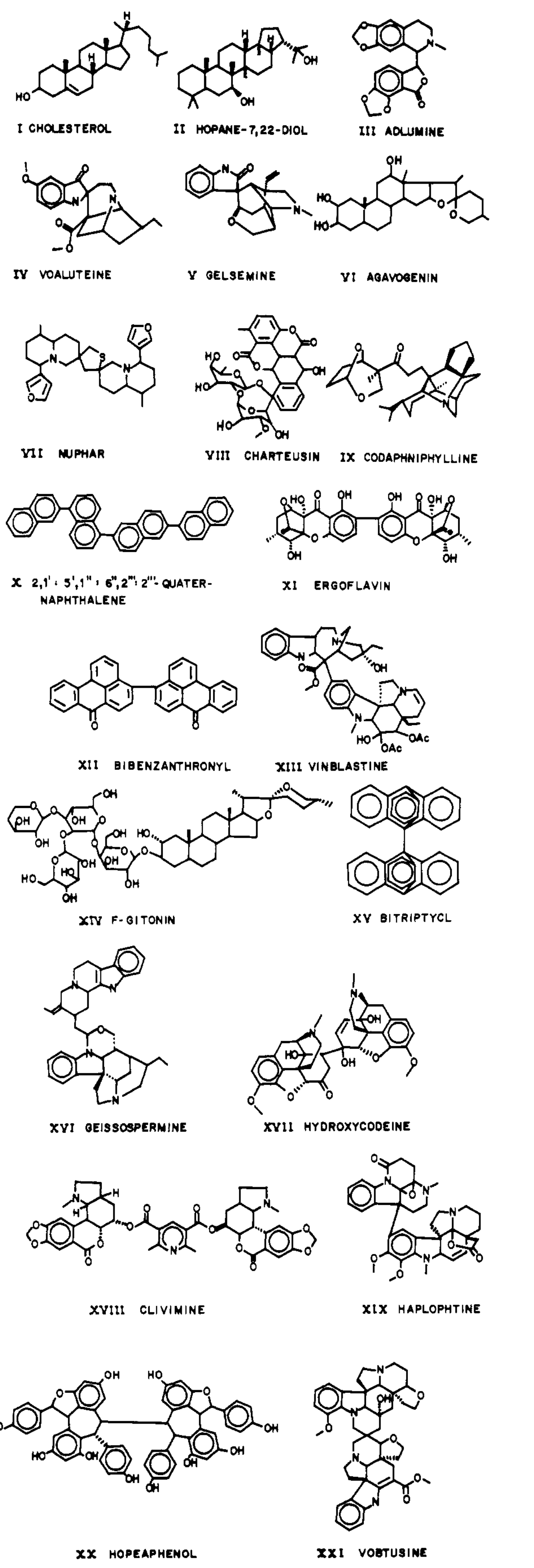


Figure 1. Polycyclic structures used as a basis for comparing the ring algorithms.

Table I. Storage Requirements and Execution Times for the Structures Shown in Figure 1, with and without the Use of Assemblies[a]

			Gibbs		Assembly-Gibbs			
Structure	N	ni	Storage	Time	Storage	Time	Time improvement factor	Total time
I	4	4	16	0.004	16	0.004	1	0.015
II	5	5	32	0.011	32	0.011	1	0.024
III	5	3,2	32	0.010	8	0.006	1	0.020
IV	5	3,2	32	0.021	8	0.009	2	0.022
V	6	4,2	64	0.030	16	0.014	2	0.027
VI	6	5,1	64	0.034	32	0.026	1	0.039
VII	7	2,2,1,1,1	128	0.022	4	0.010	2	0.043
VIII	7	5,1,1	128	0.041	32	0.023	1	0.044
IX	7	5,2	128	0.100	32	0.021	4	0.040
X	8	2,2,2,2	256	0.083	4	0.014	5	0.049
XI	8	4,4	256	0.300	16	0.021	14	0.055
XII	8	4,4	256	0.433	16	0.016	27	0.044
XIII	9	5,4	512	0.633	32	0.024	26	0.057
XIV	10	5,1,1,1,1,1	1024	0.167	32	0.024	7	0.087
XV	10	5,5	1024	0.466	32	0.026	17	0.052
XVI	10	6,4	1024	4.283	64	0.053	80	0.096
XVII	10	5,5	1024	5.684	32	0.054	105	0.092
XVIII	11	5,5,1	2048	4.917	32	0.041	119	0.107
XIX	11	6,5	2048	10.434	64	0.077	135	0.110
XX	12	4,4,1,1,1,1	4096	1.067	16	0.019	56	0.067
XXI	13	7,6	8196	203.217	128	0.150	1354	0.200

[a] Time is given in seconds, storage in words.

Table II. Storage Requirements of Various Algorithms Developed to Find a Basis Set of Rings for a Graph

	Storage required in machine words	
Algorithm	Symbolic	Actual[a]
Welch (as reported)[3]	$2n\left\lceil \frac{m}{w} \right\rceil + 2m + n$	504
Welch (as implemented)	$n\left\lceil \frac{m}{w} \right\rceil + \left\lceil \frac{m}{w} \right\rceil + \left\lceil \frac{n}{w} \right\rceil + n$	220
Gotlieb and Corneil[4]	$3n\left\lceil \frac{n}{w} \right\rceil + 2\left\lceil \frac{n}{w} \right\rceil + n$	506
Paton[5]	$2n\left\lceil \frac{n}{w} \right\rceil + 3n - 2$	502

[a] When $n = 72$, $m = 72$, and $w = 36$.

and largely removing the exponential dependence. The results were obtained on a DEC PDP-10. The program was written in Fortran IV, with a few assembly language routines for efficient manipulation of binary sets, as described in the Appendix.

RING ALGORITHM IMPLEMENTED IN SECS

The current algorithm used in the SECS program[1a] (Simulation and Evaluation of Chemical Synthesis) as part of the molecular perception module could be termed a Welch-Assembly-Gibbs algorithm. A previous version obtained a basis set of rings from the Morgan name. That version is no longer used since the proper implementation of the Morgan naming algorithm requires recognition of aromatic bonds which in turn requires the recognition of rings.[14]

Welch's first algorithm[3] is used to generate a basis set of rings. It should be noted that, while it has been reported that Welch's algorithm requires considerably more storage than later algorithms,[4,5] as implemented here, it uses less than half the storage previously reported, and in fact uses less storage than the reported needs of either the Paton[5] or Gotlieb and Corneil[4] algorithms when the number of bonds is less than or equal to twice the number of atoms. (It is impossible for the number of bonds to be greater than twice the number of atoms unless each atom is on an average of four ring bonds, which is highly unlikely for a chemical structure.) The storage requirements for the three algorithms are shown in Table II, where w is the word length of the computer, n is the number of atoms, and m is the number of bonds. ($\lceil x \rceil$ denotes the smallest integer not less than x.) Figure 2 presents a flowchart of our implementation of Welch's algorithm.

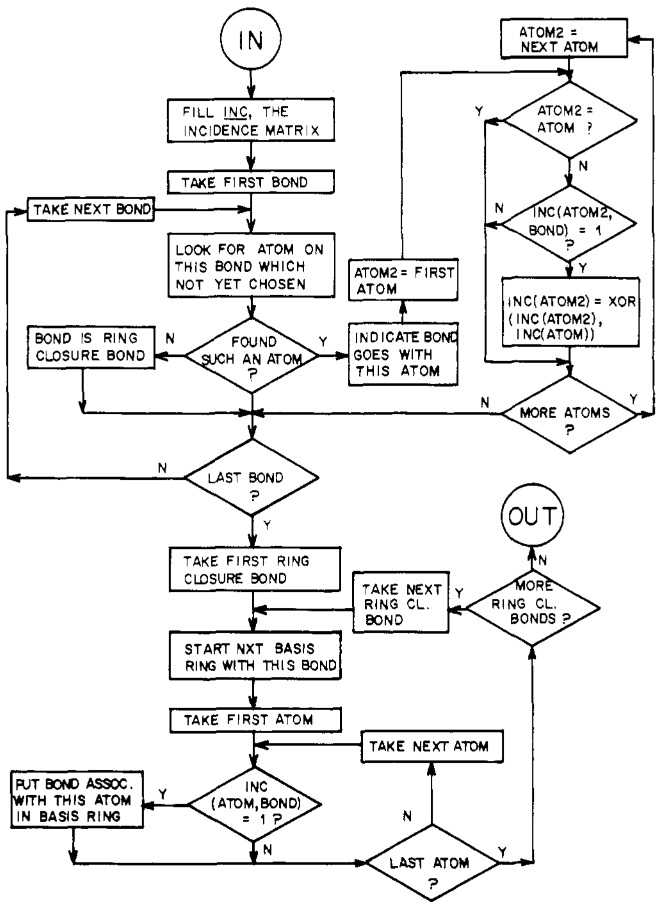

IN
FILL INC, THE INCIDENCE MATRIX
TAKE FIRST BOND
TAKE NEXT BOND
LOOK FOR ATOM ON THIS BOND WHICH NOT YET CHOSEN
FOUND SUCH AN ATOM ?
BOND IS RING CLOSURE BOND
INDICATE BOND GOES WITH THIS ATOM
ATOM2 = FIRST ATOM
ATOM2 = NEXT ATOM
ATOM2 = ATOM ?
INC(ATOM2, BOND) = 1 ?
INC(ATOM2) = XOR (INC(ATOM2), INC(ATOM))
MORE ATOMS ?
LAST BOND ?
TAKE FIRST RING CLOSURE BOND
TAKE NEXT RING CL. BOND
MORE RING CL. BONDS ?
OUT
START NXT BASIS RING WITH THIS BOND
TAKE FIRST ATOM
TAKE NEXT ATOM
INC (ATOM,BOND) = 1 ?
PUT BOND ASSOC. WITH THIS ATOM IN BASIS RING
LAST ATOM ?
Y
N

Figure 2. Flow chart of the implementation of Welch's #1 algorithm for finding a basis set of rings.

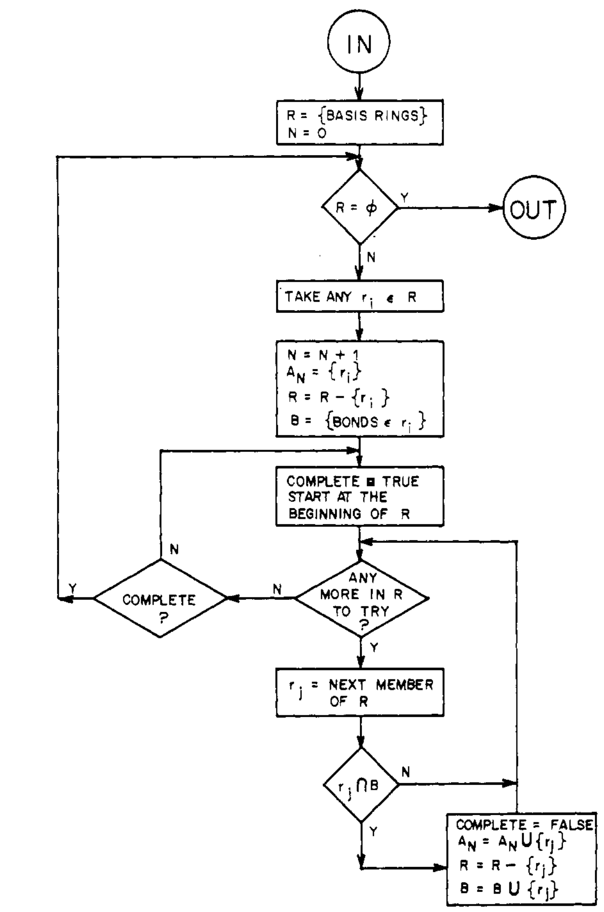


Figure 3. Flowchart of the routine which finds assemblies or clusters of rings.

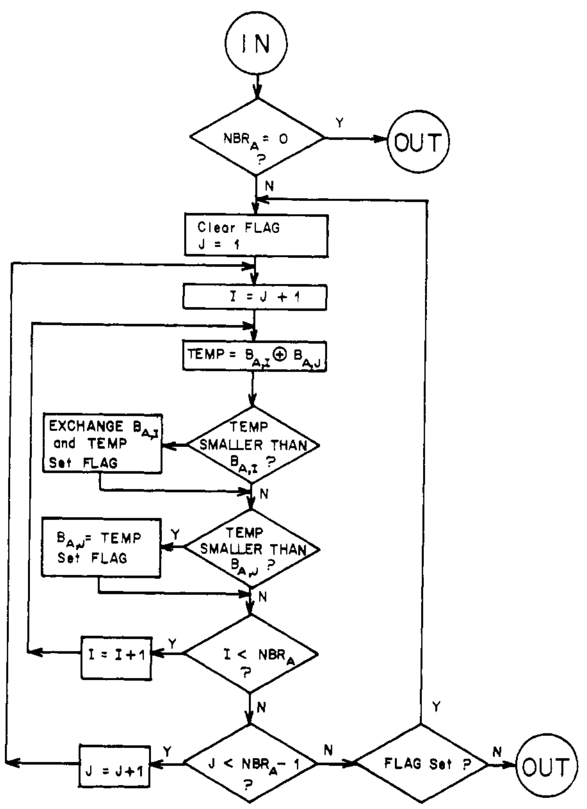

IN
$NBR_A = 0$?
Y
OUT
N
Clear FLAG
J = 1
I = J + 1
TEMP = $B_{A,I} \oplus B_{A,J}$
TEMP SMALLER THAN $B_{A,I}$?
EXCHANGE $B_{A,I}$ and TEMP Set FLAG
N
$B_{A,J}$ = TEMP Set FLAG
Y
TEMP SMALLER THAN $B_{A,J}$?
N
I = I+1
Y
I < NBR_A ?
N
J = J+1
Y
J < $NBR_A - 1$?
N
FLAG Set ?
Y
N
OUT

Figure 5. Flowchart of routine which finds a reduced basis set of rings, given a basis set of rings. NBR_A is the number of basis rings in assembly A.

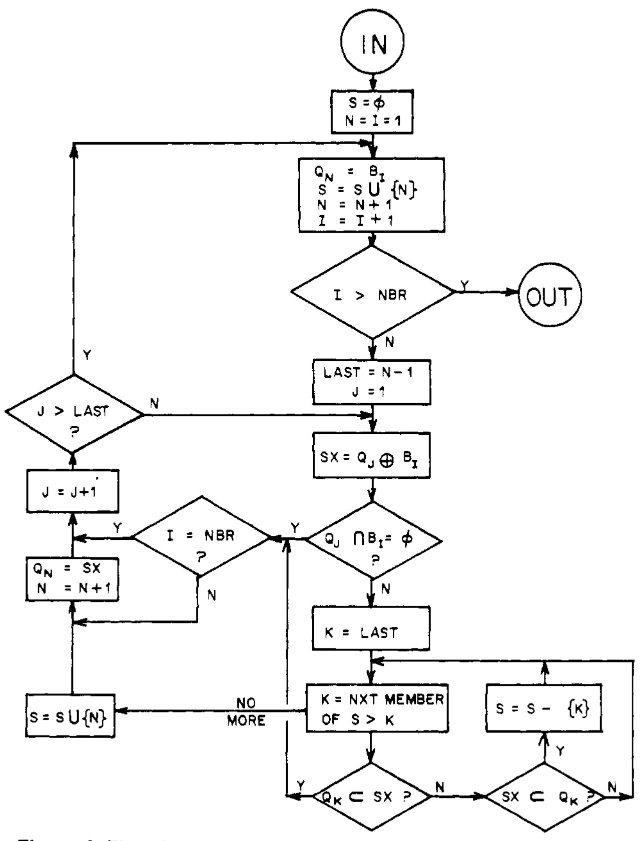


Figure 4. Flowchart of the implementation of Gibbs' algorithm for finding all rings given a basis set of rings. NBR is the number of basis rings in this assembly of rings.

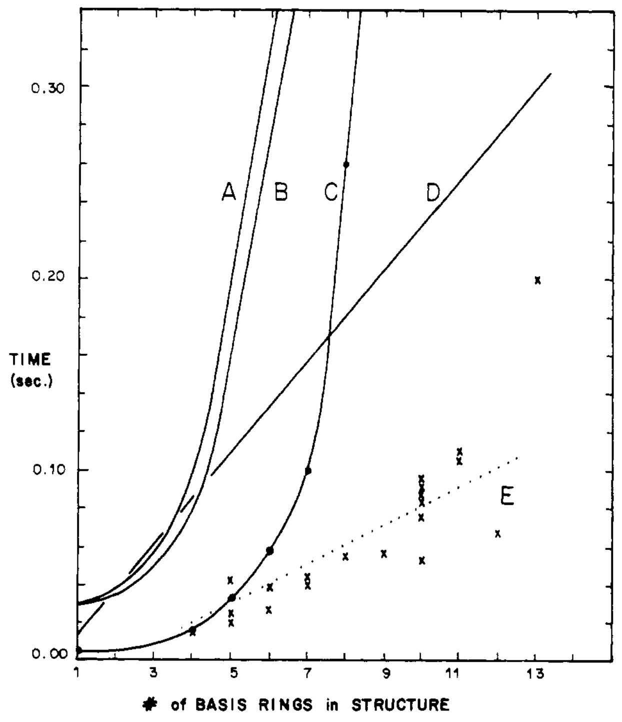


Figure 6. Comparison of the execution speed of various algorithms: (A) Morgan-Gibbs' algorithm, slow implementation[8b] on an IBM 360/75; (B) Morgan-Gibbs' algorithm, fast implementation[8b] on an IBM 360/75; (C) Welch-Gibbs' algorithm on a DEC PDP-10; (D) Paton-"walking" algorithm[10] on a DEC PDP-10. (this algorithm does not find all rings); (E) Welch-assembly-Gibbs' algorithm on a DEC PDP-10 (these times include finding all of the rings and then discarding all but the "chemically interesting" rings, which include all rings up to eight-membered and any other rings in the reduced basis set).

```
C
C
C  RING FINDS ALL OF THE RINGS IN A STRUCTURE
C
C
      SUBROUTINE RING
C
      IMPLICIT INTEGER (A-Z)
C  SETS: QSETS,AND,XOR,NOT,OR,BASISRG,CBASISRG,INC,ATLEFT,
C        BDLEFT, AND THOSE THAT START WITH LETTER S
      INTEGER S,SETLEN
      LOGICAL ON,OFF,SUB,DELFLG,ISM
      COMMON/STORAGE/S(3Ø),CBASISRG(1Ø),QSETS(612)
      COMMON NA,NB,SETLEN,SOCUPA,SOCUPB,AT1(36),AT2(36)
      DIMENSION LABEL(36),SCLRG(2Ø),BASISRG(5Ø),INC(36)
      EQUIVALENCE (INC,QSETS),(LABEL(1),QSETS(1ØØ))
      DATA NBPWD/36/


      SETLEN = 1
      SNULL = Ø
      CALL    ZEROM(BASISRG,5Ø*SETLEN)
      IF(NB.EQ.(NA-1)) RETURN


C   FILL MATRIX INC, THE INCIDENCE MATRIX
   5  CALL ZEROM(INC,36*SETLEN)
      BOND=Ø
  1Ø  BOND=NXM(SOCUPB,BOND)
      IF(BOND.LE.Ø) GO TO 2Ø
      CALL ON(INC(AT1(BOND)),BOND)
      CALL ON(INC(AT2(BOND)),BOND)
      GO TO 1Ø
  2Ø  ATLEFT = SOCUPA
      BDLEFT = SOCUPB
C
C   WELCH'S ALGORITHM #1
      BOND = Ø
  3Ø  BOND = NXM(SOCUPB,BOND)
      IF(BOND.LE.Ø) GO TO 6Ø
C   LOOK FOR AN UNCHOSEN ATOM ON BOND. IF THERE IS NONE, THE BOND
C   IS A RING-CLOSURE BOND.
      ATOM = Ø
  4Ø  ATOM = NXM(ATLEFT,ATOM)
      IF(ATOM.LE.Ø) GO TO 3Ø
      IF(.NOT.ISM(INC(ATOM),BOND)) GO TO 4Ø
C   FOUND AN ATOM ON THIS BOND THAT HAS NOT YET BEEN CHOSEN, SO
C   EXTEND THE SPANNING TREE OUT TO THAT ATOM. THIS IS DONE BY
C   LABELING THE ATOM WITH THE BOND NUMBER.
      LABEL(ATOM)=BOND
C   REMOVE THIS ATOM AND BOND FROM THE SETS OF THOSE UNCHOSEN
      CALL    OFF(ATLEFT,ATOM)
      CALL    OFF(BDLEFT,BOND)
      ATOM2 = Ø
  5Ø  ATOM2 = NXM(SOCUPA,ATOM2)
      IF(ATOM2.LE.Ø) GO TO 3Ø
C   IF THIS IS THE ATOM JUST CHOSEN DO NOTHING
      IF(ATOM2.EQ.ATOM) GO TO 5Ø
C   IF ATOM "INC" TO SAME EDGE XOR THEM
      IF(ISM(INC(ATOM2),BOND)) INC(ATOM2)=XOR(INC(ATOM2),INC(ATOM))
      GO TO 5Ø
C
C   COLLECT BASIS RINGS. THE BONDS LEFT ARE THE RING-CLOSURE BONDS.
  6Ø  NBR=Ø
      BOND=Ø
  7Ø  BOND=NXM(BDLEFT,BOND)
      IF(BOND.LE.Ø) GO TO 9Ø
C   START A NEW BASIS RING WITH THIS RING-CLOSURE BOND
      NBR=NBR+1
      CALL ON(BASISRG(NBR),BOND)
      ATOM = Ø
  8Ø  ATOM = NXM(SOCUPA,ATOM)
      IF(ATOM.LE.Ø) GO TO 85
C   IF ATOM IS ON BOND PUT THE BOND LABELING THE ATOM IN RING SET
      IF(ISM(INC(ATOM),BOND)) CALL ON(BASISRG(NBR),LABEL(ATOM))
      GO TO 8Ø
C
C   THIS BASIS RING IS COMPLETE
C   OR INTO THE SET OF RING BONDS
  85  SRINGB = OR(SRINGB,BASISRG(NBR))
      GO TO 7Ø
C   END OF WELCH'S ALGORITHM


C   THIS IS THE SECTION THAT ATTEMPTS TO FIND RING CLUSTERS
  9Ø  CONTINUE
      STEMP = SNULL
      DO 22Ø  I=1,NBR
      CALL    ON(STEMP,I)
 22Ø  CONTINUE
      I = Ø
      NCLUSTERS = Ø
C   TAKE A BASIS RING
 222  I = NXM(STEMP,I)
      IF(I.EQ.Ø) GO TO 226
      NCLUSTERS = NCLUSTERS + 1
C   START A NEW CLUSTER
      SCLRG(NCLUSTERS) = SNULL
      CALL    ON(SCLRG(NCLUSTERS),I)
      SCLBD = BASISRG(I)
 223  J = I
      DELFLG = .FALSE.
 224  J = NXM(STEMP,J)
      IF(J.EQ.Ø) GO TO 225
      IF(AND(SCLBD,BASISRG(J)).EQ.SNULL) GO TO 224
```

```
C   ADD THIS RING TO THE CLUSTER
      DELFLG = .TRUE.
      CALL    ON(SCLRG(NCLUSTERS),J)
      CALL    OFF(STEMP,J)
      SCLBD = OR(SCLBD,BASISRG(J))
      GO TO 224
C   IF ADDED MORE RINGS TO CLUSTER REPEAT
 225  IF(DELFLG) GO TO 223
      GO TO 222

C   HAVE FOUND ALL CLUSTERS
C   NOW PROCESS EACH CLUSTER OF RINGS
 226  DO 163Ø INDEX=1,NCLUSTERS
      SETLEN = 1
      RNCL = CNT(SCLRG(INDEX))
C   THE MAX IS 1Ø BASIS RINGS PER CLUSTER
      IF(RNCL.GT.1Ø) RNCL = 1Ø
      LENSET = (2**RNCL)/NBPWD + 1
      I = Ø
      J = Ø
 23Ø  I = NXM(SCLRG(INDEX),I)
      IF(I.EQ.Ø) GO TO 235
      J = J + 1
      CBASISRG(J) = BASISRG(I)
      GO TO 23Ø
 235  CONTINUE

C   THIS SECTION IS USED TO CALCULATE THE REDUCED BASIS SET FROM
C   THE ARBITRARY BASIS SET THAT WE JUST OBTAINED

C   THIS IS NOT NECESSARY IF THERE IS ONLY 1 RING
      IF(RNCL.LE.1) GO TO 28Ø
      N2 = RNCL-1
 245  DELFLG = .FALSE.
      DO 275  J=1,N2
      CNTRJ = CNT(CBASISRG(J))
      I2 = J + 1
      DO 27Ø  I=I2,RNCL
      CNTRI = CNT(CBASISRG(I))
      STEMP = XOR(CBASISRG(I),CBASISRG(J))
      CNTRIJ = CNT(STEMP)
C   IF THE EXCLUSIVE OR IS SMALLER THAN EITHER OF THE ORIGINAL
C   TWO BASIS RINGS, REPLACE THE LARGER OF THE TWO BASIS RINGS
C   WITH IT.
      IF(CNTRIJ.GE.CNTRI) GO TO 25Ø
      DELFLG = .TRUE.
      TEMP = CNTRI
      CNTRI = CNTRIJ
      CNTRIJ = TEMP
      STEMP2 = CBASISRG(I)
      CBASISRG(I) = STEMP
      STEMP = STEMP2
 25Ø  IF(CNTRIJ.GE.CNTRJ) GO TO 27Ø
      DELFLG = .TRUE.
      CNTRJ = CNTRIJ
      CBASISRG(J) = STEMP
 27Ø  CONTINUE
 275  CONTINUE
      IF(DELFLG) GO TO 245

 28Ø  CONTINUE

C   GIBB'S ALGORITHM
C   QSETS ARE ALL OF THE COMBINATIONS OF CBASISRG BOND SETS
C   QFREE IS A POINTER TO THE NEXT FREE LOCATION IN THE QSETS TABLE
C   RNCL IS THE NUMBER OF BASIS RINGS IN THE CLUSTER
C   S IS THE SET THAT EVENTUALLY CONTAINS ALL OF THE RINGS
      SETLEN = LENSET
      CALL ZEROM(S,3Ø)
      QFREE = 1
C
      DO 12Ø  I=1,RNCL
      IF(I.EQ.1) GO TO 115
      LAST = QFREE - 1
      DO 11Ø J=1,LAST
      SXOR = XOR(QSETS(J),CBASISRG(I))
C   IF NO INTERSECTION THEN NOT A NEW RING
      IF(AND(CBASISRG(I),QSETS(J)).EQ.SNULL) GO TO 1Ø6
C   IF ANOTHER SET IS A SUBSET OF THIS SET THEN NOT NEW RING.
C   IF THIS SET IS A SUBSET OF ANOTHER SET REMOVE THE OTHER.
      K = LAST
 1Ø2  K = NXM(S,K)
      IF(K.LE.Ø) GO TO 1Ø4
      IF(SUB(QSETS(K),SXOR)) GO TO 1Ø
      IF(SUB(SXOR,QSETS(K))) CALL OFF(S,K)
      GO TO 1Ø2
 1Ø4  CALL    ON(S,QFREE)
      GO TO 1Ø8
 1Ø6  IF(I.EQ.RNCL) GO TO 11Ø
 1Ø8  QSETS(QFREE) = SXOR
      QFREE = QFREE + 1
 11Ø  CONTINUE
C
C   ADD THE CBASISRG TO THE S SET NOW TOO
 115  QSETS(QFREE)=CBASISRG(I)
      CALL ON(S,QFREE)
 12Ø  QFREE=QFREE+1
C   THE ALGORITHM IS NOW FINISHED
C   INTRA CLUSTER PROCESSING HERE, EG, FIND BRIDGES
163Ø  CONTINUE
C  SECS THEN SELECTS USEFUL RINGS AND CREATES RING LISTS
C  S IS THE SET OF RINGS, QSET(I) THE SET OF BONDS IN RING I
C  BASISRG(I) THE BONDS IN BASIS RING I
      RETURN
      END
```

Figure 7. Program listing of Welch-Assembly-Gibbs Algorithm.

The ring assemblies are then found from the basis rings by the algorithm presented earlier. A flowchart of the process is given in Figure 3. Each ring assembly is then processed by Gibbs' algorithm,[9] which finds all of the rings in that assembly by taking the logical exclusive or of all combinations of 1 to n of the n basis rings in the assembly, and examining each combination to see if it is a new ring. Gibbs presented the algorithm in a symbolically concise manner; however, the implementations[8b] of the algorithm which are most apparent from the symbolic description are much less efficient in execution time than the one implemented here. The implementation used is better represented symbolically as:

```
1. $nq \leftarrow 0; i \leftarrow 0, S \leftarrow \phi$
2. $nq \leftarrow nq + 1; i \leftarrow i + 1; Q_{nq} \leftarrow B_i; S \leftarrow S \cup \{nq\}$; last
   $\leftarrow nq; j \leftarrow 1$; if $i >$ NBR then go to step 6 (NBR is
   number of basis rings)
3. $nq \leftarrow nq + 1; Q_{nq} \leftarrow Q_j \oplus B_i$; if $Q_j \cap Q_{nq} = \phi$ then go
   to step 5
4. $S \leftarrow S \cup \{nq\}$. For every $k \in S: k >$ last do begin
   if $Q_{nq} \subset Q_k$, then $S \leftarrow S - \{k\}$;
   if $Q_k \subset Q_{nq}$, then begin $S \leftarrow S - \{nq\}$;
   go to step 5 end end
5. $j \leftarrow j + 1$; if $j \leq$ last then go to step 3, else go to step 2
6. Done. The set of all rings .=. $\{Q_i: i \in S\}$.
```

This implementation requires only one array Q and one "superset" S. This implementation avoids the costly shuffling and transferring of large numbers of sets. Additional storage efficiency is possible since when $i = k$, sets need only be stored in Q if they are in fact new rings. Figure 4 is a flowchart of Gibbs' algorithm as actually implemented by us.

A *reduced basis set* of rings is then found for each assembly. A reduced basis set is an attempt, not always a successful one, to find a basis set of rings which consists of the smallest rings that can form a basis set,[10] sometimes called a *minimal covering set*.[2] A reduced basis set is obtained by taking the XOR of every combination of two of the bond sets which represent the rings in the basis set, and then retaining the two smaller of the three sets. This process is repeated until the sets retained remain unchanged. A flowchart of this process is presented in Figure 5.

The "Chemically interesting" rings are then selected from all of the rings. "Chemically interesting" rings are defined for our purposes as the reduced basis rings and any other rings of up to eight bonds. Lists of the atoms and bonds in order around these rings are then generated. A complete listing of the ring perception routine used in SECS is given in the Appendix.

It would seem to be very inefficient to generate all rings and then select the desired rings from them, as opposed to only generating the desired rings, but owing to the speed of the set operations used this is not the case. The algorithm presented here is faster than any previously published algorithm[2,8-10] for finding rings in chemical structures. The graph in Figure 6 shows the reported times for various algorithms. Since the ring assembly algorithm introduces additional parameters, namely the number of assemblies and the distribution of the basis rings among them, it is impossible to plot the execution time versus the number of basis rings as a smooth curve, which even for the other algorithms is very crude. Therefore the times given in Table I for typical polycyclic chemical structures are merely plotted as X's.

APPENDIX. WELCH-ASSEMBLY-GIBBS ALGORITHM.

This appendix describes the implementation of the Welch-Assembly-Gibbs algorithm on a Digital Equipment PDP-10 computer in Fortran IV (see Figure 7). For further explanation of the set operations used here, see p 157–9, ref 1a. For up to 36 atoms, the sets used may be of length one work (SETLEN = 1) except during Gibb's algorithm when SETLEN = LENSET. The use of the assignment statement with sets is straightforward when they are only one word long, e.g., SETA = XOR(SETB,SETC). If more than one word is needed, then one may use a subroutine to perform assignment, or on some computers one may use double precision. NBPWD is the number of bits per word of the computer, set to 36 for the PDP-10. AT1(i) and AT2(i) are the two atoms connected by bond i. The input to RING consists of NA (the number of atoms), NB (number of bonds), SOCUPA (set of atoms in structure), SOCUPB (set of bonds in structure), and arrays AT1 and AT2. SOCUPA and SOCUPB are used to allow "holes" in the bond table; i.e., the NB bonds need not occupy the first NB positions of arrays AT1 and AT2.

RING is dimensioned for 36 atoms (LABEL, INC, AT1, AT2), 20 assemblies (SCLRG), a maximum of 10 basis rings per assembly (CBASISRG), 50 total basis rings (BASISRG), and 612 total rings (QSETS) which allows on the PDP-10 the use of one word sets. In SECS, we use two word sets for 72 atoms.

The algorithm given here finds all rings. The definition of the set of "interesting" rings is application dependent and therefore must be added at the end. The superset S is the set of all rings. The set of bonds in ring i is contained in QSETS(i).

Subroutines and functions used are described in Table III.

ACKNOWLEDGMENTS

This work was supported by National Institutes of Health Research Grant No. RR-00578 from the Division of Research Resources and in part by E. Merck, Darmstadt.

Table III

General functions:	
OR(SET1, SET2)	*Inclusive or* of two sets; union of sets
AND(SET1, SET2)	*Intersection* of two sets
XOR(SET1, SET2)	*Exclusive or* of two sets
NOT(SET1)	*Complement* of a set
Logical functions:	
EQV(SET1, SET2)	T if two sets are equivalent, F otherwise
ISM(SET1, N)	T if N is a member of set SET1, else F.
ON(SET1, N)	T if N was member of set SET1, else F. Side effect: makes N a member of set SET1.
OFF(SET1, N)	T if N was not member of set SET1, else F. Side effect: removes N from set SET1.
SUB(SET1, SET2)	T if SET1 is subset of SET2. (proper subset)
Integer functions:	
CNT(SET1)	Value is number of members in SET1.
NXM(SET1, N)	Value is next member of set after N. If no more members, value is zero.
Subroutines:	
ZEROM(ARRAY, NWORDS)	Zeros NWORDS of ARRAY

LITERATURE CITED

(1) (a) This work was presented in part at the 7th Middle Atlantic Regional Meeting of the American Chemical Society, Philadelphia, Feb 14–17, 1972. For an earlier paper and leading references see W. T. Wipke, S. R. Heller, R. J. Feldmann, and E. Hyde, "Computer Representation and Manipulation of Chemical Information," Wiley, New York, N.Y., 1974. (b) Merck Career Development Award. Current address: Department of Chemistry, University of California, Santa Cruz, Calif. 95060. (c) NSF Trainee, 1969–71.

(2) Corey, E. J., and Wipke, W. T., *Science*, **166**, 178 (1969).

(3) Welch, J. T., Jr., *Assoc. Computing Mach.*, **13**, 205 (1966).

(4) Gotlieb, C. C., and Corneil, D. G., *Comm. Assoc. Computing Mach.*, **10**, 780 (1967).

(5) Paton, K., *Comm. Assoc. Computing Mach.*, **12**, 514 (1969).

(6) Tiernan, J. C., *Comm. Assoc. Computing Mach.*, **13**, 722 (1970).

(7) Fugmann, R., Dolling, U., and Nickelson, H., *Angew. Chem., Int. Ed. Engl.*, **6**, 723 (1967).

(8) (a) Long, P. L., Masters Thesis, Ohio State University, 1970; (b) Phares, R. F., and White, L. J., Ohio State University Computer Information Science Research Technical Report, OSU-CISRC-TR-70-7; (c) Long, P. L., Phares, R. F., Rush, J. E., and White, L. J., Abstract CHLT-15, 160th National Meeting of the American Chemical Society, Chicago, Ill., Sept 1970.

(9) Gibbs, N. E., *J. Assoc. Computing Mach.*, **16**, 564 (1969).

(10) Corey, E. J., and Peterson, G. A., *J. Am. Chem. Soc.*, **94**, 460 (1972); Plotkin, M., *J. Chem. Doc.*, **11**, 60 (1971); Bersohn, M., *J. Chem. Soc., Perkin Trans. 1*, 1239 (1973).

(11) Ring *i* is a member of the same assembly *k* as ring *j* if ring *i* shares one or more *edges* with ring *j*, or if ring *i* shares at least one edge with another ring previously found to be a member of assembly *k*. Note that sharing only a *node* (spiro linkage) between ring *i* and a member ring of assembly *k* is insufficient for inclusion of ring *i* in assembly *k*.

(12) An adjacency matrix is a square matrix where a 1 in the *i,j* position indicates atoms *i* and *j* are bonded (adjacent), while a 0 in the *k,l* position indicates atoms *k* and *l* are not directly bonded (not adjacent). In an incidence matrix, the column index refers to the bond while the row index refers to the atom, so that if the *i,j* position is a 1, atom *i* is on bond *j* and if the *k,l* position is a 0, atom *k* is not on bond *l*. See also M. F. Lynch, J. M. Harrison, W. G. Town, and J. E. Ash, "Computer Handling of Chemical Structure Information," American Elsevier, New York, N.Y., 1971.

(13) Morgan, H. L., *J. Chem. Doc.*, **5**, 107 (1965). The Morgan name is the canonical name used by the Chemical Abstracts Registry system.

(14) For a stereochemically unique naming algorithm, see W. T. Wipke and T. M. Dyott, *J. Am. Chem. Soc.*, **96**, 4834 (1974).

(15) Logical exclusive or gives all of the elements in either set, but not in both. The importance of set representations was first pointed out by C. H. Sussenguth, *J. Chem. Doc.*, **5**, 36 (1965).

(16) Without the use of sets of the "exclusive or" sets, the storage required is essentially twice as great.[7b] The use of the *sets of sets* also speeds up the algorithm considerably, since it replaces the physical transfer of the "exclusive or" sets with a fast operation on the set of the "exclusive or" sets.

(17) The structures given in the "Dictionary of Organic Compounds" and the 18th edition of the "Merck Index" are used as the justification for this statement.

An Efficient Design for Chemical Structure Searching. I. The Screens[†]

ALFRED FELDMAN*

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

LOUIS HODES

National Cancer Institute, National Institutes of Health, Bethesda, Maryland 20014

Received January 21, 1975

A method has been developed for generating efficient screens for chemical structures. Fragments are generated by an algorithm under control of file statistics. The fragments obtained are normalized by weighting their code patterns. Superimposition of these codes yields the screen codes for the structures.

I. INTRODUCTION

The chemical structure search system at the Walter Reed Army Institute of Research (WRAIR) has been in operation since 1962.[1] In this period, its files have grown to about a quarter million compounds. Currently, the system is being converted, from a sequential tape file operation, to a direct access file based design. As part of this effort, an improved system of screens has been developed.

Screens are used in all files with substructure search capability. Their use makes it possible to avoid much laborious atom-by-atom matching: the more efficient the screens, the fewer the compounds that must be eliminated on the basis of atom-by-atom matching.

[†] Paper presented at the 167th and 168th National Meetings of the American Chemical Society, Atlantic City, N.J., Sept 1974, and Philadelphia, Pa., April 1975.

Screening can be thought of as a conventional search, in which structural fragments are used as descriptors for compounds and queries. Where these descriptors cannot be matched, further searching is unnecessary. Screening is thus related to the classification problem, where it is asked what are the best attributes for classification, and how many should there be. Compared with nonchemical data files, collections of chemical structures are remarkable in that their attributes—generally their fragments—are precise and abundant. This abundance, evident from the example shown in Figure 1, forces a choice among fragments and introduces the problem of optimizing screens for searching.

As yet, no system has advanced a set of optimum screens. Chemical files, consequently, are partly underscreened and partly overscreened. They cannot be efficiently searched nor, for that matter, efficiently stored. There is not even agreement as to the approach to be taken toward screen