A Hyperstructure Model for Chemical Structure Handling: Generation and Atom-by-Atom Searching of Hyperstructures

Robert D. Brown, Geoffrey M. Downs, and Peter Willett*

Department of Information Studies, University of Sheffield, Western Bank, Sheffield S10 2TN, U.K.

Anthony P. F. Cook†

ORAC Ltd., 18 Blenheim Terrace, Leeds LS2 9HD, U.K.

Received April 10, 1992

This paper discusses the representation of sets of chemical structures by hyperstructures, pseudomolecules in which areas of structural commonality are stored nonredundantly. Two methods are described for the construction of hyperstructures, and it is shown that one of these methods, the atom-assignment method, is far less demanding of computational resources than the other. Experiments with a file of 10K structures demonstrate that the hyperstructure for a dataset requires less storage than the original set of structures; however, the difference is not large. The experiments also demonstrate that hyperstructures have at least some potential for increasing the speed of atom-by-atom searching, as compared with conventional chemical database systems.

1. BACKGROUND

Substructure searching in chemical information systems is usually effected by a two-stage retrieval mechanism. In this, a detailed atom-by-atom mapping procedure, which uses a subgraph isomorphism algorithm, is preceded by a rapid, bit string screening search, which eliminates the great bulk of the search file from the atom-by-atom search.[1] A database of structures is usually organized either as a serial file or as an inverted file. Vladutz and Gould[2] have proposed a novel file organization based on the use of a *hyperstructure*, which is a pseudomolecule that is formed by the superimposition of sets of molecules and which stores areas of structural commonality only once. The connectivity of a group of molecules is thus encoded with minimal redundancy, with each individual structure being contained as a substructure of the complete hyperstructure. The elimination of duplicate substructural moieties means that a hyperstructure is expected to require less storage than its constituent molecules, resulting in some degree of data compression; moreover, the elimination of these moieties may result in increased substructure-searching speeds, since the repeating units need be searched only once. An example of a hyperstructure is shown in Figure 1, in which molecules 1–3 are contained in the hyperstructure, *H*. An analgous data structure, called a *trie*, is used in text-processing systems for the storage and searching of large dictionaries.[3]

Hyperstructures have a number of features that distinguish them from conventional molecules, as follows:

1. Atoms are normally linked to a small number of neighboring atoms, with the coordination number being limited by the maximum valence for the atom type. In a hyperstructure, conversely, an atom may be connected to any, or all, of the other atoms in the hyperstructure; for example, nodes c and d in Figure 1 represent a pentavalent carbon atom and a divalent chlorine atom, respectively.
2. In a hyperstructure, any two nodes may be linked by any number of different bond types, whereas each connection in a normal molecule involves only a single bond type. In a hyperstructure, therefore, each bond has an associated list of identifiers indicating the structure(s) from which that bond originated. For example, the bond a–b in Figure 1 originates from molecules 1 and 2, the bond a–c from molecule 1, the bond c–d from molecule 3, and the bond b–d from molecule 2.
3. A hyperstructure may contain *ghost* substructures, i.e., substructures that are made up from bonds occurring in different constituent molecules and that do not actually occur in any of the molecules that comprise the hyperstructure. An example of a ghost substructure is the a–b–c–d feature in Figure 1.

Structural identifiers are associated with the various components of hyperstructure so that it contains sufficient information to allow the reconstruction of any of its constituent structures.

Although hyperstructures differ from conventional, 2D chemical structures, they have at least some similarities with two forms of structure representation that have become of increasing importance over the last few years, viz., three-dimensional (3D) chemical structures and Markush structures. Three-dimensional database searching systems use interatomic

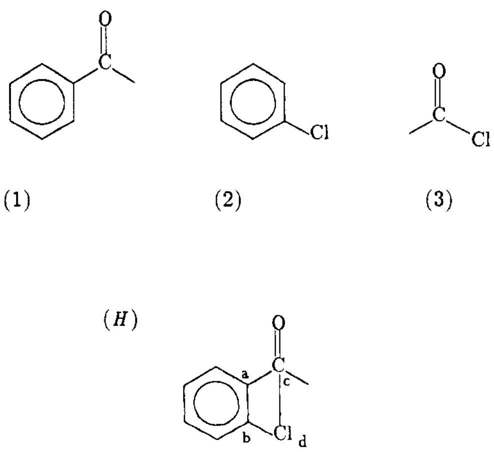


Figure 1. Hyperstructure *H* of input molecules 1–3.

* To whom all correspondence should be addressed.
† Present address: Synopsys Scientific Systems Ltd., 175 Woodhouse Ln., Leeds LS2 3AR, U.K.

distance matrices, which contain the distances between all pairs of atoms in a molecule.[4] This is an example of a *fully-connected graph*, i.e., one in which there is an edge between each and every pair of nodes; an alternative example would be the limiting case of a hyperstructure in which each atom is linked to every other atom. A Markush structure is used to denote all of the individual structures that are described, both explicitly and implicitly, by a chemical patent.[5] The graphs representing Markush structures can thus be extremely complex, describing alternative connectivity patterns between some number of substructural moieties or logical relationships between the presence of such moieties, e.g., a Markush structure might encode the fact that substructures X and Y can be linked in two different ways or that X is present only in the absence of Y.

Vladutz and Gould suggest two approaches to the construction of hyperstructures.[2] The first involves the stepwise superimposition of a series of structures and achieves the greatest possible degree of compression by the selection of the maximal overlap between the hyperstructure and an input structure; the overlap is identified using a maximal-common-subgraph (MCS) algorithm. Alternatively, they suggest that structures may be randomly superimposed onto a preconstructed complete graph, in which all of the constituent atoms are bonded to each other. Such a graph will contain all possible structures as subgraphs, and the superimposition is then followed by the deletion of any hyperstructure nodes and arcs that have not been used during the construction process.

In this paper, we discuss two methods that have been developed for the construction of hyperstructures, both based on stepwise assembly. The first, the *maximal-overlap-set method*, uses an MCS-like superimposition algorithm; the second, the *atom-assignment method*, uses a simple atom-assignment technique that is closely related to Vladutz and Gould's random method but that dispenses with the need to preconstruct the complete graph. In both cases, the mapping procedure forms the first stage of a repeated two-stage process, in which the second stage uses the mapping produced between the input and hyperstructure nodes to update the hyperstructure in preparation for the next mapping to be generated. The processing can be illustrated by the pseudocode below, where H is a hyperstructure, $S(X)$ is the Xth member ($1 \le X \le N$) of a file of N structures, $COMPARE(H,S(X))$ is a procedure that maps $S(X)$ to the current hyperstructure to identify the areas of structural commonality and difference, and $UPDATE(H)$ is a procedure that updates the connection table-like data structure that is used to represent H (and that is discussed in detail later in this paper).

```
H := S(1)
FOR X := 2 TO N DO
  BEGIN
    COMPARE (H,S(X))
    UPDATE (H)
  END
```

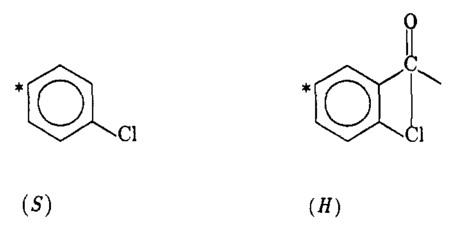
Vladutz and Gould do not appear to have implemented their ideas. In this paper, we evaluate the utility of hyperstructures for the storage of files of structures. The next two sections describe the maximal-overlap-set and atom-assignment methods, and this is followed by a description of the use of hyperstructures for organizing both small and large files of molecules. We discuss the storage savings that might be expected to accrue from the use of a hyperstructure organization, and then compare the execution times obtained in atom-by-atom searches of hyperstructures and of conventional, serial files of structures. The paper concludes with a summary of our main findings.

2. MAXIMAL-OVERLAP-SET METHOD

2.1. Introduction. The maximal common subgraph of two graphs may be defined as the portion of each graph containing the largest number of arcs that is common to the two.[6] The problem of locating the MCS for two graphs is known to be NP-complete, and thus extremely demanding of computational resources. Efficiency of execution is obtained by use of a backtracking, depth-first tree search in which areas of structural commonality are 'grown', one atom at a time. The method described below is based on the MCS algorithm by McGregor,[6] with the addition of node-set partitioning before the backtrack search. The partitioning seeks to classify the nodes of one graph into sets, one for each node of the other graph, indicating which nodes could possibly correspond to which. These sets are ordered such that the nodes with the most similar environment to the target node appear first. This ordering should facilitate the pruning of the search tree during the time-consuming backtracking stage by allowing good solutions to be identified at an early stage of the search.

2.2. Node-Set Partitioning. Read and Corneil[7] note that any property which is invariant under isomorphism may be used for node-set partitioning, but that methods which compare the degrees, or connectivities, of nodes are conventionally used. Examples of the use of such methods in the chemical context are described by Morgan[8] and by Lynch and Willett.[9] These methods can be very efficient for chemical applications because of the simplicity of the graphs that are being compared; specifically, the fact that both the nodes and the edges of the graphs are labeled and that the nodes have low connectivities (since an individual atom in a conventional molecule is typically bonded to only a small number of other atoms). As we have noted above, however, any node in a hyperstructure may be connected to any, or all, of the other nodes, with the result that a hyperstructure may contain many, very highly-connected nodes in similar and complex environments. Standard partitioning methods are not applicable in such a situation, and we have thus adopted a solution which allows the identification of hyperstructure nodes whose environments include that of each node of the input structure, $S(X)$, that is to be mapped to the current hyperstructure, H.

To allow this comparison to be made, the environments of nodes in each structure are coded on the basis of the atom types of their successively more-distant neighbors, out to a predefined maximum radius. This is, in principle, similar to a method described by Randić[10] and by Randić and Wilkins[11] for increasing the speed of substructure searching. Specifically, each atom in each structure is represented by an $R \times T$ matrix where T represents the number of atom types considered and R is the radius in terms of numbers of bonds. After initial experiments, T was set to 4 (the types being carbon, nitrogen, oxygen, and any other), and R was set to 6. The Ith row of the matrix ($1 \le I \le 6$), therefore, represents the number of C, N, O, and X (other) atoms I bonds away from the atom in question. A matrix is produced for each node in the input structure and in the hyperstructure, and the matrices of hyperstructure and input structure nodes of the same atomic type are then compared by using the asymmetric similarity coefficient.[12] For an input structure node represented by a matrix MS and a hyperstructure node represented



MS	C	N	O	X
1	2	0	0	0
2	2	0	0	0
3	1	0	0	0
4	0	0	0	1

MH	C	N	O	X
1	2	0	0	0
2	2	0	0	0
3	2	0	0	0
4	1	0	1	1

$min\{MS, MH\}$	C	N	O	X
1	2	0	0	0
2	2	0	0	0
3	1	0	0	0
4	0	0	0	1

Figure 2. Calculation of the asymmetric similarity coefficient between the atoms marked by an asterisk in the input molecule, *S*, and the hyperstructure, *H*. The value of the asymmetric similarity is 6/9, i.e., 0.667.

by the matrix *MH*, the asymmetric coefficient is given by

$$\frac{\sum_{I=1}^{R}\sum_{J=1}^{T}\min\{MS(I,J), MH(I,J)\}}{\sum_{I=1}^{R}\sum_{J=1}^{T} MH(I,J)}$$

where the numerator is the sum of the minimum components of the two matrices and where the denominator is the sum of the elements of the matrix *MH*. The asymmetric coefficient measures the degree of inclusion of one vector in another and is thus appropriate to the present situation, where an input node may match any hyperstructure node whose environment contains its own. A list of candidate hyperstructure nodes is produced for each input node, with each list ordered on the basis of the asymmetric similarities, and these lists are then passed to the backtrack-searching algorithm. Figure 2 shows one example of this calculation for the addition of a further structure, *S*, to the hyperstructure, *H*, given in Figure 1; the matrices in Figure 2 are shown for the atoms marked with an asterisk.

2.3. Backtrack Searching. We have noted previously that the MCS for two structures may be defined as the portion of each graph containing the largest number of arcs which is in common to the two. In fact, McGregor's algorithm identifies the MCS according to an extended definition, which states that, given that the maximum number of arcs is in common between the two graphs, the MCS is the subgraph also containing the maximum number of nodes.[6] McGregor's algorithm uses a backtrack search to assemble substructures in stepwise fashion, so that a fragment is increased in size by tentatively assigning node correspondences between nodes adjacent to the fragment already produced from each structure. The maximum number of arc correspondences that could result from each tentative node assignment is calculated and the search pruned as soon as this number falls below that of the best solution found so far. Once all possible correspondences have been exhausted, an additional step is included to check whether a better solution could possibly result from the current node in the input structure having no matching node in the hyperstructure.

To produce the hyperstructure with the minimum number of nodes, however, it is necessary to identify the *maximal overlap set*, or MOS, between two graphs, rather than the MCS.[2] The maximal overlap set between two structures is defined here as a set of common subgraphs of the structures, such that the maximum number of arcs of one graph has a correspondence in the other and, additionally, such that the maximum number of nodes is also in common. An example is given in Figure 3. McGregor's algorithm builds the MCS by requiring that each node that is matched should be adjacent to a node in the partial solution that has already been generated. The MOS can be produced, rather than the MCS, by dropping this adjacency requirement. There is a computational cost involved in this, since many more correspondences are available at each point of the search tree and so much less pruning can be carried out than is the case when the MCS is to be identified.

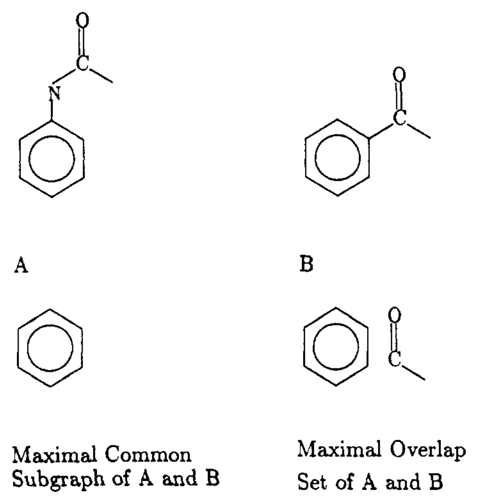

A
B
Maximal Common Subgraph of A and B
Maximal Overlap Set of A and B

Figure 3. MCS and MOS of structures A and B.

2.4. Order of Processing. The method described here assembles hyperstructures by the successive superimposition of single molecules, and the order in which the structures are added will consequently affect the ease with which the mapping between a structure and the hyperstructure can be identified. The functioning of the MOS algorithm is such that it is possible to confirm that a certain node will have no correspondence in the target structure only after all of the possible correspondences have been dismissed. Accordingly, the greater the degree of overlap between two structures of comparable molecular size, the more quickly the solution can be identified (providing that the node-partitioning procedure has been successful).

The structure file is preprocessed so that each structure in the file is followed by the remaining structure with which it has the greatest similarity, which should ensure that a large overlap will exist between each structure and the hyperstructure onto which it will be mapped. Each structure is characterized in terms of the types and numbers of augmented atoms present: a Tanimoto coefficient of similarity[13] is then produced between every pair of molecules and the structures are ordered on the basis of these coefficients.

Several possibilities exist for the selection of the first structure in the series. The following orders of processing were examined.

1. Random, i.e., no order
2. Largest to smallest structure, without any similarity calculation
3. Ordered by similarity coefficients, starting with the two most similar structures and selecting the largest molecule when two or more coefficients were equal
4. Ordered by similarity coefficients, starting with the largest structure in the file
5. Ordered by similarity coefficients, starting with the two most similar structures and selecting the smallest molecule when two or more coefficients were equal

6. Ordered by similarity coefficients, starting with the smallest structure in the file

The rationale for selecting the largest structures is to provide the greatest selection of hyperstructure nodes for the new structure nodes to map to, hence producing a large overlap and speeding the backtrack search. The final two orders provide a contrast to this.

3. ATOM-ASSIGNMENT METHOD

The atom-assignment, or AA, method provides an alternative way of producing a mapping between an input structure and hyperstructure. This again uses Vladutz and Gould's stepwise superimposition approach but takes account only of the atoms of the two structures and takes no account of the patterns of bonds in the two structures. The mapping of structure atoms to hyperstructure atoms is made by selecting the first available hyperstructure atom of the correct type to match each successive input atom. There is no attempt to match bonding patterns in the two structures, and extra bonds in the hyperstructure are simply created as necessary to correspond to the bonds that are present in the structure that is being mapped to the hyperstructure. Rather than have the connections between hyperstructure nodes distributed over all parts of the structure, the input structure nodes are mapped in order of decreasing connectivity, starting with the most highly coordinated, so concentrating the bulk of the connections onto the first nodes of the hyperstructure. This should reduce the number of nodes that will have to be considered for most queries in subsequent substructure searches of the final hyperstructure.

A pseudocode description of the atom-assignment method is as follows, where H and S refer to the hyperstructure and the input structure, respectively:

```
FOR I := 1 TO NO_OF_S_NODES DO
  BEGIN
    J := 1
    REPEAT
      IF (H_ATOM_TYPE[J] = S_ATOM_TYPE[I])
      AND NOT (ASSIGNED[J])
      THEN
        BEGIN
          STORE[I] := J
          ASSIGNED[J] := TRUE
        END
      ELSE J := J+1
    UNTIL (J > NO_OF_H_NODES) OR (STORE[I] <> 0)
  END
```

In this pseudocode, $STORE[I]$ records the number of the hyperstructure node that is matched to the Ith input structure node, and $ASSIGNED$ records which hyperstructure nodes have been previously matched and are hence not available for assignment. If $STORE[I] = 0$ for any input structure node after the procedure has finished, then no matching hyperstructure node exists and a new one will be created.

This method will produce a more highly-connected hypergraph than the MOS method, which is expected to have a detrimental effect on the degree of compression that is achieved in the hyperstructure. The size of the hyperstructure in terms of nodes will not vary, however, between the two methods since this is simply determined by the sum of the maximum number of nodes of each type in any one input molecule.

4. GENERATION OF HYPERSTRUCTURES

4.1. Small Datasets. The two methods described above were used to process a series of small datasets into hyperstructures. The investigation considered two features of the construction process. The first, and more important, consideration was the computational effort required for the process. If either method is to be of practical value, it must be possible to assemble hyperstructures for large datasets in a realistic amount of time. The second consideration was the saving in storage, if any, that results from building a hyperstructure.

Table I. Generation of Hyperstructures for Small Datasets[a]

dataset	no. of structures	CPU time		C_{Nodes}		C_{Arcs}	
A	37	–[b]	2.8	–	6.1	–	18.7
B	27	154.0	2.6	6.2	6.2	18.3	18.6
C	25	12.9	1.3	5.2	5.2	12.6	16.2
D	56	176.1	1.9	3.6	3.6	8.1	10.3
E	20	63.8	2.7	10.7	10.7	36.9	36.9
F	105	–	10.7	–	1.7	–	8.4
G	46	–	6.0	–	4.9	–	17.7
H	137	578.1	4.8	2.1	2.1	7.5	8.5

[a] Each column in the right-hand part of the table contains two figures, the first of which corresponds to the MOS method and the second to the AA method. [b] Blank entries (–) are for datasets where it was not possible to complete the processing within 5400 CPU s using the MOS method.

Eight small datasets were used to test the two methods for constructing hyperstructures: (A) 37 structurally diverse compounds; (B) 28 substituted benzohydroxamic acids; (C) 25 aliphatic and carbocyclic ethers; (D) 56 aliphatic alcohols, ketones, ethers, and esters; (E) 20 naturally-occurring amino acids; (F) 105 benzamides; (G) 46 quinazolines; (H) 137 alkanes, alkenes, alcohols, ketones, and benzene and pyridine derivatives. Datasets B, C, D, E, and H are largely homogeneous while A, F, and G show much greater heterogeneity.

Each dataset was processed using the MOS method, with the structures ordered as given previously. Table I shows the processing times required in CPU seconds for Pascal implementations on an IBM 3083 mainframe running under VM/CMS. Full details of the findings discussed in this section are given in Brown.[14] A time limit of 5400 s was imposed upon the processing; any datasets that could not be processed within this time limit are left blank in the table. It was possible to produce hyperstructures only for those datasets that contained small numbers of highly homogeneous structures: with large molecules, the backtracking search tree was too large to be searched to completion, while the small degree of overlap that exists in the heterogeneous sets of structures means that little pruning of the tree was possible. The search tree grows as the hyperstructure itself grows and becomes more complex, and the task of mapping successive structures therefore becomes more difficult. A similar problem was encountered by Okada and Wipke, who noted that 'the memory and cpu time required are too large for routine use' when using McGregor's algorithm in their CLUSMOL system.[15]

For those datasets from which hyperstructures could be constructed, the most favorable method for preprocessing the structures was found to be selecting the largest structure in the dataset and then selecting the most similar structures in sequence after that (order 4). The only occasion on which this failed was for dataset B, in which the largest molecule contained complex features not found in the remainder of the dataset. In the best case, for set E, the processing time was almost five times better than by any of the other orderings. Those orders involving the initial selection of the smallest molecules were worse than random in most cases.

Let structure $S(X)$ $(1 \leq X \leq N)$ contain $NS(X)$ nodes and $AS(X)$ arcs, and let the hyperstructure H contain NH nodes

Table II. Variation in Percentage Storage Requirements with Number of Structures Used for Creation of Global Hyperstructure for EINECS Dataset

no. of structures	C_{Nodes}	C_{Arcs}
1	100.00	100.00
1 000	1.15	8.62
2 000	0.52	4.23
3 000	0.30	3.15
4 000	0.22	2.34
5 000	0.19	1.96
6 000	0.16	1.71
7 000	0.15	1.61
8 000	0.14	1.51
9 000	0.12	1.36
10 000	0.12	1.29
10 794	0.11	1.08

and *AH* arcs. Then the percentage storage requirements for nodes and arcs, C_{Nodes} and C_{Arcs}, are given by

$$C_{\text{Nodes}} = \frac{NH \times 100}{\sum_{X=1}^{N} NS(X)}$$

and

$$C_{\text{Arcs}} = \frac{AH \times 100}{\sum_{X=1}^{N} AS(X)}$$

respectively. Thus, the smaller the C_{Nodes} and C_{Arcs} values, the greater the degree of compression that has been achieved. Table I details the node and arc storage requirements achieved by using the most favorable order of processing, where it will be seen that very substantial compressions are observed for those datasets for which it was possible to generate a hyperstructure.

The eight datasets were also processed using the AA method. In this case the order of processing is not important, and the structures were processed in the order in which they occurred in the original dataset. The results in Table I show that the AA method allows hyperstructures to be produced from any given dataset in an acceptable time. The AA method is between 1 and 2 orders of magnitude faster than the MOS method for those datasets that could be processed by the latter method within the time limit (and even faster for those that could not). While the node compressions are the same for both methods (the size of the hyperstructure being determined by the sum of the maximum number of nodes of each atom type in any one input structure), the arc compressions resulting from the AA method are generally marginally inferior to those resulting from the MOS method (in those cases where it was possible to make a direct comparison of the two approaches).

4.2. Large Dataset. The experimental results given in this section are based on a set of 10794 structures drawn at random from the EINECS inventory, which is stored in the ECDIN databank.[16] The EINECS (European Inventory of Existing Chemical Substances) is an inventory of the chemical substances which were commercially available in the European Community between January 1971 and September 1981. All of the results discussed here and in the remainder of the paper relate to the AA method, since initial experiments with the MOS method showed that it could never process more than a few tens of the EINECS structures within the time available.

The AA method processed the dataset in 220 s, a rate of 49 structures per CPU second. Table II shows the variation of storage requirements when hyperstructures were generated using increasing numbers of input structures. This information is shown graphically in Figure 4. The graphs show that the storage requirements fall dramatically over the first few thousand structures and then begin to level off rapidly. The node storage decreases more rapidly than the arc storage and

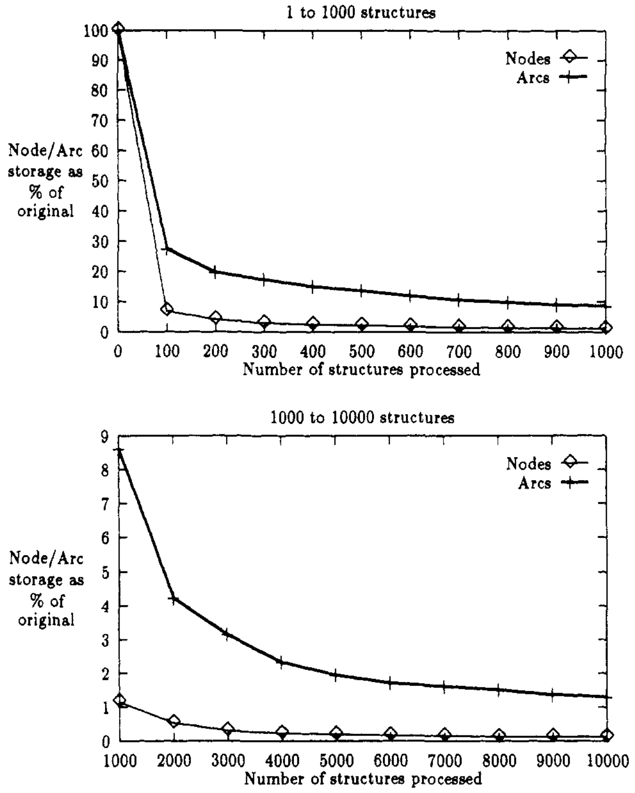
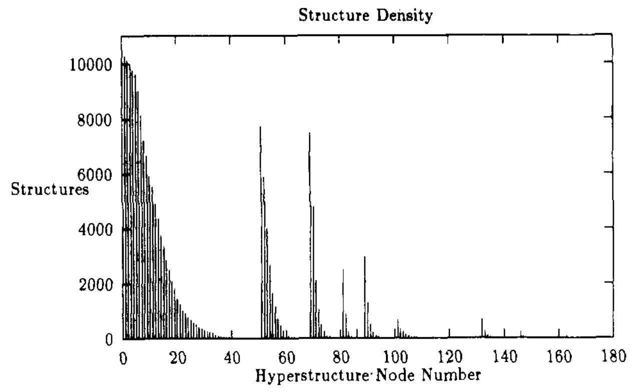


Figure 4. Variation in node and arc storage with hyperstructure size.

levels off more quickly; the former also reaches a lower level than the latter. The smaller storage requirement for the nodes is a direct result of the fact that the AA method takes no account of bonding patterns when matching nodes.

An estimate of the trends in compression as the dataset size increases may be calculated from figures given by Vladutz and Gould.[2] They estimated that a collection of 10^7 compounds could be stored in a hyperstructure of 360 nodes and 64 0000 edges if a typical input molecule has 25 non-hydrogen atoms and 30 bonds: these figures correspond to node and arc storages of 0.00014% and 0.0021%, respectively. The EINECS hyperstructure contained 222 nodes and 3813 edges. Thus, an approximately thousandfold increase in the size of a dataset (from the 10^4 structures used here to the 10^7 structures assumed by Vladutz and Gould) is expected to result in about a 0.6-fold increase in the number of nodes and a 17-fold increase in the number of edges (although it should be noted that the average molecular sizes assumed by Vladutz and Gould are slightly larger than those in the EINECS dataset, where the average number of both nodes and arcs was approximately 20).

Graphs are given in Figure 5 of the density of nodes and arcs in the global hyperstructure. The *X* axes of the graphs plot the number of the hyperstructure node. For these figures the nodes are ordered in terms of atomic type, with, for example, the first 51 nodes being carbon, the next 18 being nitrogen, and so on. This accounts for the wavelike form of the graph since the first nodes of each type will be mapped to first each time a new structure is added, and the later nodes only when a structure contains a large number of any one atom type. The structure density graph plots the number of structure nodes mapping onto each hyperstructure node. The connection density graph shows the number of connections each hyperstructure node has to others in the structure. The graphs show that the hypergraphs are very highly connected at their 'centers' with the majority of the structures' nodes mapping onto the central nodes. Further out, the complexity of the structure falls rapidly with only nodes of the larger



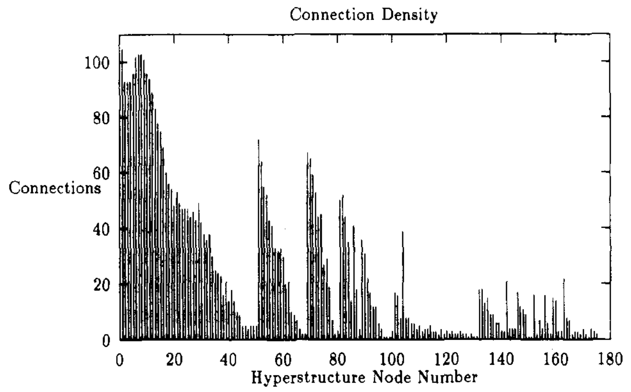


Figure 5. Characteristics of the EINECS global hyperstructure. Key: 1–51, C; 52–69, O; 70–81, N; 82–86, S; 87–89, S; 90–101, Cl; 102–132, F; 132–142, Br; 143–146, Si; 147–152, I; 153–176 Al, Cd, Sn, B, Pb, Hg, Se, Cr, Fe, Cu, Bi, Ti, Au, Rh, and Zn.

structures or those with less common atom types mapping to these areas.

As a hyperstructure grows, it would be expected that the first nodes for any one atom type would reach a high level of connectivity very rapidly and that the rate of growth would then tail off. However, the connectivity of the more outlying nodes would be expected to continue to grow at a more rapid rate than those near the core.

Vladutz and Gould[2] envisaged processing any given dataset into one single hyperstructure. This approach, which we have termed the *global hyperstructure* approach, will ensure that the maximum compression of data will occur but it is likely that problems may arise when attempting to search the complex, highly-connected graphs that would result from the processing of very large datasets. Specifically, a global hyperstructure has no obvious screening potential, unless some form of masking can be produced to exclude certain hyperstructure nodes from consideration, and substructure searching will thus require an exhaustive atom-by-atom search. An alternative approach, which we refer to as the *clustered-hyperstructure* approach, involves the division of a dataset into clusters of structurally-similar molecules, between which areas of commonality would be expected to be large, and then to construct a separate hyperstructure for each of these clusters. A database would thus be represented by a series of clustered hyperstructures, rather than by a single global hyperstructure. This approach would allow for the introduction of a screening step into a substructure search system for hyperstructures, which would seek to eliminate whole hyperstructures, and thus all of the structures they represent, from consideration. A set of screens to characterize a hyperstructure could use a combination of screens assigned to the constituent structures or could possibly be produced from substructural fragments of the hyperstructure itself.

The cluster analysis software used here is a program package, called SIMCLUS, that was developed for QSAR studies.[17] The structures in a dataset are characterized by the substructural fragments they contain. The fragment types used are augmented atoms with up to four neighbors, atom sequences of lengths 4–6, bond sequences of lengths 3–5, and ring descriptors that are assigned using the ring-perception algorithm developed by Downs.[18] The similarity measure used in SIMCLUS is the Tanimoto coefficient.[13] The similarity of each structure in the file is calculated with every other structure, and the resulting sets of nearest neighbors are passed to the cluster-generation module. A threshold may be applied at this point so that only structures with a similarity coefficient greater than the threshold are included in this nearest-neighbor list. This helps to avoid the problems that can arise where structures with a very low similarity coefficient become included in the nearest-neighbor lists. The clustering method used was that devised by Jarvis and Patrick,[13,19] in which clusters are built by considering the number of nearest neighbors common to a pair of structures. The clustering process is controlled by two parameters: K, which is the number of nearest neighbors to be examined for common structures, and K_{Min}, which is the minimum number of the K nearest neighbors that must be in common if the two structures are to be clustered together. The clustering criteria for two structures, X and Y, to be clustered together are as follows:

1. X is a nearest neighbor of Y,
2. Y is a nearest neighbor of X, and
3. X and Y have at least K_{Min} of their K nearest neighbors in common.

A program option allows for the relocation of singletons (structures which do not initially cluster with anything else) into the cluster that contains the most similar structure to the singleton.

Clustered hyperstructures were produced by running the Jarvis–Patrick method on the EINECS dataset with $K = 15$ and $K_{\mathrm{Min}} = 7$. The results obtained for these clusters without relocation and with thresholds of 0.0, 0.8, and 0.9 are given in Table III. With a threshold applied, a large number of structures have no nearest neighbors within that threshold and hence must be included in the file as singletons. Using a 0.8-threshold, 1576 structures had no nearest neighbors, and using a 0.9-threshold, 3873 structures, or just over one-third of the file, had no nearest neighbors. It is the inclusion of these that causes the node and arc percentage storage requirements to increase with increasing threshold, as shown in the table, and as a result the best storage savings were obtained using a 0.0 threshold. The dataset was processed at a rate of about 50 structures/s, thus suggesting that the AA method is sufficiently fast in operation to allow the generation of clustered hyperstructures in realistic amounts of time, given that it is possible to generate the initial set of clusters.

Table III. Hyperstructure Construction for EINECS Dataset Using Different Similarity Thresholds for Jarvis–Patrick Clustering Method

threshold	C_{Nodes}	C_{Arcs}
0.0	23.1	34.0
0.8	29.6	40.0
0.9	52.7	60.3

5. ABSOLUTE STORAGE REQUIREMENTS

An obvious way of storing a hyperstructure is by means of an extended connection table, in which each node has a linked list of connections to other nodes with which it is bonded. Each of these connections has associated with it a series of bond types forming the arc, which allows for the fact that any pair of hyperstructure nodes may be joined by any number of different types of bond. Each of these bond types then has linked to it a list of structural identifiers indicating the molecules from which it originated.

This approach has been rejected in favor of a slightly more complex, but more space-efficient, approach in which the hyperstructure is stored as an adjacency matrix in which each element of the matrix is itself an array, the dimension of which is equal to the maximum number of different bond types in use. Each element of each of these arrays stores a linked list of structure numbers from which the hyperstructure bond indicated by the matrix position originates, with the bond type indicated by the position in the array. An empty list in any position in any of the arrays indicates that a connection is not formed by that particular bond type in the current hyperstructure. Variations of this data structure include the use of a bit map, rather than a list of structure numbers, to indicate from which structures the bond originates and the use of a combined representation that uses both of the previously-mentioned approaches. All three techniques are considered further below.

We now estimate the absolute storage requirements of the hyperstructure for the EINECS data, using the adjacency-matrix approach. The original dataset contains 10 794 structures having 193 715 atoms and 203 757 arcs, if each arc is only stored once. Each node label requires 1 byte (thus allowing the representation of 128 different atom types), and each bond requires 1 byte to store the number of the other node to which it connects (assuming that none of the original structures contain more than 128 atoms) and 3 bits to store the bond type (since eight bond types were defined in these experiments). If a further 2 bytes are used with each structure to store the structure identification number in the range 1–10794, then the total storage required for the data is 495.4 KB, with an average of 45 bytes per compound.

The hyperstructure has 222 nodes and a total of 3813 structure lists and is stored in three components. The basic adjacency matrix for a hyperstructure containing N nodes contains N^2 elements. Each of these elements contains 1 bit for each of the possible bond types in the dataset, with the Ith of these bits in the XYth element being set if the Ith type of bond exists between the Xth and Yth atoms in the hyperstructure. The (nonredundant) storage requirement for this matrix is 24.6 KB. The atom-type label requires 1 byte for each node, giving a total of 0.2 KB. The largest part of the hyperstructure storage is the set of structure lists, which, as mentioned above, can be implemented in three different ways:

1. The simplest approach is to allocate a dedicated bit string for each bond, with the setting of the Ith bit denoting the presence of that bond in the Ith structure. In this case, a 10 794-bit string will be required for each of the 3813 bonds, giving a storage requirement of 5145 KB.
2. The list approach uses 2-byte integers (each of which can store an integer in the range 0–65 535). The 3813 bond lists contain a total of 203 757 such identifiers, i.e., one for each bond in the original dataset, requiring a total of 407.5 KB. If a proportion of the structural identifier lists exceeded half the possible maximum length, i.e., if a hyperstructure bond originated from more than 5397 structures, those lists exceeding 5397 structures could instead be stored as a list of those structures *not* forming the bond. This would require 1 extra bit per list to indicate which type each list is, thus limiting the maximum length of any list to half the total number of structures in the file. These extra check bits were not required in the case of the EINECS hyperstructure, where the longest list contained 2753 structural identifiers; the bits would be necessary for hyperstructures that represented millions of structures, where certain bonds would be expected to originate from a high proportion of all structures.3.
3. The combined approach utilizes both of the previous approaches. A bit string of 10 794 bits requires approximately the same amount of storage as 675 2-byte integers; any structure list containing more than 675 identifiers is thus more efficiently stored as a bit string, while any containing less than 675 is more efficiently stored as a list of integers. Analysis of the structure lists for the EINECS hyperstructure shows that 3750 of the 3813 bond lists contain less than 675 identifiers and that 63 contain 675 or more. The storage requirements for the 3750 short lists, which contain a total of 110 473 integers, is 220.9 KB, while the 63 10794-element bit strings require 85.0 KB. An additional cost of 1 bit per list will be incurred to indicate which type of storage is being used, requiring a further 0.5 KB for the 3813 lists. The structure lists thus require, in toto, 306.4 KB, which is 6% of that required if all storage is as bit strings and 75% of that required if all storage is as integer lists. The use of a combined representation has been discussed previously by Reddaway[20] in the context of an inverted file system for a large text database.

Using the last of these three approaches, the total storage requirement of the hyperstructure is thus 331.2 KB, which is 67% of the 495.4 KB required for the original data. It is anticipated that these storage savings might be expected to increase with substantially larger files, especially as the average length of each structural-identifier list grows, making the use of bit maps to store the lists more effective. It should be remembered that for a ten-million structure file, structural identifiers will have to be stored as 4-byte rather than 2-byte integers; the bit-map approach will thus be more efficient for any list containing more than 312 000 integers.

6. ATOM-BY-ATOM SEARCHING OF HYPERSTRUCTURES

In this section, we report atom-by-atom searching experiments that were carried out using the hyperstructure and non-hyperstructure versions of the EINECS dataset. The atom-by-atom procedure was based on Ullmann's subgraph-isomorphism algorithm,[21] which has been shown to be well-suited to atom-by-atom searching using both 2D and 3D connection tables.[22,23] Ullmann's algorithm was used to search the file of individual structures and, in suitably-modified forms, for searching both the global-hyperstructure and clustered-hyperstructure versions of the dataset. The algorithm was implemented in C and run on an Evans and Sutherland ESV 3 workstation under UNIX.

A set of 27 substructural queries was used to search the EINECS dataset in the following ways:

1. The conventional, non-hyperstructure searches, which are referred to subsequently as the *non-H searches*. Two types of non-H search were carried out:
   a. The atom-by-atom search of each structure was terminated as soon as a subgraph isomorphism had been identified (or as soon as it was certain that no such isomorphism was present). This is the way that an atom-by-atom search is implemented

Table IV. Characteristics of Various Versions of EINECS Dataset That Were Searched in Atom-by-Atom Experiments

K_{Min}	mean size of clusters	largest cluster	C_{Nodes}	C_{Arcs}	max nodes per hyperstructure	max arcs per hyperstructure
non-H	1.0	1	100.0	100.0	69	74
19	1.8	21	57.4	64.5	81	173
15	3.1	135	36.5	46.5	81	320
9	4.0	1 392	27.9	36.2	129	770
5	4.1	1 906	26.7	34.1	138	923
1	4.2	2 102	25.9	32.9	140	1043
global	10 794	10 794	0.1	1.3	222	3813

Table V. Median Search Times (in CPU s) on ESV 3 Workstation and Median Number of Subgraph Matches Identified, Averaged over 27 Atom-by-Atom Searches of EINECS Dataset

K_{min}	median search time	median no. of subgraphs
non-H1	39.9	122
non-H2	46.5	318
19	34.9	240
15	33.6	199
9	24.9	184
5	23.9	184
1	24.6	184
global	302.0	122

in a conventional substructure-searching system and is referred to subsequently as a *non-H1 search.*

b. The atom-by-atom search of each structure was terminated only when all of the possible subgraph isomorphisms had been identified. This approach, which is referred to subsequently as a *non-H2 search*, more closely resembles the processing of a hyperstructure, where all of the isomorphisms must be identified if the search is not to miss some of the structures that contain the query substructure.

2. The searches of the global and clustered hyperstructures. The clustered-hyperstructure searches were carried out using Jarvis–Patrick clusters in which K was set to 20 and in which K_{Min} was given the values 1, 5, 9, 15, and 19. A threshold similarity of 0.8 was used to ensure a high degree of homogeneity between the structures in each cluster; no relocation was used, and thus all singleton molecules, i.e., those having nearest neighbors for which the similarity was less than the threshold value, remained in the clustered file as single structures. The variation of K_{Min} created a series of clusters with differing mean cluster sizes and numbers of structures in the largest cluster, as is illustrated in Table IV, which details the various characterizations of the EINECS dataset that were used in these experiments.

The median execution times for each of these types of search, when averaged over the set of the 27 substructural queries, are listed in Table V, together with the median numbers of subgraph isomorphisms that were identified in the course of these searches. The queries themselves are shown in Figure 6: it will be seen that many of them are very simple in character, but this ensures the retrieval of at least some structures from the search file used here. It should be emphasized that the searches have not used any type of screening procedure, so that all of the molecules in the non-hyperstructure versions of the dataset have been submitted to the subgraph-isomorphism algorithm.

An inspection of the results in Table V shows that all of the clustered hyperstructure searches were faster than the conventional, non-H searches, although the difference is quite small with the clustered hyperstructures that were generated using the largest values of the K_{Min} parameter (these values corresponding to the smallest clusters, as is evidenced by the figures in Table IV). The global-hyperstructure searches are far more time-consuming than the clustered-hyperstructure searches; in fact, it must be remembered that these are median figures, and some of the global-hyperstructure searches for queries involving complex ring systems were 1 or 2 orders of magnitude slower than the figures listed in the table. That said, there were several queries where the global hyperstructures gave the fastest searches; this tended to be the case with acyclic queries. The clustered searches were fastest with the more specific queries, e.g., infrequently-occurring ring systems or queries containing several heteroatoms, while the serial searches, conversely, were fastest with the more general queries. As would be expected the non-H1 searches were consistently faster than the non-H2 searches.

It is, perhaps, rather surprising that the hyperstructures perform so well, given that the non-H searches have several a priori advantages.

1. The hyperstructures need to be searched to completion, whereas the individual molecules in the non-H1 searches only need to be searched until the first occurrence of the query subgraph has been identified.
2. Hyperstructures are much larger and more highly connected than single molecules, with a resultant increase in the complexity of the search trees that must be explored when compared to those of even the most complex single molecules. Particular problems are encountered with queries that have any degree of symmetry, since there will then be very many different orientations of the query onto each occurrence of the sought subgraph in the hyperstructure.
3. The hyperstructure searches involve the elimination of the ghost substructures that have been referred to in section 1, i.e., substructures that occur in the hyperstructure but that are made up from bonds occurring in different constituent structures.

Given the presence of these factors, it is most gratifying that the results support the contention that an atom-by-atom search of a hyperstructure can be more efficient than comparable searches of that hyperstructure's constituent molecules. Further support is given by the subgraph results in Table V, where the median number of subgraph isomorphisms for the hyperstructure searches is consistently less than the corresponding number for the non-H2 searches, i.e., those where the atom-by-atom search is continued until all of the possible subgraph isomorphisms have been identified. This is particularly so, as would be expected, in the case of the global-hyperstructure searches (although these are by far the most time-consuming). It may thus be concluded that the hyperstructure-generation routines have been at least partially successful in superimposing multiple occurrences of the same substructural features.

An examination of the numbers of subgraph isomorphisms identified for individual queries (rather than the average figures listed in the table and discussed so far) shows that there were nine cases in which both the global and the clustered searches

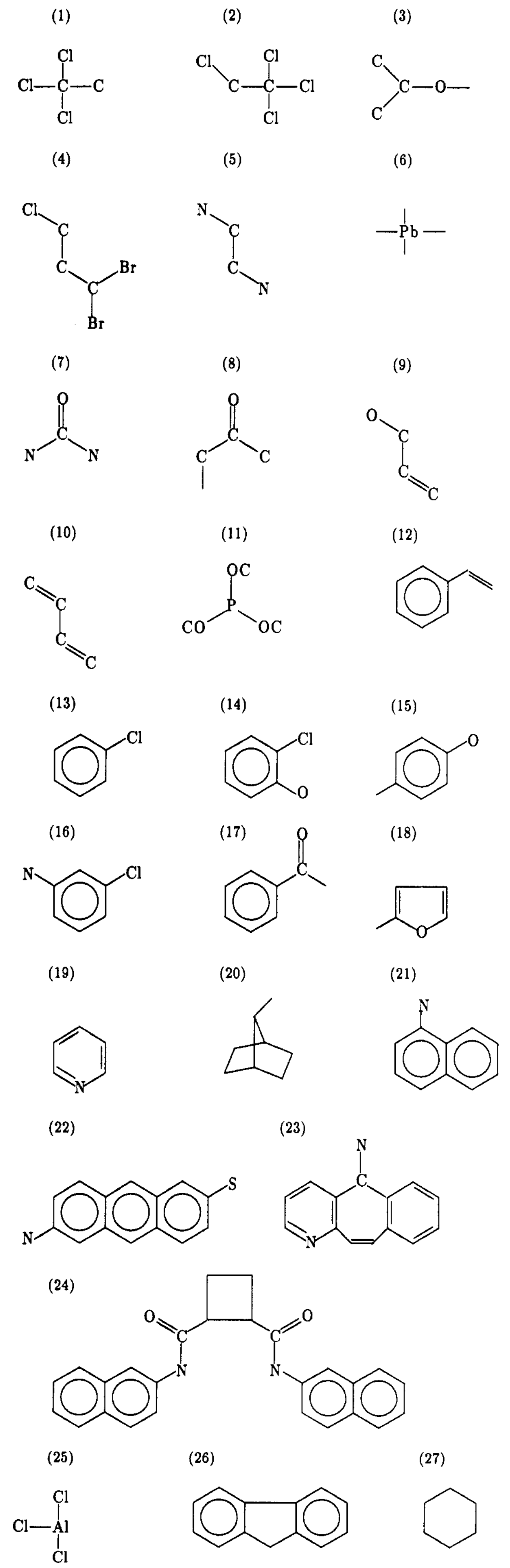


Figure 6. Substructural queries.

identified less subgraph isomorphisms than either of the types of non-H search. These searches provide the best support for the hyperstructure concept. There were eight queries, all involving only small numbers of hits, where all of the types of searches retrieved approximately equal numbers of isomorphisms. The non-H searches retrieved the fewest isomorphisms in the remaining 10 searches, although even here, there were several occasions when the hyperstructure searches retrieved fewer isomorphisms than the non-H2 searches. There did not appear to be any obvious correlation, for the set of searches studied here, between the absolute or relative run times and the numbers of subgraphs retrieved.

7. CONCLUSION

Hyperstructures have been suggested as a way of representing the structural information in a set of connection tables in a nonredundant form, with consequent advantages in storage and processing costs. This work has shown that hyperstructures do, indeed, provide an extremely compact representation of the connectivity of a series of chemical structures, achieving node and arc compressions of 1000- and 100-fold, respectively, in a file of 10K structures. However, the overall degree of compression that can be achieved by using the hyperstructure approach is far less than is indicated by these figures: for the EINECS data studied here, an overall compression of approximately one-third has been achieved in the total amount of data that needs to be stored, a figure that might be expected to increase as the number of covered structures increases.

Of the two methods of construction studied, only the AA method was found to be viable for large and general datasets. While the results obtained using the MOS method suggest that this would produce more compact hyperstructures, it has been shown to be totally unfeasible for practical use. Using the AA method, both global and clustered hyperstructures can be produced efficiently. The global approach provides a much greater degree of compression, but the hyperstructures are extremely large and much more highly-connected than the clustered hyperstructures. These characteristics are reflected in the substructure-searching experiments, where the run times are substantially greater for the global hyperstructure searches than for the clustered hyperstructure searches.

The run time and retrieved-subgraph results obtained here support the belief that the hyperstructure approach has at least the potential to increase the efficiency of substructure searching, although it is not clear what improvements in performance, if any, would be achieved if this approach was to be implemented in practice on a file of nontrivial size. We are currently investigating techniques for screening hyperstructures, with the aim both of eliminating entire hyperstructures from the atom-by-atom search (in the case of the clustered hyperstructures) and of eliminating parts of hyperstructures (in the case of both types of hyperstructure).

ACKNOWLEDGMENT

We thank the European Community Joint Research Centre for the provision of the EINECS data, Barnard Chemical Information Ltd. for the provision of the SIMCLUS software, Tripos Associates for the provision of the Evans and Sutherland workstation, and Bill Fisanick for comments on an earlier draft of this manuscript. The work is funded by a Science and Engineering Research Council CASE Studentship in collaboration with ORAC Ltd.

REFERENCES AND NOTES

(1) Ash, J. E.; Warr, W. A.; Willett, P., Eds. *Chemical Structure Systems*; Ellis Horwood: Chichester, 1991.

(2) Vladutz, G.; Gould, S. R. Joint Compound/Reaction Storage and Retrieval and Possibilities of a Hyperstructure-Based Solution. In *Chemical Structures. The International Language of Chemistry*; Warr, W. A., Ed.; Springer Verlag: Berlin, 1988, pp 371–384.
(3) Fredkin, E. Trie Memory. *Commun. ACM* **1962**, *3*, 490–499.
(4) Willett, P. *Three-Dimensional Chemical Structure Handling*; Research Studies Press: Taunton, 1991.
(5) Fisanick, W. The Chemical Abstracts Service Generic Chemical (Markush) Structure Storage and Retrieval Capability. 1. Basic Concepts. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 145–155.
(6) McGregor, J. J. Backtrack Search Algorithms and the Maximal Common Subgraph Problem. *Software—Pract. Exp.* **1982**, *12*, 23–34.
(7) Read, R. C.; Corneil, D. G. The Graph Isomorphism Disease. *J. Graph Theory* **1977**, *1*, 339–363.
(8) Morgan, H. L. The Generation of a Unique Machine-Description for Chemical Structures—a Technique Developed at Chemical Abstracts Service. *J. Chem. Doc.* **1965**, *5*, 107–113.
(9) Lynch, M. F.; Willett, P. The Automatic Detection of Chemical Reaction Sites. *J. Chem. Inf. Comput. Sci.* **1978**, *18*, 154–159.
(10) Randić, M. Fragment Search in Acyclic Structures. *J. Chem. Inf. Comput. Sci.* **1978**, *19*, 101–107.
(11) Randić, M.; Wilkins, C. L. Graph-Based Fragment Searches in Polycyclic Structures. *J. Chem. Inf. Comput. Sci.* **1979**, *19*, 23–31.
(12) Salton, G.; McGill, M. J. *Introduction to Modern Information Retrieval*; McGraw-Hill: New York, 1983.
(13) Willett, P. *Similarity and Clustering in Chemical Information Systems*; Research Studies Press: Letchworth, 1987.
(14) Brown, R. D. A Hyperstructure Model For Chemical Structure Handling M.Sc. Dissertation, University of Sheffield, 1990.
(15) Okada, T.; Wipke, W. T. CLUSMOL: a System for the Conceptual Clustering of Molecules. *Tetrahedron Comput. Methodol.* **1989**, *2*, 249–264.
(16) Norager, O. ECDIN, Environmental Chemicals Data and Information Network. In *Chemical Structures. The International Language of Chemistry*; Warr, W. A., Ed.; Springer Verlag: Berlin, 1988; pp 195–209.
(17) Downs, G. M.; Walsh, P. T.; Booth, A. M. Similarity and Clustering of Chemical Structures for Property Prediction. Presented at the 2nd International Workshop on Computer Chemistry: Structure Property Relations, Germany, Mersburg, Oct 2–4, 1990.
(18) Downs, G. M. Computer Storage and Retrieval of Generic Structures in Patents: Ring Perception and Screening to Extend the Search Capabilities. Ph.D. Thesis, University of Sheffield, 1988.
(19) Jarvis, R. A.; Patrick, E. A. Clustering Using a Similarity Measure Based on Shared Nearest Neighbours. *IEEE Trans. Comput.* **1973**, *C-22*, 1025–1034.
(20) Reddaway, S. F. High Speed Text Retrieval from Large Databases on a Massively Parallel Processor. *Inf. Process. Manage.* **1991**, *27*, 311–316.
(21) Ullmann, J. R. An Algorithm for Subgraph Isomorphism. *J. ACM* **1976**, *23*, 31–42.
(22) Brint, A. T.; Willett, P. Pharmacophoric Pattern Matching in Files of Three-Dimensional Chemical Structures: Comparison of Geometric Searching Algorithms. *J. Mol. Graphics* **1987**, *5*, 49–56.
(23) Downs, G. M.; Lynch, M. F.; Willett, P.; Manson, G. A.; Wilson, G. A. Transputer Implementations of Chemical Substructure Searching Algorithms. *Tetrahedron Comput. Methodol.* **1988**, *1*, 207–217.