Hyperstructure Model for Chemical Structure Handling: Techniques for Substructure Searching[†]

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

Krebs Institute for Biomolecular Research, Department of Information Studies, University of Sheffield, Western Bank, Sheffield, S10 2TN, U.K.

Received May 7, 1993[®]

A hyperstructure is a novel representation for a set of 2-D chemical structures that has been suggested for increasing the speed of substructure searches, when compared with a conventional sequential search. This paper describes three developments that are aimed at increasing the efficiency of hyperstructure searching. The first involves the use of a genetic algorithm to detect the maximal overlap between structures and hence to decrease the size of the hyperstructure that is used to represent a set of structures; the second is to introduce a screening system for one form of hyperstructure; and the third is to implement the atom-by-atom search algorithm on massively-parallel hardware. The first two developments were shown to increase the speed of search considerably, while the last was much less successful.

1. BACKGROUND

Vladutz and Gould[1] have described a new method for the representation and storage of two-dimensional (2-D) chemical structures based on a file organization known as a *hyperstructure*. This is a pseudomolecule formed by the superimposition of sets of molecules in such a way that areas of structural commonality are stored only once, and Vladutz and Gould suggest that this characteristic may enable substructure searches to be carried out on hyperstructures with great efficiency. The potential for increased search speed arises from the fact that substructures common to more than one database structure occur in the hyperstructure only once, so that the identification of a single occurrence of the query substructure in the hyperstructure may correspond to the identification of that substructure in large numbers of database structures. Related approaches have been discussed by several authors.[2–4]

Hyperstructure generation is a stepwise process in which molecules are added into an existing hyperstructure one at a time. The process requires that the maximum-possible overlap is identified between atoms and bonds in the input structure and in the current hyperstructure. We are engaged in a project to evaluate the effectiveness of the hyperstructure approach to the representation and substructure searching of databases of 2-D structures and have recently investigated two methods for the generation of hyperstructures.[5] The first, based on the use of a *maximal overlap set* (MOS) algorithm, which is very similar to a maximal common subgraph algorithm, proved to be too demanding of computational resources for practical use. The second, known as *atom assignment* (AA), was much faster but less precise, in that it identified matching atoms without taking any account of the patterns of bonding in the two structures. This was thought to have a detrimental effect on the data compression and subsequent substructure search times compared to those which would have been expected from the equivalent hyperstructure produced by the MOS method (if this method had, in fact, been feasible). Evidence that this was the case was suggested by some experiments on very small data sets for which both the MOS and AA methods could produce hyperstructures.

Two file organizations were described for hyperstructures.[5] The first was a *global hyperstructure* in which all structures in a given data set were overlaid to produce a single hyperstructure. The second, *cluster hyperstructures*, produced a series of hyperstructures from a data set which had previously been partitioned into clusters by the Jarvis–Patrick clustering method.[6,7] Substructure searching of both types of hyperstructure was effected using a modified form of Ullmann's subgraph isomorphism algorithm.[8] Experiment showed that the substructure search of hyperstructures was more efficient than the equivalent sequential search of the original data set for 18 of the 27 queries tested.

This paper reports on three developments of our methods that are aimed at increasing the efficiency of substructure searching. The first produces a better hyperstructure during the generation process, while the second and third concentrate on the substructure search process itself. Section 2 describes the use of a genetic algorithm for MOS detection in an effort to generate hyperstructures in which the input structures map onto one another more accurately than is possible using the AA method. Sections 3 and 4 discuss the implementation of a screening system for cluster hyperstructures and the use of massively-parallel hardware for substructure searching, respectively; and the paper ends with a summary of the main conclusions that can be drawn from this study. With the exception of the parallel-processing experiments described in section 4, all of the programs here were implemented on an Evans and Sutherland ESV-30 UNIX workstation, and all CPU times refer to this machine. Our tests used a data set of 10 794 diverse structures drawn at random from the EINECS (European Inventory of Existing Chemical Substances) database, which lists all chemicals commercially available in the European Community between 1971 and 1981.[9]

2. HYPERSTRUCTURE GENERATION USING A GENETIC ALGORITHM

The comparison of the MOS and AA generation methods mentioned previously showed that one means of improving the speed of the substructure search for hyperstructures would be to produce a better mapping between the current hyperstructure and successive input structures than that produced by the AA method. This would result in a less highly connected

* To whom all correspondence should be addressed.
† Presented at the Third International Conference on Chemical Structures, 6th-10th June 1993.
® Abstract published in *Advance ACS Abstracts*, January 15, 1994.

and more efficiently searchable hyperstructure. An improved method must therefore take into account the bonding patterns of the input structure and hyperstructure but be much faster in execution than the MOS method.

The problem of producing a mapping between two structures may be expressed in terms of maximizing the number of bonds present in the (possibly-disconnected) common subgraph of an input structure and the current hyperstructure. The genetic-algorithm (GA) paradigm is based upon the processes which occur in biological evolution and has been shown to be able to produce good solutions to a wide range of complex optimization problems, particularly in cases where deterministic algorithms have been unsuccessful.[10,11] The use of GAs for processing chemical structures was first described by Fountain[12] in the context of synthesis design; Brown *et al.*[13] provide a full description of a GA that has been developed for the generation of hyperstructures by means of disconnected-MOS detection and of the parameters that are associated with the use of this algorithm: we now discuss its application for this purpose.

2.1. Optimizing the Genetic Algorithm. Initial testing showed that the GA method of hyperstructure construction became increasingly more efficient than the AA method when the minimum cluster size of clusters produced from the EINECS data set was increased, when the structures were ordered before input, and when candidate lists were used. These strategies are discussed further below.

The EINECS data set is very diverse, so that clustering was very uneven with many single-structure and double-structure clusters. The AA method is likely to produce a better approximation to the disconnected MOS when the current hyperstructure is very small, as will be the case for clusters containing a very few structures. In such cases, of course, the resulting file will contain a very large number of small clusters and be slow to search. A better comparison is obtained of the true relative efficiency of the two generation methods when a minimum cluster size of five is imposed, and all of the cluster-hyperstructure results reported below use this set of clusters.

We have noted previously[5] that a set of structures can be preprocessed to determine the best order in which they should be added to the hyperstructure. It was found that the best order for hyperstructure assembly was to start with the largest structure, follow this with the one with the greatest similarity to it, continuing to add the remaining structure with the highest similarity to the last added until the whole cluster or data set has been processed. It was found that use of this preprocessing allowed the GA to produce a hyperstructure more quickly than without it, and also marginally improved the substructure search times of the resulting hyperstructure(s).

We described a method to produce a candidate list of hyperstructure nodes for each input structure node during the mapping process of one structure onto the current hyperstructure.[5] These candidate lists have been used to produce *greedy chromosomes* which have a better-than-average fitness. Details of the method are given by Brown *et al.*[13] The initial, randomly-generated population was seeded with these better-than-average chromosomes. It was found that the use of this method with the cluster hyperstructures allowed a more compact hyperstructure to be generated more quickly than without its use and also resulted in a speedup in substructure searching. However, the generation and search times of the global hyperstructures were significantly impaired when the greedy-chromosome approach was used, so the normal approach was used for the global-hyperstructure experiments and the greedy-chromosome approach for the cluster-hyperstructure experiments. GAs are highly nonlinear in character, and it is thus often difficult to predict or to explain their behavior, as is the case with the marked differences observed here.

Table 1. Variation in Search Times, in CPU Seconds on an Evans and Sutherland ESV-30 Workstation, for Substructure Searches of Cluster Hyperstructures Using 50 Invocations of the Genetic Algorithm

query no.	mean	std dev	query no.	mean	std dev
1	3.5	0.39	12	40.1	7.48
2	3.1	0.49	13	295.4	16.66
3	20.6	1.38	14	40.1	3.14
4	3.2	0.69	15	167.6	10.11
5	5.6	0.54	16	20.0	1.86
7	2.0	0.28	17	102.2	13.90
8	12.7	1.15	18	2.3	0.14
9	3.5	0.69	19	16.9	1.70
10	3.4	0.80	21	386.5	28.38
11	11.6	1.38	27	58.7	5.69

2.2. Variation in the Genetic Algorithm. One important feature of any GA is that it is nondeterministic,[10,11] so any mapping that the GA produces between a given input structure and the current hyperstructure may be different if the algorithm is rerun. During the generation of a large hyperstructure, therefore, the potential number of different mappings at each stage will mean that two very different hyperstructures may be produced from the same data simply by rerunning the algorithm. It was thus necessary to establish an estimate of the variability introduced into the generation and substructure-search processes by the nondeterministic nature of the GA.

This variability was investigated using a subset of the EINECS data set with 6392 structures in 516 clusters, all of which contained at least 5 structures. This data set was processed into hyperstructures 50 times and then searched for each of 20 substructural queries (these being those queries from the set of 27 queries used in our previous experiments[5] that yielded at least some hits in the set of 6392 structures used for this experiment). This resulted in a sample of 50 construction times, and 50 search times for each of the 20 query substructures. Since the same data were used each time under the same conditions, these samples reflect the populations of possible outcomes arising from the variability of the GA.

The mean construction time was 7409.2 CPU s on the ESV-30 workstation, with a standard deviation of 175.4 CPU s; the mean number of arcs in the resulting hyperstructures was 19 242.0, with a standard deviation of 76.0. The means and standard deviations of the search times are listed in Table 1, where the query numbers relate to the substructure diagrams given in ref 5. The 50 measurements for the construction time and for each of the substructure search times may be treated as samples drawn from separate populations. Frequency plots of these distributions show them to have the "bell-shape" typical of a normal distribution.[14]

The standard deviation is just 2.4% of the mean construction time, and there is a still smaller variation (under 0.4%) in the overall number of arcs that result and, therefore, in the overall data compression. However, the standard deviations for the search times in Table 1 show that the variation in substructure-search time for any given query is much larger, since some of the standard deviations were in excess of 10% of the mean search times.

These results show that different sets of hyperstructures built from the same data and having overall arc compressions

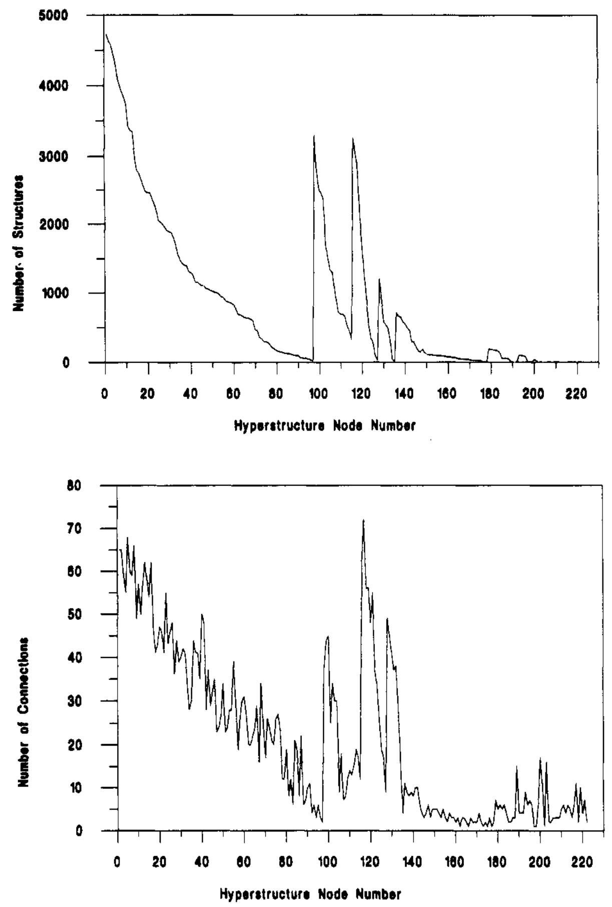

Figure 1. Variation of structure and connection density with hyperstructure node number. Hyperstructure node atomic types are as follows: 1–96 C; 97–115 O, 116–127 N; 128–132 S; 133–135 P; 136–147 Cl; 148–178 F; 179–188 Br; 189–192 Si; 193–198 I; 199–222 Al, As, Au, B, Bi, Cd, Cr, Cu, Eu, Fe, Hg, Mg, Pb, Rh, Sb, Se, Sn, Ti, Zn.

within 1% of each other can have widely-differing substructure search times for any given query. This suggests, firstly, that very different hyperstructures are being built each time by the GA method and, secondly, that the actual patterns of overlap within the hyperstructure are more important in determining the efficiency of subsequent substructure searches than the overall data compression. The latter finding, coupled with the plummeting costs of memory and backing storage, rather undermines one of the rationales for the use of hyperstructures that were put forward by Vladutz and Gould,[1] *viz.*, its potential for lowering storage requirements.

2.3. Comparison of the Genetic-Algorithm and Atom-Assignment Methods. Once the inherent variation of the GA had been investigated, it was possible to compare the performance of the GA and AA methods, in terms of both the outcome of the generation and the subsequent substructure-search times; the performance of the two approaches in terms of the actual structures retrieved is, of course, precisely the same.

The AA method was found previously to be able to generate hyperstructures in 0.01–0.02 s/structure.[5] The range of processing times for adding a structure to the global hyperstructure of the 10794 EINECS structures using the GA method was between 0.1 and 252.9 s, with a mean build time of 6.6 s/structure and a standard deviation of 17.5 s/structure. The AA method is thus 2–3 orders of magnitude faster than the GA method. However, the resulting hyperstructures are far less compact than when the GA is used. Thus, the 222 nodes in the global hyperstructure for the EINECS data set

Table 2. Comparison of Search Times, in CPU Seconds on an Evans and Sutherland ESV-30 Workstation, for Substructure Searches of GA and AA Cluster and Global Hyperstructures

	cluster hyperstructures			global hyperstructures		
query no.	AA search time	GA search time	speedup	AA search time	GA search time	speedup
1	5.5	4.0	1.4	14.3	6.0	2.4
2	6.9	3.0	2.3	12.9	4.0	3.3
3	57.9	21.8	2.7	302.0	238.3	1.3
4	4.1	3.1	1.3	10.3	5.5	1.9
5	11.2	5.3	2.1	19.6	13.8	1.5
6	2.2	2.1	1.0	1.2	0.9	1.4
7	4.1	2.0	2.1	4.8	2.3	2.1
8	41.9	12.1	3.5	305.8	132.5	2.3
9	4.7	3.3	1.4	26.7	15.3	1.8
10	4.8	3.0	1.6	11.6	12.4	0.9
11	21.3	13.9	1.6	152.9	139.0	1.1
12	176.4	36.1	4.9	12737.6	9544.4	1.3
13	963.9	297.6	3.2	21035.3	12153.5	1.7
14	130.8	42.5	3.1	3940.5	3261.5	1.2
15	659.0	167.3	3.9	23535.8	23351.6	1.0
16	43.1	17.9	2.4	10479.2	7295.0	1.4
17	317.5	86.3	3.7	17107.6	25088.7	0.7
18	2.4	2.3	1.0	12.8	11.5	1.2
19	53.0	16.3	3.3	1779.4	1431.0	1.3
20	2.9	2.7	1.1	138.9	117.4	1.2
21	931.3	403.3	2.3	30391.9	34728.8	0.9
22	4.6	5.0	0.9	1610.7	16128.9	0.1
23	4.7	4.7	1.0	232.1	271.5	0.9
24	17.8	18.8	0.9	17382.3	10345.0	1.7
25	0.2	0.2	1.0	0.1	0.1	1.0
26	154.2	72.6	2.0	1213.9	3654.0	0.3
27	172.2	55.6	3.1	6207.4	4334.6	1.4

contained 1966 arcs when generated using the GA method as against 3813 when generated using the AA method (i.e., there were no less than 94% more arcs in the AA hyperstructure). The corresponding figures for the 13 601 nodes in the cluster hyperstructures were 16 854 and 28 753 (so that the AA hyperstructures here contained 71% more arcs).

We reported previously[5] on the number, or density, of structure atoms mapping onto each hyperstructure mode and on the complexity of the hyperstructure in terms of the numbers of connections emanating from each hyperstructure node. In both cases it was shown that the greatest density is on the first nodes of each atom type, with a rapid decline over later nodes of the same type. Similar plots are given in Figure 1 for the GA global hyperstructure. These show a similar pattern occurring, but the slopes are less steep and the average structure and connection density considerably lower, when compared with the analogous graphs for the AA global hyperstructure. For example, the first carbon atom here has only half the number of structure atoms mapped to it when compared with the same carbon in the AA hyperstructure, and it has only three-quarters of the number of connections to other hyperstructure nodes. This means that the input structures are being mapped more evenly across the hyperstructure than was the case with the AA method. The decline in density over a given atom type remains, however, since the higher-numbered nodes of any one type will not be created until some way through the construction process and so will not be available for mapping to the earlier input structures.

Table 2 shows the GA and AA substructure-search times for the cluster and global hyperstructures of the full EINECS data set, where GA search (or AA search) refers to the use of the Ullmann algorithm to search hyperstructures built with the GA method (or the AA method). The cluster hyperstructures used in these experiments were those obtained with a value of 15 for the K_{min} parameter in the Jarvis–Patrick clustering method and with a minimum cluster size of five compounds.[6] This parameter specifies the number of nearest-neighbors which two structures must have in common if they are to cluster together. It will be seen that the GA search times are usually less than the AA search time (though it should be remembered that the GA substructure-search times vary due to the nondeterministic nature of the algorithm). A speedup is given for GA search over AA search which is calculated as the ratio of the AA search time to the GA search time. The results show a median speedup of 2.1 times for cluster hyperstructures and of 1.3 times for global hyperstructure search, with peak speedups of 4.9 and 3.3 times, respectively. In the case of the cluster hyperstructures, the better speedups were obtained for the more complex, time-consuming queries; there seems to be no particular pattern to the speedups observed with the global hyperstructure. A one-tailed Sign test[15] on the speedups in Table 2 shows that the GA method is significantly faster than the AA method ($p < 0.001$ for cluster hyperstructures and $p < 0.004$ for global hyperstructures).

The main criterion for the success of the hyperstructure method is that the resulting substructure search should be faster than the equivalent sequential search.[1,5] In the cluster-hyperstructure searches, hyperstructure search was faster than sequential search for 18 of the test set of 27 substructure queries using AA hyperstructures, and 23 of the queries using GA hyperstructures. In the global-hyperstructure searches, hyperstructure searching was faster for 8 of the queries using AA hyperstructures and 9 of the queries using GA hyperstructures. These results suggest that the GA hyperstructures, particularly the cluster hyperstructures, can allow substructure searching to be carried out more efficiently than a sequential search of a set of compounds in some cases.

2.4. Use of the Genetic Algorithm with Larger Data Sets. The results in the previous section demonstrate that the GA produces more satisfactory hyperstructures than the AA method. However, to be useful it must be able to process much larger data sets than the one used here. The AA procedure is simple and so will be unaffected by the size and complexity of the hyperstructure onto which it must map structures.

Experiments with the order of processing of the EINECS data[16] show that the mean time to map a given structure to a growing global hyperstructure depends on the size of that input structure but decreases with the increasing number of structures already contained in the hyperstructure (and hence with the latter's size and complexity). The reason for this behavior is that as the hyperstructure grows, it becomes increasingly likely that the next input structure will already be entirely contained within the hyperstructure as a constituent substructure. In this case the fitness function of the genetic algorithm will be able to reach its maximum value and there will be an increased likelihood that the GA will not have to run to its maximum number of operations,[13] thus reducing the mean mapping times. This finding suggests that the GA method should be applicable to much larger data sets than that studied here, since the mapping time should depend predominantly on the size of the input structures.

3. SCREENING CLUSTER HYPERSTRUCTURES

In conventional substructure search systems, such as those used in CAS Online, MACCS, and DARC,[17] searching is a two-stage process. The initial screening search rapidly eliminates the majority of the database from further consideration, and the time-consuming atom-by-atom search then identifies the hits among those structures that pass screening.

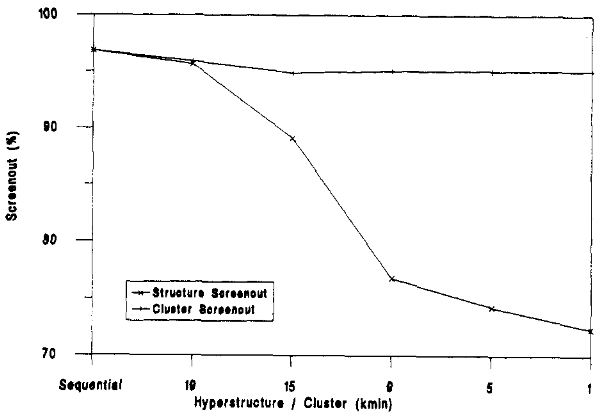

Figure 2. Variation in structure and cluster screenout with increasing average cluster size.

A screening system has been developed for cluster hyperstructures. This seeks to eliminate individual hyperstructures, and therefore all of their constituent structures, from consideration prior to the atom-by-atom search. The procedure is as follows:

1. A screen dictionary is used to characterize each structure in the dataset in terms of the presence of screens from that dictionary. The screen occurrences are recorded in a bit string. The dictionary used here is a subset of the CAS Online screen dictionary with the addition of extra ring screens developed by Downs.[18]
2. The data set is clustered using the Jarvis–Patrick clustering method, which uses the screens to calculate the intermolecular similarities.
3. A set of screens for a single cluster hyperstructure is produced by the logical union (i.e., the OR operation) of all the screens contained in each constituent structure.
4. The substructural query is characterized using the same screen dictionary.
5. The Ullmann algorithm is then applied only to those hyperstructures that contain all of the screens that are contained in the query.

The use of superimposed screens in the third stage of this procedure to represent a set of structures is analogous to the procedure that has been described by Murrall and Davies[19] to represent a set of conformers in the context of flexible 3-D searching. Superimposition was used in the present context to eliminate the effect of "ghost substructures" on the screen search, where ghost substructures are substructures that are present in a hyperstructure but not in any individual input structure.[5] The use of the individual structure screens, as opposed to screens derived from the hyperstructures themselves, avoids the problem of eliminating screen matches arising from ghosts.[16] The use of the individual structure screens also means that the same screen record, and hence the same screenout, will be obtained whichever method (AA or GA) is used to generate the hyperstructure that is used in the final Ullmann search.

The screening of whole clusters using this method is necessarily less precise than for individual structures, since clusters which pass screening may contain some structures which do not contain all the necessary screens, and may even contain no structures which individually contain all of the necessary screens. Figure 2 shows how the screenout obtained for one particular query changes as the average cluster size increases as a result of varying the $k_{\min}$ parameter. As noted in section 2.3, this specifies the number of nearest neighbors which two structures must have in common if they are to cluster together: thus, a large value for $k_{\min}$ gives smaller

Table 3. Screenouts and Speedups for Screened Substructure Searches of Cluster Hyperstructures[a]

hyperstructure series (k_{min})	cluster screenout	structure screenout	speedup
non-H1	96.8 (77.0–98.5)	96.8 (77.0–98.5)	2.9 (1.5–8.0)
non-H2	96.8 (77.0–98.5)	96.8 (77.0–98.5)	2.2 (1.3–7.4)
19	96.1 (75.3–98.2)	96.4 (74.7–98.1)	1.6 (1.3–5.3)
15	94.9 (76.9–97.8)	93.4 (64.2–95.6)	1.5 (1.1–3.4)
9	95.0 (78.6–97.6)	80.8 (51.6–94.9)	1.4 (1.1–2.8)
5	95.0 (79.1–97.5)	77.2 (49.9–94.9)	1.4 (1.1–2.5)
1	95.0 (79.5–97.5)	72.3 (48.1–94.6)	1.3 (1.0–2.3)

[a] Non-H1 and non-H2 denote serial searches that identify either the first occurrence or all occurrences of a query substructure (as discussed by Brown *et al.*[1]). The average cluster size is controlled by the value of k_{min}: 19 gives the smallest clusters and 1 the largest clusters (see text in section 3). In each case, we quote the median value with the interquartile range in brackets.

clusters and *vice versa*. The sequential file is shown at the left hand side; this is treated as a file with one structure per cluster. Screenout is measured in two ways, firstly as a percentage of clusters eliminated, *cluster screenout*, and also as a percentage of the original input structures eliminated, *structure screenout*. This graph shows that although the cluster screenout remains approximately constant, at around 97% for this query, there is a decrease in the actual number of structures being screened-out as the average size of the clusters increases. This is due to the increased likelihood that a given cluster will pass screening while individual structures contained in that cluster do not contain all the screens present in the query.

In our previous paper,[5] the EINECS data were clustered under a range of conditions to produce five different series of clusters with increasing average cluster size. These were converted to hyperstructures using the AA method and searched against our test set of 27 substructural queries. These experiments were repeated with the addition of the screening system, giving another set of substructure search times which could be compared to those obtained previously. From these a speedup has been calculated for each query searched against each set of clusters. A summary of these results is shown in Table 3, where the speedup is calculated as the CPU time for nonscreened search divided by the CPU time for the screened search. The table shows, for each set of clusters and the sequential search, the median speedup over the 27 queries together with the interquartile range and maximum value. The median speedup disguises the fact that several of the searches were very much better. There were six queries in which the speedup exceeded 1 order of magnitude, including one in which the difference was 2 orders of magnitude.

As with a conventional substructure-search system, the screenout that is obtained is strongly dependent on the specificity of a query substructure, with queries exhibiting complex or unusual features resulting in high screenout. Several individual queries produced very low screenout, and consequently little speedup, because only a very few bits were set: this may be due, in part at least, to the use of a screen dictionary that was not specifically developed for this set of structures (since even the conventional sequential searches, the non-H1 and non-H2 searches,[5] have relatively low speedups and screenouts). Hardly surprisingly, the screenout decreases as larger clusters are searched, since the bit strings then represent a less precise characterization of the structures contained with an individual cluster hyperstructure.

Our experiments have involved screening cluster-hyperstructure searches. A screening system for a global hyperstructure might seek to eliminate parts of the hyperstructure which could not contain the query, in effect masking out particular hyperstructure nodes. One method would be to store with each hyperstructure node a list of the screens resulting from the input structure atoms mapped to that node. A query would then be analyzed on the basis of the same screen dictionary, and only those hyperstructure nodes that contained all of the query atom screens would then be possible matches. We have not evaluated this approach to date.

Generally, the introduction of screening has increased the efficiency of substructure searching in clustered hyperstructures, particularly when the average cluster size is low. The best speedups were of 1 or 2 orders of magnitude. The proposed method for screening individual parts of hyperstructures would be needed for a global hyperstructure, and this might also be appropriate for searching large cluster hyperstructures (in which case the two screening methods would be complementary).

4. SUBSTRUCTURE SEARCH USING MASSIVELY-PARALLEL HARDWARE

Wipke and Rogers[20] have described a method for MCS detection which involved collecting common segments of chemical graphs in a tree in which the nodes represented the segments, and the branches the connections between them. This, they suggested, was in some senses a software paralization of the problem of MCS detection. Similarly, the use of a hyperstructure may be considered to be one software solution to the parallelization of substructure searching, since it allows structures to be represented, and therefore searched, in parallel. In this section we consider the use of a hardware parallel algorithm to attempt to speed the substructure-search process.

Willett *et al.*[21] have considered an implementation of the Ullmann algorithm on a massively-parallel processor for 2-D substructure searching and showed that the use of this hardware led to an increase in substructure search speed for 2-D chemical structures. They considered two approaches: the first, *outer-loop* or *data-parallel*, approach distributed one structure to each processor and searched these concurrently using the serial algorithm, while the second, *inner-loop* or *algorithm-parallel*, approach exploited the inherent parallelism within the algorithm itself, and searched one structure at a time, distributing it over the processor array. The inner-loop algorithm was found to be the less efficient of these two approaches, and it was suggested that this was because the 2-D chemical graphs that were studied had too low a degree of connectivity to utilize fully the available processors. Hyperstructures are much larger and more highly connected than conventional molecules, and it was thus thought that the inner-loop parallization of the Ullmann algorithm might be more appropriate in the present context than when we had used it for searching individual structures. The inner-loop algorithm is designed to search one chemical graph at a time, and is thus most obviously applied to increasing the efficiency of searching a global hyperstructure (or a single, large cluster hyperstructure).

The parallel Ullmann algorithm for conventional molecules was described by Willett *et al.*[21] A number of modifications had to be made to allow for the parallel search of hyperstructures, as detailed by Brown.[16] The modified version was implemented on two massively-parallel machines: the Active Memory Technology Distributed Array Processor[22] and the MasPar MP-1,[23] specifically a DAP 610 and an MP-1104. Both are single-instruction stream, multiple-data stream (SIMD) machines with 4096 processing elements in a square array; further details of the use of these two machines for database searching are given by Wild and Willett.[24] The algorithm was implemented on both machines in their own versions of Fortran with parallel extensions.

The algorithm was initially tested on the DAP 610. Each processing element, or PE, on this machine has 8 Kbytes of local memory, and this proved insufficient to load the global hyperstructure of the EINECS data set into memory. Instead, a subset of 761 structures was used for testing. When the global hyperstructure of this subset was searched, it was found that the search using parallel hardware gave a similar performance in terms of search time to the equivalent search on the serial ESV-30 workstation: each method was the faster in approximately half of the cases examined. A peak speedup of around five times was recorded for two of the most general queries.

The MP-1104 has 16 Kbytes in each PE, and this meant that it was possible to test the algorithm with the full data set. However, the results obtained from searching the global hyperstructure were most disappointing. No speedup was obtained for any of the queries, when compared with an Evans and Sutherland ESV-30 workstation, and in many cases the search times were very much slower than for the serial implementation.[16] The reason for this is thought to be due to the nature of the parallel version of the Ullmann algorithm. Specifically, the inner-loop algorithm makes extensive use of Boolean logical operations,[21] and these are accomplished most efficiently if the adjacency matrices that are used to represent a query and a database structure are of the same size, and if the match matrix, which records the possible matches of query atoms with database-structure atoms, is also of the same size. These requirements are met by adding dummy atoms to the query so that it is the same size as the current database structure (thus causing a computational overhead since all of the matrix elements have to be processed, even if they contain only dummy values). In the case of conventional 2-D structures, the queries and database structures were of comparable size so that there was little additional processing. However, the global hyperstructure of the EINECS data set had 222 nodes; this greatly exceeded the query sizes, with the result that the great bulk (typically over 99.5%) of the expanded query and match matrices were zero-values with a comparably large processing overhead. In addition, all of these matrices contained over 12 times as many elements as the number of available PEs and this incurs additional significant overheads when a large amount of matrix processing is required, as was the case here. These two factors caused the severe deterioration that was observed in the performance of the algorithm, when compared with the serial implementation.

5. CONCLUSIONS

In this paper, we have studied three approaches to increasing the speed of substructure search in hyperstructures.

The first involved the use of a GA to produce a more-compact, better-mapped hyperstructure than is possible using the AA method investigated previously. This permits a statistically-significant reduction in substructure-searching times when compared with the AA method; however, the generation time per structure is much greater than for the AA method (although this is a one-off cost when the database is built). The second has been to introduce a screening system for cluster hyperstructures. This again has led to an increase in search efficiency for many queries; however, a screening system for global and large cluster hyperstructures would have to be developed before a full comparison could be made between screened and nonscreened searches. The final approach, the use of parallel hardware, proved to be most disappointing; reasons for this behavior are detailed in section 4.

The hyperstructure concept was originally proposed by Vladutz and Gould,[1] although they did not implement their ideas. The work reported in this and our previous paper[5] shows that there is at least some basis to their claims that hyperstructures could increase the efficiency of substructure searching, though we have also presented evidence to suggest that the degree of compression achieved during the hyperstructure-generation phase is not a crucial determinant of search efficiency. Vladutz and Gould suggested that they would expect to obtain an order-of-magnitude speedup in substructure search time over sequential search, based on calculations using a theoretical hyperstructure containing 10 million compounds. While we have not implemented the method on anywhere near this number of compounds, we have demonstrated two methods by which it would be possible to generate hyperstructures from nontrivial data sets and have obtained speedups of the order suggested by Vladutz and Gould for several well-defined, specific queries. That said, it should be noted that the advent of the S4 system[4] means that 2-D substructure searching can now be carried out very much faster than it could when Vladutz and Gould first put forward their ideas: it seems most unlikely that the hyperstructure approach (at least in the form studied by Vladutz and Gould and by ourselves) could provide a comparable level of efficiency.

ACKNOWLEDGMENT

We thank Molecular Design Ltd., ORAC Ltd., the Science and Engineering Research Council, and Wellcome Research Laboratories for the award of CASE Studentships to R.D.B. and G.J.; the European Community Joint Research Centre for the provision of the EINECS data set; Barnard Chemical Information Ltd. for the clustering software and the fragment dictionary used for screening; Tripos Associates for the provision of the Evans and Sutherland workstation; Queen Mary and Westfield College, University of London, and MasPar Computer Ltd. (Reading, U.K.) for the provision of parallel computing facilities; and Tony Cook, Nigel Jagger, and John Steel for many helpful discussions. The Krebs Institute for Biomolecular Research is a designated Centre for Molecular Recognition Studies under the Molecular Recognition Initiative of the Science and Engineering Research Council.

REFERENCES AND NOTES

(1) 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.

(2) Nagy, M. A.; Kozics, S.; Veszpremi, T.; Bruck, P. Substructure Search on Very Large Files Using Tree-Structured Databases. In *Chemical Structures. The International Language of Chemistry*; Warr, W. A., Ed.; Springer Verlag: Berlin, 1988; pp 127–130.

(3) Okada, T.; Wipke, W. T. CLUSMOL: a System for the Conceptual Clustering of Molecules. *Tetrahedron Comput. Methodol.* **1989**, *2*, 249–264.

(4) Hicks, M. G.; Jochum, C.; Maier, H. Substructure Searching Systems for Large Chemical Databases. *Anal. Chim. Acta* **1990**, *235*, 87–92. Bartmann, A.; Maier, A.; Walkowiak, D.; Roth, B.; Hicks, M. G. Substructure Searching on Very Large Files by Using Multiple Storage Techniques. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 539–541.

(5) Brown, R. D.; Downs, G. M.; Willett, P.; Cook, A. P. F. A Hyperstructure Model for Chemical Structure Handling: Generation and Atom-by-Atom Searching of Hyperstructures. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 522–531.

(6) Jarvis, R. A.; Patrick, E. A. Clustering Using a Similarity Measure Based on Shared Nearest Neighbours. *IEEE Trans. Comput.* **1973**, *C-22*, 1025–1034.

(7) Willett, P.; Winterman, V.; Bawden, D. Implementation of Non-Hierarchic Cluster Analysis Methods in Chemical Information Systems: Selection of Compounds for Biological Testing and Clustering of Substructure Search Output *J. Chem. Inf. Comput. Sci.* **1986**, *26*, 109–118.

(8) Ullmann, J. R. An Algorithm for Subgraph Isomorphism. *J. Assoc. Comput. Mach.* **1976**, *23*, 31–42.

(9) 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 195–215.

(10) Davis, L., Ed. *Handbook of Genetic Algorithms*; Van Nostrand Reinhold: New York, 1991.

(11) Goldberg, D. E. *Genetic Algorithms in Search, Optimisation and Machine Learning*; Addison-Wesley: Wokingham, U.K., 1989.

(12) Fountain, E. Application of Genetic Algorithms in the Field of Constitutional Similarity. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 748–752.

(13) Brown, R. D.; Jones, G.; Willett, P.; Glen, R. C. Matching Two-Dimensional Chemical Graphs Using Genetic Algorithms. *J. Chem. Inf. Comput. Sci.*, paper appearing elsewhere in this issue.

(14) Chase, W.; Bown, F. *General Statistics*; Wiley: New York, 1986.

(15) Siegel, S. *Nonparametric Statistics for the Social Sciences*; McGraw-Hill: Tokyo, 1956.

(16) Brown, R. D. A Hyperstructure Model for Chemical Structure Handling. Ph.D. thesis, University of Sheffield, 1993.

(17) Warr, W. A. Systems for Chemical Structure Handling. In *Chemical Structure Systems*; Ash, J. E., Warr, W. A., Willett, P., Eds.; Ellis Horwood: Chichester, U.K., 1991; pp 88–125.

(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) Murrall, N. W.; Davies, E. K. Conformational Freedom in 3-D Databases. 1. Techniques. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 312–316.

(20) Wipke, W. T.; Rogers, D. Tree-Structured Maximal Common Subgraph Searching. An Example of Parallel Computation with a Single Sequential Processor. *Tetrahedron Comput. Methodol.* **1989**, *2*, 177–202.

(21) Willett, P.; Wilson, T.; Reddaway, S. F. Atom-By-Atom Searching Using Massive Parallelism. Implementation of the Ullmann Subgraph Isomorphism Algorithm on the Distributed Array Processor. *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 225–233.

(22) Parkinson, D., Litt, J., Eds. *Massively Parallel Computing with the DAP*; Pitman: London, 1990.

(23) Blank, T. The MasPar MP-1 Architecture. Paper presented at Compcon Spring 90 - The 35th IEEE Computer Society International Conference, San Francisco, Feb 26–Mar 2, 1990.

(24) Wild, D. J.; Willett, P. Similarity Searching in Files of Three-Dimensional Chemical Structures: Implementation of Atom Mapping on the Distributed Array Processor DAP-610, the MasPar MP-1104, and the Connection Machine CM-200. *J. Chem. Inf. Comput. Sci.*, paper appearing elsewhere in this issue.