Matching Two-Dimensional Chemical Graphs Using Genetic Algorithms[†]

Robert D. Brown, Gareth Jones, and Peter Willett*

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

Robert C. Glen

Wellcome Research Laboratories, Langley Park, Beckenham, Kent BR3 3BS, U.K.

Received May 19, 1993[®]

This paper describes the application of a genetic algorithm (GA) in the field of 2-D chemical structure handling. A GA is an algorithm based on the process of Darwinian Evolution, whereby potential solutions to a problem are manipulated according to the mechanics of natural selection. A graph-matching GA has been developed for substructure searching of 2-D chemical databases. A variant of the GA has been used to generate hyperstructures, where a hyperstructure is a novel method of representation for 2-D chemical structures. Both GAs were extensively tested. The GA for substructure search was shown to be considerably less effective than conventional search methods, while the hyperstructure-generation algorithm was found to be highly successful.

1. INTRODUCTION

Genetic algorithms, or GAs, are based on the process of Darwinian Evolution where successive generations of populations of artificial strings are manipulated according to the mechanics of evolution. GAs were originally developed by Holland[1] and have become widely used over the last few years for a wide range of applications that involve combinatorial optimization.[2,3] The basic design of the GA used in this work is that of a *steady-state-with-no-duplicates GA* as described in ref 3. The GA works by manipulating *chromosomes*, which are encoded representations of the problem. Following the creation of an initial population of random chromosomes a *fitness function* is used to assign a *fitness value* to each chromosome, where the fitness value reflects how close a particular chromosome is to the solution of the problem under investigation. Successive operators are then applied to the population until an optimal (or near-optimal) solution is obtained. These operators are typically either *crossover* or *mutation*. During mutation a single parent is selected from the population, and this parent's chromosome is perturbed to produce a child chromosome. The crossover operator requires two parents, whose chromosomes are combined to produce two child chromosomes. Following the application of an operator the children produced by that operator replace the least fit members of the population.

The childrens' parents are selected using a mechanism known as *roulette-wheel selection*. This process is best understood by visualising a roulette wheel on which every potential parent has a slice linearly proportional to its fitness value. The wheel is then spun to select the parent. Clearly, the most fit members of the population are most likely to become parents. The roulette-wheel mechanism is also used in determining which operator to apply to the population. A set of operators is defined and each operator given a weight. Roulette-wheel selection is then performed using these weights to determine which operator to apply, so that the operators with the highest weights have the greatest likelihood of being chosen. As increasing numbers of operations are applied to the population, the average fitness of the population rises until, hopefully, an optimal solution is found. The GA can be viewed as a multidimensional search algorithm, with the mutation operators allowing the exploration of the full search space and the crossover operators allowing the refinement of intermediate solutions. The GA is summarized in Figure 1 (the concept of *windowed fitness* in step 4 of this algorithm is described in section 2.4).

In this paper, we discuss the use of GAs to process the labeled graphs that are used to represent 2-D chemical structure diagrams in chemical information systems.[4] Specifically, we report the development of a GA for the detection of subgraph isomorphism and maximal common subgraph isomorphism, both of which are known to be NP-complete computational problems. In section 2 we detail the development of a GA for substructure searching and compare the algorithm's performance with the Ullmann algorithm, which is the deterministic algorithm used in many commercial substructure search systems. In section 3 the GA is adapted for a similarity problem: the computation of the *maximum overlap set* (or disconnected maximum common subgraph) between a structure and hyperstructure.[5] The paper concludes in section 4 with a summary of our major findings.

2. A GA FOR SUBSTRUCTURE SEARCHING

2.1. The Problem. The problem is to verify whether or not a given query substructure is contained within a database structure. By generation mappings of nodes between a query structure and database structure, the GA will attempt to maximize the number of matching edges (bonds) between the query and structure, over the mapping. If all bonds in the query are matched in the structure, then the query is a substructure of the structure. The discussion that follows will be exemplified by the database structure and query substructure shown in Figures 2 and 3, respectively. Inspection of these two figures shows that the query structure is a substructure of the database structure, with two possible overlaps.

2.2. Encoding the Problem. In this problem an integer-string chromosome is used. The chromosome represents a

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

1. A set of reproduction operators (crossover, mutation etc) is chosen. Each operator is assigned a weight.
2. An initial population is created randomly and the fitnesses of its members determined.
3. An operator is chosen using roulette wheel selection based on operator weights.
4. The parents required by the operator are chosen using roulette wheel selection based on windowed fitness.
5. The operator is applied and child chromosomes produced. Their fitness is evaluated.
6. The children replace the least fit members of the population.
7. If an acceptable solution has not been found and the threshold time has not been reached then go to Step 3.

Figure 1. Operator-based GA.

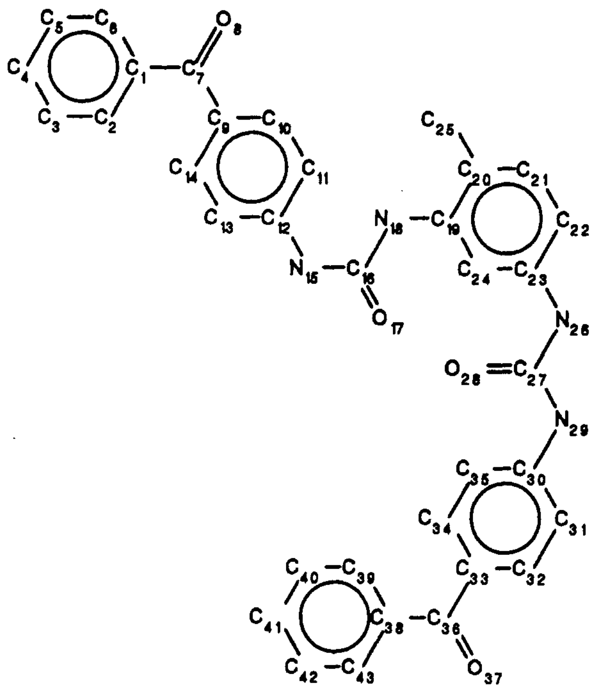

Figure 2. Example database structure.

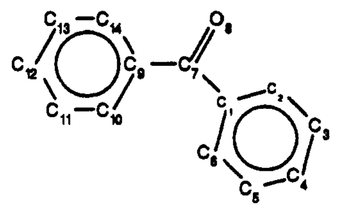

Figure 3. Example query structure.

mapping between query and structure nodes, with each integer in the string being the label of a structure node and the position of that integer being the label of a query node. Thus a query structure containing N non-hydrogen atoms will yield a chromosome with N integers.

As an example chromosome, consider the following (poor) mapping between the example query structure and database structure.

$$\{\{1 \to 9\}, \{2 \to 10\}, \{3 \to 40\}, \{4 \to 38\}, \{5 \to 42\}, \{6 \to 31\}, \{7 \to 7\}, \{8 \to 8\}, \{9 \to 1\}, \{10 \to 21\}, \{11 \to 23\}, \{12 \to 30\}, \{13 \to 6\}, \{14 \to 4\}\}$$

This mapping encodes as the following chromosome:

9 10 40 38 42 31 7 8 1 21 23 30 6 4

Chromosome values are restricted so that the mapping represented by the chromosome is meaningful, in that no duplicates are permitted in the chromosome, query atoms must map onto structure atoms of the same type (i.e., carbons map only to carbons, etc.), and the set of bonds surrounding a structure node must be contained within the set of bonds surrounding the query node which maps onto it.

2.3. The Fitness Function. The number of bonds (edges) that match over the mapping encoded in a chromosome are used to determine its fitness. Let A and B be the labels of two atoms in the query structure and let A and B be connected by a bond of type n. If a decoded chromosome gives rise to the mappings $A \to X$ and $B \to Y$, then an edge match occurs if and only if database structure atoms X and Y are connected by a bond of type n. A possible measure of fitness might then be the number of matching edges. However, the set of matching edges may be comprised of two or more disjoint query structure fragments. As there is no guarantee that these separate fragments can combine into the query structure, a better measure is the number of bonds (edges) in the largest fragment.

In graph-theoretical terms, a *path* in a graph is a sequence of edges joing two nodes. A *connected graph* has paths between any pair of nodes, and a *maximal connected graph* is a connected graph which is not a subset of any other connected graph. It follows that any graph can be partitiond into a set of maximal connected graphs. The set of matching edges for a mapping, obtained by decoding a chromosome, is partitioned into maximal connected graphs, and the fitness of a chromosome is then the number of edges in the largest maximal connected graph.

As an example consider the chromosome

9 10 40 38 42 31 7 8 1 21 23 30 6 4

Query edges {{7,8}, {7,1}, {7,9}, {1,2}} map onto structure edges. This gives rise to a fitness of 4 since all of these edges comprise a connected graph.

As a second example consider the chromosome

32 33 39 21 23 6 36 17 9 10 11 12 13 14

The set of query edges that map onto structure edges is

{{9 ,10}, {10, 11}, {11, 12}, {12, 13}, {13, 14}, {14, 9}, {1, 2}}

which gives rise to two connected graphs:

{{9, 10}, {10, 11}, {11, 12}, {12, 13}, {13, 14}, {14, 9}} and {1, 2}

Thus the fitness of this chromosome is 6, equal to the number of edges in the larger connected graph.

It should be noted that only part of a chromosome contributes to the fitness of that chromosome. To illustrate this, consider the example chromosome

9 10 40 38 42 31 7 8 1 21 23 30 6 4

Let each value in the chromosome that does not contribute to the fitness be replaced by a "*" or "do not care" symbol, then the following *node-set representation* is derived

9 10 * * * * 7 8 1 * * * * *

This representation defines the useful information within a chromosome. As another example, the chromosome

32 33 39 21 23 6 36 17 9 10 11 12 13 14

has the following node-set representation:

* * * * * * * * 9 10 11 12 13 14

2.4. Parent Selection. This section is concerned with step 4 in Figure 1. As the GA runs, the minimum fitness in the population will tend to rise. This has the result that the fitter members of the population will not have such a large advantage when roulette wheel selection is performed. In order to focus attention on these individuals, the GA uses a fitness scaling technique known as *windowed fitness scaling* (this and other fitness scaling techniques are described by Davis[3]). Let F_{raw} be the fitness described in section 2.3 and F_{min} be the minimum fitness present in the population: then the windowed fitness is $F_{raw} - F_{min} + 1$. During parent selection, roulette-wheel selection was performed on these scaled fitness values.

2.5. Mutation. This section describes the mutation operator. A single parent is selected. A position on the chromosome string is selected at random and the integer value at that position mutated to another value. During mutation the integer value is replaced by another structure node to which that query node is allowed to map.

Consider the example chromosome:

$$9\ 10\ 40\ 38\ 42\ \underline{31}\ 7\ 8\ 1\ 21\ 23\ 30\ 6\ 4$$

Suppose that position 6 is chosen for mutation. The current value at this position is 31, which is underlined. Query node 6 is a carbon atom ring-bonded to two neighbors. The set of structure nodes that query node 6 can map to is

$$\{1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 14, 19, 20, 21, 22, 23, 24, 30, 31, 32, 33, 34, 35, 38, 39, 40, 41, 42, 43\}$$

Let 35 be randomly chosen from this list as the new value for position 6, so that the new mutated chromosome is then

$$9\ 10\ 40\ 38\ 42\ \underline{35}\ 7\ 8\ 1\ 21\ 23\ 30\ 6\ 4$$

However, if 21 were selected from the list, then the chromosome would contain duplicate values, since 21 occurs at position 10. In this case the two values are simply exchanged, so that position 6 is set to 21 and position 10 is set to 31, giving the chromosome

$$9\ 10\ 40\ 38\ 42\ \underline{21}\ 7\ 8\ 1\ \underline{31}\ 23\ 30\ 6\ 4$$

There is one final problem with this mutation algorithm. Suppose that 9 is chosen from the list as the mapping for query node 6. The value 9 already occurs at position 1, so the following chromosome is generated (after an exchange analogous to that described above):

$$\underline{31}\ 10\ 40\ 38\ 42\ \underline{9}\ 7\ 8\ 1\ 21\ 23\ 30\ 6\ 4$$

However, query node 1 is not permitted to map to structure node 31, since query node 1 has three neighbors while structure node 31 has only two. In this case another mutation is attempted on the same chromosome.

The mutation operator then returns the child chromosome for insertion into the population.

2.6. Crossover. The current literature provides several crossover mechanisms for chromosome strings of permutations of integers[2,3,6] that can be applied to this problem. Additionally a new crossover mechanism, called *node-based crossover*, has been developed for this problem.

The crossover mechanisms described here use a reordering mechanism to remove duplicates. This mechanism can give rise to unsuitable chromosomes in the same way that illegal chromosomes can be generated by mutation. However, rather than repeating the crossover operator, the child chromosome is discarded, and not added into the population.

2.6.1. Uniform Crossover. *Uniform crossover* was developed by Syswerda[7] in order to overcome theoretical limitations of the traditional point crossovers commonly used in GAs.

Uniform crossover is illustrated in Figure 4, in which two parents are selected and two children are produced. For each integer position in the first child chromosome, a random decision is made to determine which parent will contribute its value at that position to the child. The second child receives its integer value at that position from the other parent. In Figure 4 the template *M* is produced by *N* random {0, 1} decisions, where *N* is the length of the chromosome. *A′* takes its value from *A* when the template is 1 or from *B* when the template is 0. Likewise *B′* takes its value from *B* when *M* is 1 or from *A* when *M* is 0. A consequence of uniform crossover is that duplicate integer values may occur in the child chromosomes (such as 23 in *B′*). Davis[3] has developed a uniform order-based crossover that eliminates this problem. Unfortunately the re-ordering method cannot be used in the present context since this method is best suited for applications where the relative ordering (as opposed to the absolute location) of integer positions is important. Furthermore, this algorithm would result in structure atoms being mapped to query atoms of different atomic types. This cannot occur with the re-ordering mechanism used by Goldberg in *PMX crossover*.[2] The PMX re-ordering mechanism was thus incorporated into uniform crossover and *A″* and *B″* are the children produced.

A	9	10	40	38	42	31	7	8	1	21	23	30	6	4
B	33	32	39	21	23	6	36	17	9	10	11	12	13	14
M	0	1	1	1	1	0	1	0	1	1	0	0	0	1
A′	33	10	40	38	42	6	7	17	1	21	11	12	13	4
B′	9	32	39	21	**23**	31	36	8	9	10	**23**	30	6	14
A″	33	10	40	38	42	6	7	17	1	21	11	12	13	4
B″	9	32	39	21	23	31	36	8	9	10	**42**	30	6	14

A Parent 1.
B Parent 2.
M Uniform crossover template.
A′ Child 1 from conventional uniform crossover.
B′ Child 2 from conventional uniform crossover.
A″ Child 1 from uniform crossover with PMX style re-ordering.
B″ Child 2 from uniform crossover with PMX style re-ordering.

Figure 4. Uniform crossover.

Suppose that a duplicate occurs in *B′*. It is removed as follows. Let the duplicate have value *D*, and let *i* be the position of *D* in *B* and *j* be the position of *D* in *A*. Using PMX re-ordering, the value of *B″* at *i* is set to the value of *A* at *j*. If this still produces a duplicate, the process is then repeated iteratively until no duplicates occur (at which point the procedure will terminate successfully).

Both child chromosomes in Figure 4 have fitness 2. The mapping encoded in *A″* results in the following query structure edges being mapped onto database structure edges:

$$\{\{11, 12\}, \{12, 13\}, \{7, 1\}\}$$

while the mapping in *B″* maps

$$\{\{9, 10\}, \{9, 14\}\}$$

2.6.2. Node-Based Crossover. Node-based crossover is a new form of crossover that has been developed specifically for this application. Its mechanics are similar to those of uniform crossover in that a crossover template is generated and crossover then proceeds as for uniform crossover and duplicates are removed from the child chromosomes using the techniques of PMX crossover. However, in node-based crossover, the crossover template is no longer randomly generated: instead a template is used that attempts to generate child chromosomes whose node-set representation is the sum of the node-set representations of its parents. In addition to the effectiveness of node-based crossover, our experiments showed it to be about twice as fast as uniform crossover.

Node-based crossover is illustrated in Figure 5. *A* and *B* are the example chromosomes, and *A′* is the child that results from crossing *A* with *B*. *A′* is produced by taking a value

A	9	10	40	38	42	31	7	8	1	21	23	30	6	4
B	33	32	39	21	23	6	36	17	9	10	11	12	13	14
N_A	9	10	*	*	*	*	7	8	1	*	*	*	*	*
N_B	*	*	*	*	*	*	*	*	9	10	11	12	13	14
M_A	1	1	*	*	*	*	1	1	1	*	*	*	*	*
M_B	*	*	*	*	*	*	*	*	1	1	1	1	1	1
M_{AB}	1	1	0	0	0	1	1	1	1	0	0	0	0	0
A'	9	**10**	39	21	23	31	7	8	1	**10**	11	12	13	14
A''	9	10	39	21	23	31	7	8	1	**32**	11	12	13	14
M_{BA}	0	0	0	1	0	1	0	0	1	1	1	1	1	1
B'	**9**	**10**	40	21	42	6	7	8	**9**	**10**	11	12	13	14
B''	**1**	**38**	40	21	42	6	7	8	9	10	11	12	13	14

A	Parent 1.
B	Parent 2.
N_A	Node-set representation of A.
N_B	Node-set representation of B.
M_A	Contribution to template from node-set representation of A.
M_B	Contribution to template from node-set representation of B.
M_{AB}	Crossover template for crossing A with B.
A'	Child 1 from crossing A with B.
A''	Child 1 with PMX re-ordering to remove duplicates.
M_{BA}	Crossover template for crossing B with A.
B'	Child 2 from crossing B with A.
B''	Child 2 with PMX re-ordering to remove duplicates.

Figure 5. Node-based crossover.

from A when M_{AB} is 1 and a value from B when M_{AB} is 0. A'' is the child chromosome following PMX-style re-ordering to remove duplicates. M_{AB} is generated as follows: bits are set to 1 if there is not a "*" in the node-set representation for A at that point (M_A); other bits are set to 0 if there is not a "*" for B at that point (M_B) and any remaining bits are set randomly. A similar process is used to generate B'' by crossing B with A.

On decoding A'' and B'', a large number of query edges are found that map to structure edges.

The following mapped edges are found for A'':

$$\{\{1, 2\}, \{7, 8\}, \{7, 1\}, \{11, 12\}, \{12, 13\}, \{13, 14\}\}$$

This edge set is the combination of edges from A and B. Unfortunately this edge set is not a connected graph, but comprises two connected graphs

$$\{\{1, 2\}, \{7, 8\}, \{7, 1\}, \{9, 7\}\} \text{ and } \{\{11, 12\}, \{12, 13\}, \{13, 14\}\}$$

The first graph is the larger, giving A'' a fitness of 4. The first graph comes from the node set of A while the second graph comes from the node set of B. This crossover illustrates a problem with node-based crossover, in that although information from both parents is combined in the child, there will be no improvement in the fitness (given the fitness function we have chosen to use), unless that information overlaps.

The following mapped edges are found for B'':

$$\{\{1, 7\}, \{7, 8\}, \{7, 9\}, \{9, 10\}, \{10, 11\}, \{11, 12\}, \{12, 13\}, \{13, 14\}, \{14, 9\}\}$$

In this case the information from the parents has combined in a connected graph, giving B'' a fitness of 9. The reason why B'' has successfully combined the information from its parents is that position 9 is in the node set of both parents. B'' gets its value for position 9 from B, while A'' gets this value from A.

The node set representations for A and B can be found in section 2.3. The node-set representation for A'' is the same as that for A. B'' has a node-set representation that is a combination of the node-sets of A and B. Additionally, the new value of 1 at position 1 has occurred due to PMX re-ordering.

1 * * * * * 7 8 9 10 11 12 13 14

2.7. Population Restrictions. In order to prevent convergence to a suboptimal solution, the population is managed to ensure that it contains diverse and meaningful members. Firstly, no duplicate chromosomes are permitted within the population. As duplicate chromosomes contain the same information, the presence of duplicate chromosomes weights parent selection to that chromosome and can result in a speedy convergence to a suboptimal solution. Secondly, no zero-fitness chromosomes are allowed. Zero-fitness chromosomes contain no useful information whatsoever: they can be thought of as "blanks" in the population.

Unfortunately these enhancements are not sufficient to prevent convergence to a suboptimal solution. During development the algorithm rarely found the maximal mapping, and the concept of *niche restriction* was thus incorporated into the algorithm. Goldberg has suggested the use of *niches*[2,8] to prevent the convergence of an algorithm to suboptimal solutions and also to explore suboptimal solutions. A set of niches is defined over the population space, and a restricted number of chromosomes is permitted in each niche.

Consider the chromosome used in the node-based crossover example. Both A and A'' carry the same solution information and thus have the same node-set representation. However, they are clearly not duplicates. In fact there are thousands of chromosomes with this node-set representation. For this reason each node-set representation defines a niche. A niche size of 1 was used, and no two chromosomes with the same node-set representations are allowed in the population. Thus, following the example node-based crossover, A'' would not be inserted into the population, since A is already present in the population and inhabits the same niche.

2.8. Heuristics. The aim of this algorithm is to generate a substructure match between the query structure and the database structure. Near-optimal solutions are thus of no interest.

A simple heuristic was incorporated into the fitness evaluation module. When the node-set representation of a chromosome contains no "*"s, the chromosome represents a complete solution. If this is not the optimum solution (that is, does not represent a full mapping of substructure to structure), then this chromosome is likely to represent a suboptimal solution. Such suboptimal solutions are of no interest in verifying that the query is a substructure of the database structure. Therefore, the chromosome is discarded.

Additionally, if the node-set representation contains only one "*" (that is, all query nodes bar one are part of a solution), then an attempt is made to fit in an alterntive structure node that will generate an optimal solution. If this cannot be done, the fitness of the chromosome is set to zero.

2.9. Termination Conditions. The algorithm continues until a match is found between the query structure and the database structure, or until the number of operations performed (i.e., the number of invocations of steps 3–6 in the generalized operator-based GA shown in Figure 1) exceeds 20 000.

2.10. An Example. Using the database structure and query substructure shown in Figures 2 and 3, respectively, a C implementation of the GA described above found the following query-to-structure mapping in 0.48 s on a SUN 4/470 minicomputer (which represents the application of a total of 128 genetic operators in the GA of Figure 1).

$$\{\{1 \rightarrow 33\}, \{2 \rightarrow 32\}, \{3 \rightarrow 31\}, \{4 \rightarrow 30\}, \{5 \rightarrow 35\}, \{6 \rightarrow 34\}, \{7 \rightarrow 36\}, \{8 \rightarrow 37\}, \{9 \rightarrow 38\}, \{10 \rightarrow 39\}, \{11 \rightarrow 40\}, \{12 \rightarrow 41\}, \{13 \rightarrow 42\}, \{14 \rightarrow 43\}\}$$

The following GA parameters were used: a population size of 35 and operator weights of 10 for mutation and node-based crossover, i.e., both operators were considered of equal

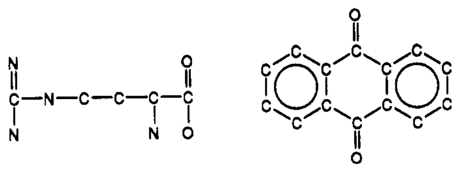

Figure 6. Query structures.

importance. The same parameters were used for the more extended experiments discussed below.

2.11. Comparison with Ullmann Algorithm. In order to test the efficiency of the algorithm a comparison run was done with the Ullmann algorithm[9] using a dataset of 3 queries and 140 structures, which were known to contain them. One query is the example substructure in Figure 3; the other two query substructures are shown in Figure 6.

As the GA has no batch run facility, the total run times for single query/structure matches was used as a comparison. Since the GA is nondeterministic, its average run time over 10 runs was computed. This time was then compared with a single run of a Fortran 77 implementation of the Ullmann algorithm. Repeating this process over all query/structure pairs, the Ullmann algorithm was found to outperform the GA by a factor of 3.5 times.

The Ullmann algorithm program is an extremely efficient piece of Fortran. The GA, however, has been written with the intention of obtaining a working algorithm and little thought has been given to efficient coding. It might be thought that to get an initial level of performance within an order of magnitude of the Ullmann program would imply that the GA has real potential for this problem. Unfortunately, such a conclusion overlooks a very real limitation of the GA in those frequent cases when the query is not a substructure of the database structure. In this case the Ullmann algorithm can swiftly verify that the query structure is not contained in the database structure; indeed, much of the efficiency of this algorithm in an operational context derives from its ability to identify such mismatches extremely rapidly. However, since the GA is nondeterministic, it can only carry on, generating more chromosomes. The only way around this problem is to set thresholds and terminate the algorithm if these thresholds are exceeded, a procedure that has undesirable consequences. In many cases the run time of the GA is likely to be longer when the query structure is not a substructure of the database structure than when it is, whereas the reverse is the case with the Ullmann algorithm. Conversely, unless these thresholds are set to exceedingly high values, there is always the possibility that an isomorphism may be missed. It would be possible to carry out a simulation study to determine the probability of missing such true isomorphisms at a given threshold. However, the first consequence above is such that it forces us to conclude that the GA is inappropriate for subgraph isomorphism applications involving graphs of the (relatively low) complexity studied here.

3. A GA FOR CONSTRUCTING HYPERSTRUCTURES

3.1. Similarity Searching. A common measure of similarity between chemical structures is the size of the maximum common subgraph between two or more structures.[10-12] In graph-theoretical terms this problem is analogous to the NP-complete problem of *maximum common subgraph isomorphism*. The GA of the previous section could easily be adapted to solve this problem. However, as in the case of substructure searching, there are already a variety of established and efficient methods[4,13,14] that have the advantage of being deterministic.

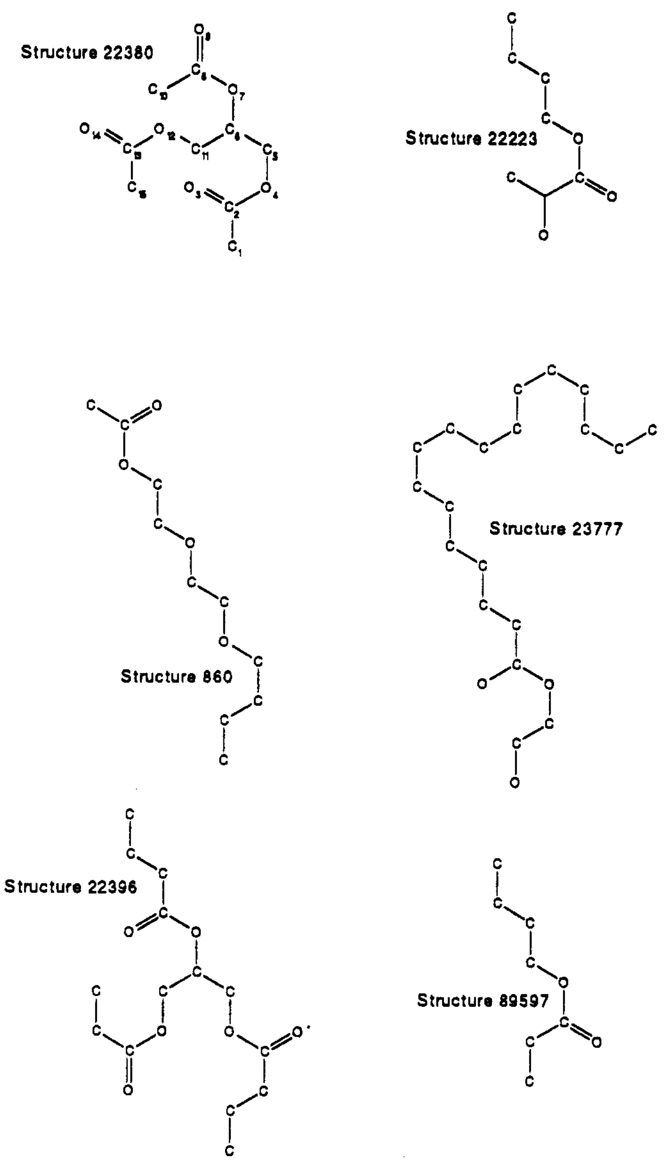

Figure 7. Example Cluster of Structures (Cluster No. 47 From The Work Described In [16]). The nodes in structure 22380 have been numbered to facilitate comparison with the hyperstructure shown in Figure 8.

A more promising similarity application for a GA is for the generation of *hyperstructures*. A hyperstructure is a single-structure representation for many structures that was first suggested by Vladutz and Gould[15] and has been studied more recently by Brown *et al.*[5,16]

A hyperstructure is a pseudomolecule that is formed by the superimposition of a set of molecules. Figure 8 is a hand-generated hyperstructure comprising all of the molecules shown in Figure 7 with the exception of structure 22380 (this structure is omitted as the GA described in the following sections attempts to superimpose this structure onto the incomplete hyperstructure). The hyperstructure encodes the connectivity of the cluster molecules with minimal redundancy, with each structure being contained as a substructure of the complete hyperstructure.

Brown *et al.*[5,16] have described the use of a *maximal overlap set* algorithm for generating hyperstructures. Unfortunately, it was found that there was no existing algorithm capable of performing this superimposition within any realistic time frame. The inefficiency of the available deterministic algorithms suggests that this application may be appropriate for the use of a GA: the substructure search GA, described in

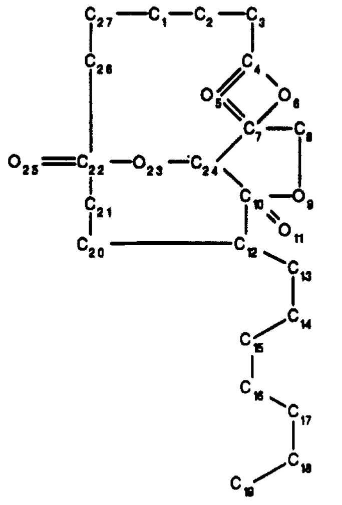

Figure 8. Hyperstructure from cluster 47, with the exception of structure 22380.

section 2, was thus converted to a maximal overlap set (MOS) algorithm.

3.2. The Problem. In order to test the feasibility of the GA an attempt was made to generate a meaningful mapping of structure 22380 in Figure 7 onto the hyperstructure in Figure 8.

The MOS is best understood in graph-theoretical terms. Both the hyperstructure and input structure can be considered as 2-D chemical graphs, with atoms as nodes and bonds forming graph edges. Given a mapping, M, the overlap set associated with that mapping is the set of input structure edges that match hyperstructure edges over M. Finding the MOS or largest-possible overlap set is an NP-complete problem. Once the MOS is found, the mapping associated with the MOS will be the best way of superimposing the input structure on top of the hyperstructure.

This problem is one that is beyond the power of conventional relaxation or backtrack algorithms.[11,13] The high complexity of the hyperstructure compared to normal chemical structures goes some way toward explaining why these algorithms, which perform so well in conventional structure handling, fail in this hyperstructure problem.[5] Not only do hyperstructures tend to be much larger and very much more highly connected than conventional molecules, hyperstructure atoms also do not obey the normal rules of valency (since any atom may be connected to any other atom); finally, any two hyperstructure atoms may be connected by more than one bond type.

3.3. Changes to the GA. The following changes were made to the GA for substructure search to determine the MOS and its associated mapping.

3.3.1. Encoding the Problem. The encoding mechanism described in section 2.2 remains the same. Chromosomes are thus strings of integer values, where the integer value is a hyperstructure node and the integer position represents an input structure node. The difference lies in the allowed values of the integers. Recall that, in section 2.2, query nodes can map only to structure nodes which are of the same atom type and which contain the same bond environment. In the MOS application, input structure nodes can map to any hyperstructure node of the same atom type. The restriction on bond environment is removed, since it has relevance only when the MOS comprises all of the input structure edges (and thus the structure is a substructure of the hyperstructure). Chemical MCS (or MOS) algorithms are normally used to match structures (or hyperstructures) on the basis of node-to-node equivalences, and the isomorphisms that are identified hence depend on the precise way in which the nodes have been labeled. As in the subgraph-isomorphism study reported in section 2, the work reported here characterized the nodes by the associated elemental type and the pattern of the pendant bonds, i.e., the bonded atom representation described by Adamson *et al.*[17] However, the GA is applicable to any level of node description that can be represented within the hyperstructure; for example, nodes could be characterized in terms of their hydrogen-bond donor or acceptor behavior, as used by Martin *et al.* in the DISCO system for pharmacophore mapping.[18]

3.3.2. The Fitness Evaluation Function. The fitness evaluation is similar to that described in section 2.3, with the exception that the set of query edges that map to structure edges do not have to form a connected graph. Thus a chromosome's fitness is the total number of query edges that map to structure edges over its mapping.

The definition of the node-set representation remains the same.

3.3.3. Mutation. Mutation proceeds as described in section 2.5. However, it only needs to be carried out once since the removal of the restriction that query nodes map to structure nodes with the same bond connections means that the mutation process can no longer produce unacceptable chromosomes.

3.3.4. Crossover. There is no change in the crossover operators described in section 2.6. However, there are a couple of points worth noting. As query nodes can map to any structure node of the same atom type (with no restrictions on bond environments), all of the child chromosomes produced by any of the crossover operators give rise to acceptable chromosomes, and there is thus no need for a consistency test.

Node-based crossover now proves to be much more effective. Since the sets of matching edges that give rise to a chromosome's fitness no longer need to form a connected graph, there is no need for overlap in the matching edge-sets of the parents. Thus the node-set representation of the children is invariably a combination of the node-set representations of the parents.

This has the effect that the average node-set in the population increases in size very quickly. A change in the heuristic (see below) means that chromosomes with a node-set size equal to the size of the chromosome are allowed in the population. Such a parent would merely duplicate itself during node-based crossover: in these cases, node-based crossover reverts to uniform crossover.

3.3.5. Heuristics. Since the object of this algorithm was to maximize the number of edges preserved over the mapping, rather than to find a substructure mapping, some changes were made to the method described in section 2.8. As before, if only one node is absent from the node set, an attempt was made to remap it, to increase the fitness of the chromosome; however, chromosomes which failed to show a substructure mapping were no longer discarded.

3.3.6. Dynamic Selection of Population Size. In order to cover the greater search space required by more complex problems, the algorithm requires a larger population size. Conversely, having too large a population when solving simple problems wastes resources and takes a long time to converge to the optimal solution. In this problem the structure to be mapped to the hyperstructure varied in complexity from simple five-atom molecules to large ring systems. For this reason an attempt was made to set the population size dynamically.

Obviously, it is not possible to predict the exact complexity of a given structure-to-hyperstructure mapping. In chemical databases the most common type of atom is carbon. Since carbon atoms in the input structure can map to any carbon in the hyperstructure, a good indication of the likely complexity of the problem is the number of carbon atoms in the hyperstructure. Let max be the number of carbon atoms in the hyperstructure: the population size was then defined as $10 + 10 \times \text{max}$. This number is somewhat arbitrary but does work in that it assigned larger population sizes to more complex matches.

3.3.7. Termination Conditions. The GA will only stop of its own accord if it can match all of the structure edges to hyperstructure edges. In order to accommodate those cases where the structure is not a substructure of the hyperstructure, the GA needs to terminate if there is no improvement in fitness over a significant number of operations.

Let baseops = $500 + 2 \times \text{max}^2$, where max is defined as above. The GA will always run for at least baseops operations. Suppose that the fitness increases after cnt operations. The GA will run for a further cnt operations without any improvement in fitness before terminating and returning the best mapping (if cnt is less than baseops, it will run for a further baseops operations, or if cnt is greater than ops_scale × baseops, it will run for ops_scale × baseops operations, where ops_scale = 10). There is no theoretical justification for these parameters, but they do seem to give reasonable results in practice (having been chosen on the basis of our initial experiments).

3.3.8. Provision for Special Cases. The large number of input structure/hyperstructure pairs means that the algorithm has to be able to cope with a number of atypical cases.

If a substructure mapping is generated at any time during population generation, that mapping is returned and the algorithm terminated. During population generation a count is kept of the number of chromosomes that are generated and then rejected (because they are of zero fitness, or are duplicates, or have their niches already occupied). If this count exceeds, by a factor of 12, the current number of chromosomes generated in the population, the population generation process is abandoned and the best mapping obtained so far returned. This measure prevents the algorithm hanging when it is unable to produce an initial population. For example, consider a hyperstructure comprising only ethane. If an attempt is made to map ethanol onto this hyperstructure, it will not be possible to generate a population.

It is often the case that an input structure will contain atom types not present in the hyperstructure or have more atoms of a particular type than the hyperstructure. Therefore, there will be no hyperstructure nodes to which the input structure nodes can be mapped. In this case the structure node is mapped to a "dummy" negative node. Such negative nodes are treated like normal nodes in mutation and crossover but play no part in the fitness process.

3.4. Example Run. The GA was run on the example hyperstructure and structure, with operator weights of 20 for mutation, 15 for node-based crossover, and 5 for uniform crossover. A maximal overlap was obtained in 0.98 s on a SUN 4/470 (this representing just 698 operations). The mapping is

$$\{\{1 \rightarrow 12\}, \{2 \rightarrow 10\}, \{3 \rightarrow 11\}, \{4 \rightarrow 9\}, \{5 \rightarrow 8\}, \{6 \rightarrow 24\}, \{7 \rightarrow 23\}, \{8 \rightarrow 22\}, \{9 \rightarrow 25\}, \{10 \rightarrow 21\}, \{11 \rightarrow 7\}, \{12 \rightarrow 6\}, \{13 \rightarrow 4\}, \{14 \rightarrow 5\}, \{15 \rightarrow 3\}\}$$

This result was considered very promising in that there is no known deterministic algorithm that is capable of generating this mapping within *any* reasonable amount of time.[5]

3.5. Results. Full details of the following results can be found in ref 16. Using a large data set of 10 794 structures from the EINECS (European Inventory of Existing Chemical Substance) database, hyperstructures were generated using both the GA and an *atom assignment* or AA algorithm. The AA method simply allocates input structure nodes to hyperstructure nodes on a first-come basis, with no account of bond environment. The GA was found to create much smaller hyperstructures (in terms of the number of edges), though construction times were much longer. More importantly, query search times were found to be twice as fast for hyperstructures built using the GA than the query search times for hyperstructures built using the AA method.

1. Mark all hyperstructure nodes as available and all query nodes as unused.
2. Select an unused query node, Q, at random.
3. If n, a random number, is greater than *Greedy_NodeP* then map Q to a randomly-selected available hyperstructure node of the correct atom type and goto step 5.
4. Map Q to the available hyperstructure node of the same atom type with the greatest similarity score. If there is more than one hyperstructure node available with that score than make a random choice between those nodes.
5. Mark the node that Q maps to as unavailable. Mark Q as used.
6. If there are any unused query nodes than go to Step 2.

Figure 9. Generation of greedy chromosomes.

3.6. Greedy Generation of Populations. A method was developed where an ordered list of candidate hyperstructure nodes was produced for each input node. This is based upon the method described by Brown *et al.*[5] for ordering the nodes in the hyperstructure prior to the backtrack search in the deterministic MOS algorithm.

These candidate lists, one for each input-structure node, are calculated from an analysis of the environments of nodes in each structure. The environment records the numbers and types of successively more distant neighboring atoms up to a predefined maximum radius. The environment of each node in both structures is encoded in a vector, and the asymmetric similarity coefficient[19] is calculated between all structure–hyperstructure node pairs. The asymmetric coefficient measures the inclusion of the structure atom environment within the more highly connected hyperstructure node environment. From these similarity coefficients a candidate list is produced for each input-structure node, listing all hyperstructure nodes of the same atomic type in order of decreasing similarity. The candidate lists, together with their associated similarity scores, are then passed to the GA, which uses this information to seed the initial population with some chromosomes which have a better-than-random fit.

A new GA parameter Greedy_ChomP was provided. During population generation a chromosome is created according to the greedy method with probability Greedy_ChomP. Otherwise it is generated normally. The method for generating a greedy chromosome is shown in Figure 9. The method requires a second new GA parameter, Greedy_NodeP, which is the probability that a position within a chromosome is allocated greedily rather than randomly.

The hyperstructure generation procedure was repeated using greedy initialization, with Greedy_ChromP = 0.8 and Greedy_NodeP = 0.8. Because the greedy initialization should reduce the search space, the population size and termination conditions (as described in sections 3.3.6 and 3.3.7) can be relaxed. The population size was redefined as $10 + 2 \times \text{max}$; baseops was redefined as $100 + 2 \times \text{max}^2$ and ops_scale as 2.

The results are detailed in ref 16. For hyperstructures built from clusters of chemical compounds the search times were up to five times faster than searches on the equivalent hyperstructures built using AA.

4. CONCLUSIONS

In this paper, we have discussed the use of GAs to compare the labeled graphs that are used to represent 2-D chemical compounds. We have developed ways of representing the node-to-node mappings that result from graph-matching algorithms in chromosome form and genetic operators that can be applied to these representations. The application of these methods to substructure searching proved to be of little value, owing to the inability of the GA to identify rapidly the absence of a substructure match and to the availability of an extremely efficient deterministic algorithm for the application. The methods were, however, far more successful when applied to the identification of the maximal overlap sets that are required for the generation of hyperstructures. Here, there is no efficient deterministic algorithm, and the GA was able to provide an effective means for hyperstructure generation.

GAs have now been applied to a wide range of applications in the field of chemical and biological computational sciences including conformational search of small molecules,[20] conformational analysis of DNA,[21,22] protein folding and engineering simulations,[23] chemometrics,[24] and molecular recognition.[25] We hence believe that the GA paradigm provides a valuable means of investigating a wide range of demanding matching problems in computational chemistry and are currently investigating several such problems in our laboratories.

ACKNOWLEDGMENT

We thank the Science and Engineering Research Council, the Wellcome Foundation, and Molecular Design Ltd. for funding; the European Community Joint Research Center for the provision of the EINECS data; Tripos Associates for the provision of hardware and software; and Geoff Downs and the reviewers for their helpful comments on an earlier version of this manuscript. The Krebs Institute for Biomolecular Research is a center for the Molecular Recognition Initiative of the Science and Engineering Research Council.

REFERENCES AND NOTES

(1) Holland, J. H. *Adaptation in Natural and Artificial Systems*; The University of Michigan Press: Ann Arbor, 1975.

(2) Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison-Wesley Publishing Co.: Wokingham, England, 1989.

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

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

(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) Oliver, I. M.; Smith, D. J.; Holland, J. R. C. A Study of Permutation Crossover Operators on the Travelling Salesman Problem. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*; Grefensette, J. J., Ed.; Lawrence Erlbaum Associates: London, 1987; pp 224–230.

(7) Syswerda, G. Uniform Crossover in Genetic Algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*; Schaffer, D., Ed.; Morgan Kaufmann: San Mateo, CA, 1989; pp 2–10.

(8) Goldberg, D. E.; Deb, K. An Investigation of Niche and Species Formation in Genetic Function Optimization. In *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*; Schaffer, D., Ed.; Morgan Kaufmann: San Mateo, CA, 1989; pp 42–50.

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

(10) Hagadone, T. R. Molecular Substructure Similarity Searching: Efficient Retrieval in Two-Dimensional Structure Databases. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 515–521.

(11) Bayada, D.; Simpson, R. W.; Johnson, A. P. An Algorithm for the Multiple Common Subgraph Problem. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 680–685.

(12) Brint, A. T.; Willett, P. Upperbound Procedures for the Identification of Similar Three-Dimensional Chemical Structures. *J. Comput.-Aided Mol. Des.* **1988**, *2*, 311–320.

(13) McGregor, J. J. Backtrack Search Algorithms and the Maximal Common Subgraph Problem. *Software-Pract. Exper.* **1982**, *12*, 23–34.

(14) McGregor, J. J.; P. Willett, P. Use of a Maximal Common Subgraph Algorithm in the Automatic Identification of the Ostensible Bond Changes Occurring in Chemical Reactions. *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 137–140.

(15) 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. E., Ed.; Springer Verlag: Berlin, 1988; pp 371–384.

(16) Brown, R. D.; Downs, G. M.; Jones, G.; Willett, P. A Hyperstructure Model for Chemical Structure Handling: Techniques for Substructure Searching. *J. Chem. Inf. Comput. Sci.*, paper appearing elsewhere in this issue.

(17) Adamson, G. W.; Lynch, M. F.; Town, W. G. Analysis of the Structural Characteristics of Chemical Compounds in a Large Computer-Based File. Part 2. Atom-Centred Fragments. *J. Chem. Soc. C* **1971**, 3702–3706.

(18) Martin, Y. C.; Bures, M. G.; Danaher, E. A.; DeLazzer, J.; Lico, I.; Pavlik, P. A. A Fast New Approach to Pharmacophore Mapping and Its Application to Dopaminergic and Benzodiazepine Agonists. *J. Comput.-Aided Mol. Des.* **1993**, *7*, 83–102.

(19) Salton, G.; McGill, M. J. *Introduction to Modern Information Retrieval*; McGraw Hill: London, 1983.

(20) Clark, D. E.; Jones, G.; Willett, P.; Kenny, P. W.; Glen, R. C. Pharmacophoric Pattern Matching in Files of Three-Dimensional Chemical Structures: Comparison of Conformational Searching Algorithms for Flexible Searching. *J. Chem. Inf. Comput. Sci.*, paper appearing elsewhere in this issue.

(21) Blommers, M. J.; Lucasius, C. B.; Kateman, G.; Kaptein, R. Conformational Analysis of a Dinucleotide Photodimer with the Aid of a Genetic Algorithm. *Biopolymers* **1992**, 32, 45–52.

(22) Lucasius, C. B.; Blommers, M. J.; Buydens, L. M. C.; Kateman, G. A Genetic Algorithm for Conformational Analysis of DNA. In *A Handbook of Genetic Algorithms*; Davis, L., Ed.; Van Nostrand Reinhold: New York, 1919; Chapter 18.

(23) Dandekar, T.; Argos, P. Potential of Genetic Algorithms in Protein Folding and Protein Engineering Simulations. *Protein Eng.* **1992**, *5*, 637–645.

(24) Lucasius, C. B.; Kateman, G. Application of Genetic Algorithms in Chemometrics. In *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*; Schaffer, D., Ed.; Morgan Kaufmann: San Mateo, CA, 1989; pp 170–176.

(25) Payne, A. W. R.; Glen, R. C. Molecular Recognition Using a Binary Genetic Search Algorithm. *J. Mol. Graphics* **1993**, *11*, 74–91.