An Algorithm for the Multiple Common Subgraph Problem

Denis M. Bayada, Richard W. Simpson, and A. Peter Johnson*

Maxwell Institute, School of Chemistry, University of Leeds, Leeds LS2 9JT, U.K.

Claude Laurenço

Centre CNRS-INSERM de Pharmacologie Endocrinologie, UMR 6 CNRS, rue de la Cardonille, 34094 Montpellier, France

Received July 7, 1992

The multiple largest common subgraph (MLCS) problem is presented, and a new algorithm and its implementation to solve this problem are described. The method consists of using a largest common subgraphs algorithm (LCSA) and a maximal common subgraphs algorithm (SUPLIMIT) that is a modification of LCSA. Because the largest common subgraph problem is a polynomial problem for trees, LCSA explores the spanning trees of the graphs and uses a backtracking method to be exhaustive in its search. A heuristic in LCSA gives numbers to the nodes of each graph such that nodes having a smaller number are more likely to be in a solution. LCSA, together with SUPLIMIT, is used in a series of pairwise graph comparisons to solve the MLCS problem. A preprocessing function is used to find a lower bound for the size of solutions to this problem. This improves the efficiency of the algorithm.

INTRODUCTION

In structure/activity relationship studies, the presence of a common structural pattern in a set of molecules could be responsible for their similar behavior.[1] If we describe the molecular bonding in terms of a graph, we can find such a common pattern by searching a set of molecular graphs for their common subgraphs. Finding the largest common subgraphs (LCS) for two graphs is a well-known problem,[2-6] but few algorithms have been designed to solve the multiple largest common subgraphs (MLCS) problem.[1,7] The algorithms of Varkony et al.[7] and Takahashi et al.[1] begin by choosing the smallest graph. From this, they take all the subgraphs containing just one edge and search for the presence of this subgraph in all the other graphs. If this is successful, the subgraph is grown (according to the connections in the first graph) and the process repeated until it is no longer possible to grow a subgraph compatible with all graphs. Our approach to solve the MLCS problem consists of using pairwise comparisons of graphs, i.e., we seek maximal common subgraphs (MCS[8-10]) and largest common subgraphs (LCS). This leads to a more efficient algorithm with significant savings in computer time.

In the present work, we define graphs and subgraphs and then specify the different isomorphism problems. Focusing on the LCS problem, we describe a backtracking algorithm with preprocessing function that finds the correct solutions. This algorithm uses new "favorable" numbers at each node of each graph and other heuristics to improve its efficiency. Then a multiple largest common subgraphs algorithm is exposed which incorporates a quick method to find an a priori minimal size for a solution. Finally, some results on different sets of molecular graphs are presented and discussed.

Graphs of Molecules. We represent the bonds and atoms of a molecule as a labeled graph, G(V, E, Lv, Le), where V is the set of vertices corresponding to the atoms and E is the set of edges corresponding to the bonds. Lv is a function Lv : $V \rightarrow N^+$ and Le is a function Le : $E \rightarrow N^+$. Lv (respectively Le) denotes the positive integer labels assigned to the vertices (respectively edges). This notation is similar to that used by Attias and Dubois.[11]

If we restrict ourselves to molecules (graphs) in which no atom (vertex) has more than 5 bonds (edges), then a molecular graph containing n nodes is degree-bound and has less than or equal to $5n/2$ edges. Moreover, the molecular graphs are planar except for rare molecules,[12,13] and it can be shown that the number of edges of planar graphs is less than or equal to $3n-6$. As we will show later, this linear bound on the number of edges with respect to the number of nodes of a molecular graph reduces the complexity of isomorphism problems.

A subgraph is defined in different ways in the chemical literature. It can be connected[1] or disconnected[2,3] and partial[1,3,7] or induced.[2,10] For a graph G(V, E) and one of its subgraphs g(v,e), these are defined as follows:

connected	There exists a path between all pairs of vertices, i.e., $\forall\ x, y, \in V, \exists\ p/p = (a_1, a_2, ..., a_i, ..., a_n)$ where: (i) $\forall\ i, a_i \in V$ (ii) $\forall i\ (1 \leq i < n), (a_i, a_{i+1}) \in E$ (iii) $a_1 = x, a_n = y$
disconnected	There may be pairs of vertices that are not connected by a path, i.e., G(V, E) is disconnected if: (i) $\exists\ (C_1, ..., C_i, ..., C_n)/G(V, E) = \sum C_i$ (ii) $\forall\ i, C_i$ is a connected graph (iii) $\forall\ i,j, C_i \cap C_j = \varnothing$
induced subgraph	A subset of nodes with all the edges that exist in the parent graph ($v \subset V$ and $e = E$)
partial subgraph	A set of nodes with a subset of edges from the parent graph ($v = V$ and $e \subset E$)

The connected induced subgraphs may be too restrictive to have a meaning in chemistry.[3] When bond changes occur

0095-2338/92/1632-0680$03.00/0

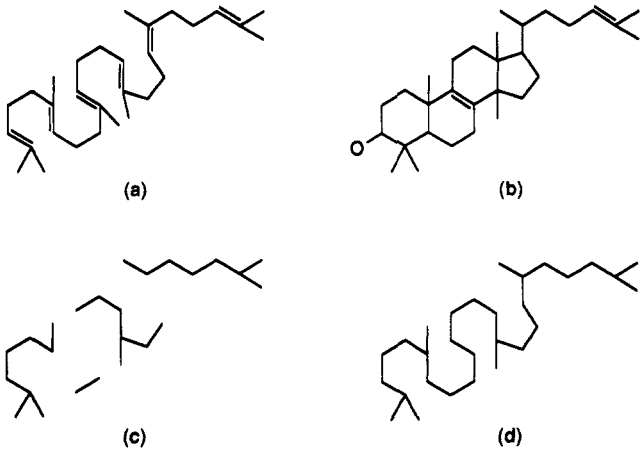

Figure 1. The starting material (a) and the target (b) have an unlabeled largest common partial subgraph (d) which shows the difference in four bonds, whereas their largest common induced subgraph (c) has lost important nodes.

during a reaction, the induced subgraph between the reactant and the product does not show bond changes but only differences between the sets of nodes, contrary to a partial subgraph (see Figure 1). Consequently, it is better to use disconnected partial subgraphs, even if in the present work only connected partial subgraphs are used. This restriction on connected subgraphs exists in the present program to lower the processing time and is not a limitation on the algorithm.

Isomorphism Problems. There are three main isomorphism problems. The first problem, P1, is known in chemistry as the canonical name problem.[14] It is a polynomial time problem[15] because the molecular graphs are degree-bounded. This problem is stated as follows:

(P1) The Isomorphism Problem

instance	Two labeled graphs $G_1(V, E, Lv, Le)$ and $G_2(V', E', Lv', Le')$.
question	Are G_1 and G_2 isomorphic, i.e., is there a one-to-one onto function $f: V \rightarrow V'/\{x, y\} \in E$ iff $\{f(x), f(y)\} \in E'$ and $Lv(x) = Lv'(f(x))$, $Lv(y) = Lv'(f(y))$, and $Le(\{x, y\}) = Le'(\{f(x), f(y)\})$?

The second problem, P2, is the subgraph problem[16,17] commonly used in substructure searching:

(P2) The Subgraph Problem

instance	Two labeled graphs G_1 and G_2.
question	Does G_1 contains a subgraph that is isomorphic to G_2?

Finally, the LCS problem, P3, is a common problem in synthetic organic chemistry.[5] For example, searching for appropriate starting materials[18] and identifying bond changes occurring within a reaction[3] are LCS problems. It can be shown that P3 can be solved in polynomial time for trees.[19] We are not aware of any proof that P3 is NP-complete for planar, connected, degree-bounded graphs (for example, molecular graphs). This is the reason why the LCS algorithm, described in the next section, is based on spanning trees.

(P3) The Largest Common Subgraph Problem

instance	Two labeled graphs G_1 and G_2 and a positive integer k where $k \leq$ minimum number of nodes of G_1 and G_2.
question	Does there exist a subgraph g_1 of G_1 isomorphic to a subgraph g_2 of G_2 such that the number of nodes of g_1 and $g_2 \geq k$?
comment	It is usually more useful to transform this decision problem into a search problem to output the largest common subgraphs, i.e., the pairs of largest g_1 and g_2. Largest in this context means containing the greatest number of nodes.

Some definitions of isomorphism include the notion of environment: A node i can "match" a node j if and only if they have the same environment,[2,4,10] i.e., degree(i) = degree(j) and (not always) all the edges that touch i must have the same labels as the edges that touch j. This type of match can be called an "exact match".

It is now possible to have a hierarchy of the problems:

$$P1 < P2 < P3$$

connected < disconnected, induced < partial and
exact match < normal match

"<" means easier in an algorithmic sense, i.e., faster on average for a computer. So an algorithm that searches LCS that are disconnected, partial, and with a normal match will find solutions "with more difficulty" than an algorithm that searches for "easier" subgraphs. If, for the graphs G_1 and G_2 we have $S_p = \{g/g$ is a subgraph of G_1 and G_2, and g has the property $p\}$, then it is straight forward that $S_{connected} \subset S_{disconnected}$, $S_{induced} \subset S_{partial}$ and $S_{exact} \subset S_{normal}$. In the same way, a positive answer to P1 implies a positive answer to P2 and P3. Moreover, a positive answer to P2 means a positive answer to P3. So one can see that P3 is the most difficult of the three isomorphism problems.

LCS ALGORITHM (LCSA) FOR TWO GRAPHS

The algorithm performs a breadth-first search on the two graphs, G_1 and G_2, and uses a backtracking algorithm[9] to find all the different solutions. As the search is a breadth-first search, the graphs are explored level by level. Initially LCSA combines the starting node, node 1, of G_1 with all possible nodes of G_2 and creates starting pairs at level 1. Then LCSA searches all the possible combinations of one level, takes one of them, and searches again all the combinations of the next level, etc. For each possible assignment, it checks the validity of the match (atom and bond types must be identical). We first state the notation used before showing an example:

combination	$[a_1, b_1, a_2, b_2, ...]$ is a list of nodes where $\forall\ i$, $a_i \in G_1$, $b_i \in G_2$, and a_i and b_i are matching nodes.
current solution	$(..., a_i, b_i; a_{i+1}, b_{i+1}, ...)$ is a list of combinations separated by a semicolon

Below is an example of execution of the algorithm based on Figure 2 (panels a and b):

level 1: starting pairs: all the nodes from G_2 that match with node 1 from G_1:
[1, I], [1, II], ..., [1, XI]
choose: [1, I]
level 2 from (1, I): if we match 1 with I, the next level combinations are:
[2, II, 3 III, 4, IV], [2, II, 3, IV, 4 III], [2, III, 3, II, 4 IV],

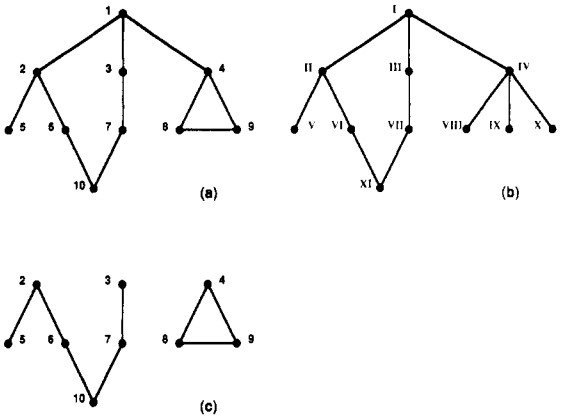

Figure 2. Two graphs, G_1 and G_2, (a and b, respectively) are shown with the nodes numbered for LCSA. When all solutions containing node 1 from G1 have been found, this node is eliminated leaving (c).

[2, III, 3, IV, 4 II], [2, IV, 3, II, 4, IV], [2, IV, 3, III, 4, II]
choose: [2, II, 3, III, 4, IV]
level 3 from (1, I; 2, II, 3, III, 4, IV):
[5, V, 6, VI, 7, VII, 8, VIII, 9, IX], [5, VI, 6, V, 7, VII, 8, VIII, 9, IX],
[5, V, 6 VI, 7, VII, 8, VIII, 9, X], ... (there are 12 combinations)
choose: [5, V, 6, VI, 7, VII, 8, VIII, 9, IX]
level 4 from (1, I; 2, II, 3, III, 4, IV; 5, V, 6, VI, 7, VII, 8, VIII, 9, IX):
[10, XI]
level 5 from (1, I; 2, II, 3, III, 4, IV; 5, V, 6, VI, 7, VII, 8, VIII, 9, IX; 10 XI):
nil.
Thus the previous level (4) contains a solution:
(1, I; 2, II, 3, III, 4, IV; 5, V, 6, VI, 7, VII, 8, VIII, 9, IX; 10, XI). This solution is stored and its size (10) is now the lower bound for solutions. The procedure backtracks to level 3 (looking for additional or better solutions)
choose: [5, VI, 6, V, 7, VII, 8, VIII, 9, IX] (the next combination).
level 4 from (1, I; 2, II, 3, III, 4, IV; 5, VI, 6, V, 7, VII, 8, VIII, 9, IX): [10, XI]; etc ...

When level 1 = nil then node 1 is eliminated (all the solutions containing node 1 have been found) and node number 2 becomes the new starting node. When the graph G_1 contains fewer nodes remaining than the best solution already found, the algorithm finishes.

Initially, a minimal size for a solution is given. It is called the limit. The user can set it to any required value, otherwise it is automatically set to 2. When a solution is found, it is checked that the solution is larger than or equal to the limit, and if it is larger, the limit is modified to be equal to the size of the new solution. If the solution is smaller than the limit, this solution is discarded. At any stage of the algorithm, the limit is a lower bound for the LCS solution. This dynamic modification of limit improves the efficiency of LCSA.

Connectivity. Each time a new starting node is chosen, it means that the old starting node has been eliminated from G_1. The algorithm checks if the graph G_1 is still connected (Figure 2c). If it is not, it seeks components having fewer nodes that the best solution already found and eliminates them. A graph can shrink rapidly if solutions are found from starting nodes of high degree because there is more chance of disconnecting a graph by eliminating these nodes.

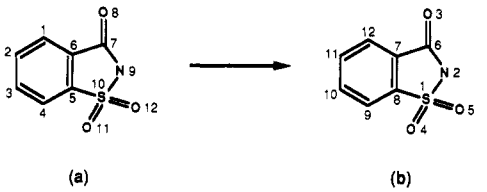

Figure 3. Example of graph renumbering to find the LCS more efficiently. The nodes of the graph (a) are numbered randomly, and the nodes of the graphs (b) are numbered according to the renumbering rules.

Renumbering. To improve the efficiency of the algorithm, we renumber the nodes in the molecular graph. This is important because the starting nodes are taken in numerical order, and at each level the nodes having a smaller number are combined first. This means that combination (1, I, 2, II, 3, III) is explored before (1, II, 2, III, 3, I). We have developed a preprocessing function to renumber each graph to give new favorable numbers to the nodes. Using these numbers we can find the best solutions rapidly. For a given node, the smaller the number, the greater the probability of that node being in the best solution. Thus, we need to be able to estimate the likelihood of a node being in the solution.

We consider that a node has a high probability of being in the best solutions if its atom type is rare, one of its bond types is rare, and its degree is high.

For a graph G, define the following quantities that correspond to these properties:

(1) $NT(x)$ is the number of nodes of the same type as x (a rare atom type will have a small value).
(2) $ET(x)$ is the occurrence in the graph of the rarest edge type that touches x [if one of the bonds to atom x is uncommon $ET(x)$ will be small].
(3) $ND(x)$ is the degree of the node x (simply the number of bonds to atom x).

A mark, $M(x)$, is given to the node x:

$$M(x) = 1000NT(x) + 10ET(x) + [5 - ND(x)] \quad \text{(i)}$$

In eq i the coefficients are chosen to give the appropriate significance to the functions $NT(x)$, $ET(x)$, and $ND(x)$.

Using a quick-sort, the nodes of each graph are reordered according to their $M(x)$ value and then renumbered according to their position in the ordered list. The first two criteria, $NT(x)$ and $ET(x)$, are not relevant when one seeks structural similarity only (treating all atom and bond types as identical).[4,5] Even so, by labeling edges differently in rings and chains, the second criterion is useful. An example of renumbering a molecular graph is shown in Figure 3.

Equivalent Nodes. The existence of equivalent nodes within a molecular graph leads to duplicate solutions. If we can identify these nodes, then we can improve the efficiency of our backtracking algorithm. We define equivalent nodes as follows:

Two nodes X_1 and X_2 are equivalent, if

(1) X_1 and X_2 are the same node or
(2) for all paths $(a_1a_2 \ldots a_n)$ there is a path $(b_1b_2 \ldots b_n)$ where a_1 is X_1, b_1 is X_2, and $\forall\, i\ 1 \le i \le n$ a_i is equivalent to b_i

In Figure 3 the two oxygens connected to the sulfur are equivalent.

As the nodes are visited in a breadth-first search, it is easy to control the sons of a node at each level. Thus, some

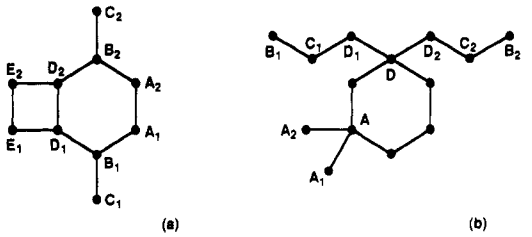

Figure 4. For the graphs a and b, X_1 is equivalent to X_2, with $X \in$ (A, B, C, D). Moreover, X is a common father of X_1 and X_2, with $X \in$ (A, D). In graph a, there are no equivalent nodes having a common father or a path leading to a common father, contrary to graph b where each node X_1 is equivalent to X_2 and has a path leading to a common father.

redundant combinations arising from two equivalent nodes, each having an identical path to a common father, are eliminated (see Figure 4).

It is not a trivial exercise to find these equivalent nodes. The algorithm we have developed is more robust than other similar methods.[18] It is based on an iterative cycle bounded by the length of the longest of the shortest paths in the graph. In each cycle, prime numbers are used to generate new equivalence marks for each node. After the procedure, the equivalent nodes have the same equivalence mark value. For the purposes of this discussion we define a cluster as a set of nodes having the same equivalence mark value. Initially, a number is assigned to each node according to its degree, its atom type, and the types of all the bonds that touch it; then the nodes are clustered according to their equivalence mark.

We assign a unique prime number to each cluster of nodes in the graph. We then enter a loop, as mentioned above, in which the numbers associated with a node and its immediate neighbors are multiplied together to give a new number for each node. The new numbers are clustered again and then replaced by a new set of unique prime numbers before iterating. After this procedure has been performed a number of times, the equivalent nodes belong to the same cluster. It can be shown that the length of the longest of the shortest paths is the minimum number of iterations required to guarantee that all the equivalent nodes will be clustered correctly. The equivalent nodes are in the same cluster at the end of this algorithm. In the present program this means when the node i is influenced by node j (and vice versa) with path ij being the longest of the shortest paths.

Unfortunately, there exist some cases where this procedure does not work. We are currently working on the possibility of assigning ring information to each node as well as its atom and bond types to improve the quality of the algorithm. This ring information will consist of the size of rings (if any) that are common to each pair of edges belonging to that node.

Modification of LCSA into SUPLIMIT. To find the partial subgraphs common to more than two graphs (MLCS), we need to have an algorithm that is able to search for solutions larger than a specified limit. This new algorithm is called SUPLIMIT. It produces a set of maximal common subgraphs (MCS) larger than the predefined limit, plus some graphs isomorphic to some of the MCS which match different nodes.

To create SUPLIMIT, it is just necessary to block, in the LCSA algorithm, the dynamic modification of limit (discussed above). The new algorithm, SUPLIMIT, now finishes when it is not possible to find any more solutions larger than the fixed, imposed limit.

Elimination of Solutions. There is a function in SUPLIMIT that eliminates some of the solutions that are not necessary. For instance the solution (1, I; 2, II) is included in (1, I; 2, II; 3, III) and so is unnecessary. There is also another method

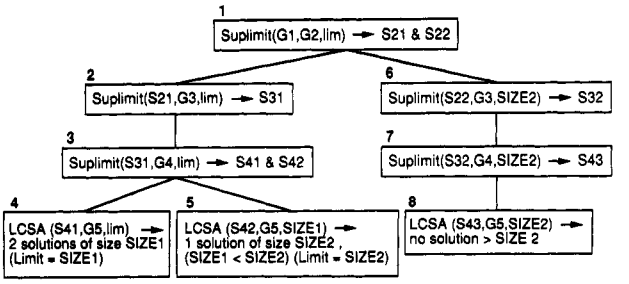

Figure 5. Example of execution of MLCS on five graphs G1, ..., G5. The numbers in bold indicate the order in which the program is executed. lim is the value of limit found during the preprocessing stage.

that we have developed that is similar but more drastic than the method used by Levi.[6] We define these two methods below with the notation of a solution specified above:

A is a solution	$(a_1, b_1; a_2, b_2; \ldots; a_n, b_n)$
B is a solution	$(c_1, d_1; c_2, d_2; \ldots; c_m, d_m)$
elimination 1	$n < m$ and $\forall\, i$: $(a_i, b_i) \in A \Rightarrow (a_i, b_i) \in B$ and so solution A is eliminated
elimination 2	$n \leq m$ and (a) $\forall\, i$, $(a_i, b_i) \in A \Rightarrow \exists\, x/(a_i, x) \in B$ or (b) $\forall\, i$, $(a_i, b_i) \in A \Rightarrow \exists\, x/(x, b_i) \in B$ and so solution A is eliminated

An example of elimination by elimination 2a is given as follows: If A = (1, I; 2, II) and B = (1, III; 2, IV; 3, V), then A is eliminated because $1 \in A$ and $1 \in B$, and $2 \in A$ and $2 \in B$. Occasionally the system of elimination 2 can eliminate potentially interesting solutions, but this is outweighted by the appreciable increase in speed gained by using this method.

MULTIPLE COMMON SUBSTRUCTURES

Now that we have the LCSA and SUPLIMIT algorithms, we are ready to expose the multiple largest common subgraph (MLCS) algorithm. This problem can be defined as follows:

(P4) The Multiple Largest Common Subgraph Problem

instance	A set of n labeled graphs $(G_1, G_2, \ldots, G_n)$ and a positive integer k where $k \leq$ minimum number of nodes of $(G_1, G_2, \ldots, G_n)$.
question	Does there exist a subgraph g_i for each G_i, such that $\forall\, i,j$ g_i is isomorphic to g_j and the number of nodes of $g_i \geq k$?
comment	It is indeed more useful to transform this decision problem into a search problem, i.e., to output the solutions as lists of LCS; one list $(g_1, g_2, \ldots, g_n)$ for each common subgraph found.

Varkony et al.[7] suggest that using an LCSA is an inefficient way of solving P4, because for n graphs, LCSA must be used $n(n-1)/2$ times. Actually, one can solve P4 by using the slightly modified version of LCSA, that is SUPLIMIT, and it is then possible to resolve this problem efficiently as will be shown later.

SUPLIMIT and LCSA have as arguments two graphs and a limit, and they produce solutions larger or equal to this limit. It is perhaps best to explain the algorithm by way of an example (as shown in Figure 5). If we take five graphs $(G_1, G_2, \ldots, G_5)$, and let S_{ij} represent the jth solution between the graphs G_{i-1} and G_i then the algorithm operates as follows:

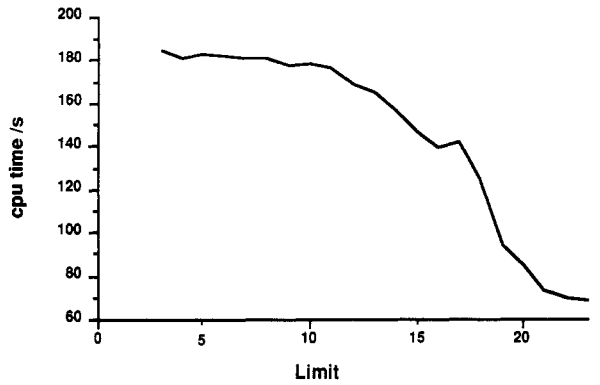

Figure 6. Time taken for the execution of SUPLIMIT on two marine sterols using different values for limit.

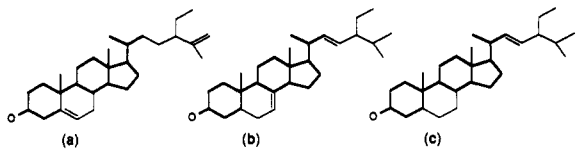

Figure 7. Three marine sterols are shown with the expected LCS emphasized.

We use SUPLIMIT on G_1 and G_2 once (step 1 in Figure 5), which produces two solutions S_{21} and S_{22}. Then we execute SUPLIMIT on S_{21} and G_3 (step 2). Similarly we execute SUPLIMIT on S_{31} and G_4 (step 3), and finally LCSA is executed on S_{41} and G_5 (step 4). The solutions it has found at this stage are stored, and the limit is modified to the size of the solutions if any is found larger than the previous limit. The algorithm backtracks to the previous level and executes again LCSA on S_{42} and G_5 (step 5). If any solution is found larger than the limit, the limit is dynamically modified again according to the size of the solutions, and the new solutions replace the ones found previously that were smaller. Again, the algorithm backtracks and looks for S_{43}. If it does not exist, it backtracks and looks for S_{32}. If only one solution was found at this level, it backtracks to level 1 and executes SUPLIMIT on G_3 and S_{22} (step 6), and so on. At the end (after step 8) a list of the largest common subgraphs remains. In the example given, only the solution found in step 5 is the MLCS.

When SUPLIMIT is used in the first steps of MLCS (1–4 in Figure 5), it is important that a suitable lower bound already exists. The cpu time required for SUPLIMIT, as a function of the value of limit, is shown in Figure 6. With a value of 3 for limit, SUPLIMIT is 3 times slower than with a value of 22. Thus, a quick preprocessing function is required to give a suitable lower bound for limit to be used by SUPLIMIT. We use LCSA several times to find this value; this is done for n graphs as follows:

LCSA is applied to G_1 and G_2 again on S_{21} and G_3, then on S_{31} and G_4, ..., finally on $S_{(n-1)1}$ and G_n. Eventually the size of the solution S_{n1} is the lower bound limit we require as S_{n1} is one common subgraph of the set of molecules.

RESULTS

We tested our program[20] on three examples taken from Varkony et al.[7] and Takahashi et al.[1]

(1) Marine Sterols. We applied the MLCS program to the three marine sterols shown in Figure 7. The results are shown in Table I. The size of the solutions (23 nodes) is larger than what one would expect (21 nodes), because the solutions found are not the chemically obvious ones (see Figure 8). If the ring bonds are labeled differently from the chain bonds, then the obvious solution is found rapidly (Table II). The huge number of solutions (Tables I and II) is due to the symmetry of partial subgraphs. This symmetry arises because a disconnected ring has the same number of nodes as a connected one, and so in breaking some rings, there are twice as many possibilities for each ring.

Table I.

limit	elimination	no. of solutions	size of solutions	cpu time/s
22 (automatic)	type 1	386	23	273
22 (automatic)	type 2	10	23	155
3	type 2	10	23	275

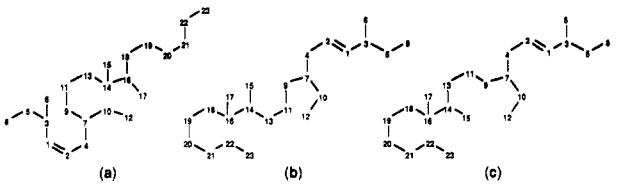

Figure 8. One of the solutions of the MLCS program on the three marine sterols of Figure 7. The nodes having the same number are matched together in this solution. It is interesting to notice the position of the nodes 1 and 2 of the first graph and to compare with the position of the same nodes in the two other graphs.

Table II.

limit	elimination	no. of solutions	size of solutions	cpu time/s
20 (automatic)	type 1	21	208	27
20 (automatic)	type 2	21	1	15
3	type 2	21	1	25

Figure 9. Three narcotics are shown with the MLCS highlighted.

Table III.

limit	elimination	no. of solutions	size of solutions	cpu time/s
13 (automatic)	type 1	8	14	1.3
13 (automatic)	type 2	1	14	0.7
3	type 2	1	14	1.5
3	type 1	8	14	41

(2) Narcotic Analgesics. Figure 9 and Table III show the data set (meperidine, morphine, and methadone) and the results produced by the MLCS program. The methadone molecule has two equivalent benzene rings having a common "father" and two equivalent methyl groups having a common "father" also. Because of the cutoff of combinations when equivalent nodes are detected, only eight solutions are found instead of 64 with the weakest elimination of solutions, and the cpu time is more than halved (1.3 s instead of 2.9 s).

(3) Macrolide Antibiotics. The 14 molecules shown in Figure 10 give rise to relatively simple graphs. The time taken to find the MLCS solution using the strongest elimination of solutions is relatively short (Table IV). However, the use of elimination 1 causes memory problems in the current version of the program due to the number of graphs involved. This could be easily avoided by allocating space more cautiously.

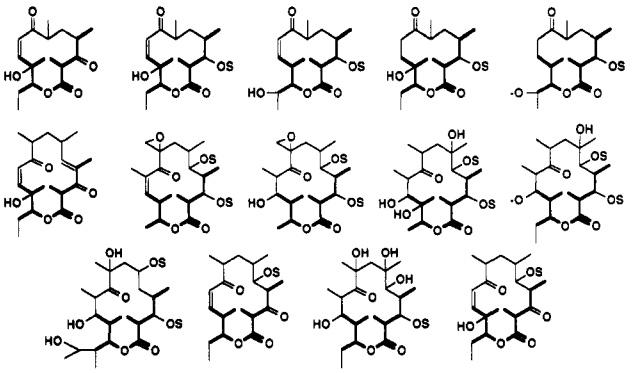

Figure 10. Selection of 14 macrolide antibiotics are shown; again, the MLCS solution found is highlighted. (Note S corresponds to sugar group.)

Table IV.

limit	elimination	no. of solutions	size of solutions	cpu time/s
12 (automatic)	type 2	1	13	17
3	type 2	1	13	24

CONCLUSION

We have presented a new algorithm able to find the largest subgraphs common to a set of molecular graphs. This algorithm uses a maximal common subgraphs (MCS) algorithm and a largest common subgraphs (LCS) algorithm. Several heuristics limit the processing time such as an algorithm that finds quickly a lower bound size for multiple largest common subgraphs. It is difficult to compare cpu times between different programs on different machines and written in different languages, but this program has the advantage of being significantly quicker than the previous published works on MLCS.[1,7]

To lower the complexity of the problem, one solution could be to associate a unique tree with each graph.[21] Since matching trees is easier than matching graphs, it should accelerate the whole process, leading to an even more efficient algorithm.

It is also possible to apply the algorithms described to a range of other chemical graph problems. This could include 3D subgraph detection for a specified set of molecules. In fact, the algorithms are sufficiently general to work in any situation where an appropriate LCSA already exists.

REFERENCES AND NOTES

(1) Takahashi, Y.; Satoh, Y.; Sasaki, S. Recognition of Largest Common Fragment among a Variety of Chemical Structures. *Anal. Sci.* **1987**, *3*, 23–28.
(2) Bersohn, M. An Algorithm for Finding the Intersection of Molecular Structures. *J. Chem. Soc., Perkin Trans. 1* **1982**, 631–637.
(3) McGregor, J. J.; Willett, P. Use of a Maximal Common Subgraph Algorithm in the Automatic Identification of the Ostensible Bond Changes Occuring in Chemical Reactions. *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 137–140.
(4) Johnson, A. P.; Marshall, C. Starting Material Oriented Retrosynthetic Analysis in the LHASA Program. 2. Mapping the SM and the Target Structures. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 418–425.
(5) Wipke, W. T.; Rogers, D. Artificial Intelligence in Organic Synthesis. SST: Starting Material Selection Strategies. An Application of Superstructure Search. *J. Chem. Inf. Comput. Sci.* **1984**, *24*, 71–81.
(6) Levi, G. A. Note on the Derivation of Maximal Common Subgraphs of Two Directed or Undirected Graphs. *Calcolo* **1972**, *9*, 341–352.
(7) Varkony, T.; Shiloach, Y.; Smith, H. Computer-Assisted Examination of Chemical Compounds for Structural Similarities. *J. Chem. Inf. Comput. Sci.* **1978**, *19*, 104–111.
(8) Tonnelier, C.; Jauffret, P.; Hanser, T.; Kaufmann, G. Machine Learning of Generic Reactions. III. An Efficient Algorithm for Maximal Common Substructure Determination. *Tetrahedron Comput. Methodol.* **1990**, *3* (6), 351–358.
(9) McGregor, J. J. Backtrack Search Algorithms and the Maximal Common Subgraph Problem. *Software-Pract. Exp.* **1982**, *12*, 23–34.
(10) Cone, M. M.; Venkataraghavan, R.; McLafferty, F. W. Molecular Structure Comparison Program for the Identification of Common Substructures. *J. Chem. Inf. Comput. Sci.* **1977**, *99*, 7668–7671.
(11) Attias, R.; Dubois, J.-E. Substructure System: Concepts and Classifications. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 2–7.
(12) Elk, S. B. Interpretation of Kuratowski's Theorem in Graph Theory as Both Topological Abstraction and a Chemical Reality. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 69–72.
(13) Benner, S. A.; Maggio, J. E.; Simmons, H. E. Rearrangement of a Geometrically Restricted Triepoxide to the First Topologically Nonplanar Molecule: A Reaction Path Elucidated by Using Oxygen Isotope Effects on Carbon-13 Chemical Shifts. *J. Am. Chem. Soc.* **1981**, *103*, 1581–1582.
(14) Morgan, H. L. The Generation of a Unique Description for Chemical Structures. A Technique Developed at Chemical Abstracts Service. *J. Chem. Doc.* **1965**, *5*, 107–113.
(15) Miller, G. Isomorphism of κ-Contractible Graphs. A Generalization of Bounded Valence and Bounded Genus. *Inf. Control* **1983**, *56*, 1–20.
(16) Harary, F. *Graph Theory*; Addison-Wesley: Reading, MA, 1969.
(17) Ullmann, J. R. An Algorithm for Subgraph Isomorphism. *J. ACM.* **1976**, *23* (1), 31–42.
(18) Marshall, C. Computer Assisted Design of Organic Synthesis. Ph.D. Thesis, University of Leeds, Leeds, U.K., 1984.
(19) Garey, M. R.; Johnson, D. S. *A Guide to the Theory of NP-Completeness*; Freeman & Co.: New York, 1979; p 202.
(20) The program has been written in C and runs on a VAX6310.
(21) Habib, M.; Maurer, M. C. On the X-Join Decomposition for Undirected Graphs. *J. Appl. Disc. Math.* **1979**, *2*, 75–90.