SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 12, 23–34 (1982)

Backtrack Search Algorithms and the Maximal Common Subgraph Problem

JAMES J. MCGREGOR
Department of Computer Science, University of Sheffield, Sheffield S10 2TN, U.K.

SUMMARY

Backtrack algorithms are applicable to a wide variety of problems. An efficient but readable version of such an algorithm is presented and its use in the problem of finding the maximal common subgraph of two graphs is described. Techniques available in this application area for ordering and pruning the backtrack search are discussed. This algorithm has been used successfully as a component of a program for analysing chemical reactions and enumerating the bond changes which have taken place.

KEY WORDS Backtrack search Search tree Maximal common subgraph

INTRODUCTION

Graph matching algorithms have been extensively studied for use in the analysis of chemical molecules and chemical reactions.[1–3] The methods discussed in the present paper were developed for use in a program for classifying chemical reactions. Given structural descriptions of the molecules present before and after the reaction, the program identifies the chemical bonds changed by the reaction. The present paper is not concerned with chemical details, which are given elsewhere,[4,5] and instead concentrates on the underlying abstract problem of determining the maximal common subgraph of two graphs. Relatively efficient solutions to this fundamental problem may find practical applications in many areas besides chemistry. For example, various police forces have attempted to match descriptions of crimes: when the descriptions, which may be in the form of graphs, are sufficiently similar, then the same gang may have committed the crimes. This and other such applications are not yet clearly established because the maximal common subgraph problem has hitherto appeared to be combinatorially prohibitive. As well as having the prospect of usefulness in many areas, the present work provides a case-study in combinatorial computing.

Levi[6] describes an algorithm for deriving the maximal common subgraph of two graphs and the use of this algorithm in comparing molecular structures has been discussed by Cone.[7] Levi's definition of the term 'maximal common subgraph' is inappropriate for the reaction classification problem. The present paper uses a more appropriate and generally useful measure of the similarity between two graphs, for which Levi's methods do not work.

We will introduce a maximal common subgraph algorithm that uses a backtrack search[8] as its basic component. Backtrack search, if used alone in a problem of this nature, is grotesquely inefficient. If, however, other techniques can be found for

0038–0644/82/010023–12$01.20

Received 16 November 1979

reducing the number of possibilities to be considered, backtrack search provides an effective means of systematically considering the remaining possibilities. The method described by Lynch and Willet[4] performs such a preliminary refinement for the reaction classification problem.

A further advantage of backtrack search in problems which involve finding a 'best' solution is that it is often possible to formulate tests which recognize at some stage that the current line of search cannot lead to a solution which is better than one found already. The use of such a 'pruning' technique requires consideration to be given to the order in which the various branches of the backtrack search tree are to be considered; the sooner 'good' solutions are found, the more effective the pruning will be. Pruning and search ordering techniques are an important component of our maximal common subgraph algorithm.

There is a close analogy between the searching, pruning and ordering techniques discussed here and those used by game playing programs in exploring a game tree. A similar analogy has been discussed by Slagle and Lee.[10]

Published descriptions of backtrack or depth-first search algorithms have usually been presented in a form which makes extensive use of **goto** statements and are therefore difficult to read and understand. More readable but less efficient versions make use of recursive techniques. In the next section we present a general description of a simple backtrack algorithm which is expressed iteratively in such a way as to combine efficiency with readability. Application of this algorithm to the maximal common subgraph problem is discussed in subsequent sections.

A SIMPLE BACKTRACK ALGORITHM

Consider the general problem of selecting a value for each of n variables, x_i $i=1, \ldots n$. The value for each x_i must be selected from a corresponding finite domain D_i. A set of values $(x_1, x_2, \ldots x_n)$ satisfying some specified conditions represents a solution to the problem. Many non-numerical problems can be expressed in this form and backtrack algorithms are widely used in solving such problems, usually in conjunction with other refinement techniques for reducing the number of combinations of values to be considered.

In Figure 1, we present the outline structure for a backtrack algorithm which systematically considers *all* possible combinations of values $(x_1, x_2, \ldots x_n)$ and tests each such combination of values to see if it is a solution. Before considering improvements in efficiency, let us illustrate how the execution of the figure 1 algorithm proceeds. Consider the simple example in which $n = 3$ $D_1 = \{1, 2\}, D_2 = \{3, 4, 5\}, D_3 = \{6, 7\}$. The algorithm of Figure 1 conducts a depth first search of the tree illustrated in Figure 2. Each arc of the tree represents a tentative decision about a value for a variable x_i. The nodes of the tree are numbered in the order in which they would be examined by the algorithm. Each execution of the main loop of the algorithm will *either* make a tentative assignment of a value to a variable x_i and increment i by 1 thus generating a new node in the search tree *or* backtrack to the node immediately above the current one in order to consider further alternatives for the previous variable ($i: = i-1$). The second course of action is taken when all alternatives for the current variable x_i have been tried.

At each step the backtrack algorithm attempts to extend the current *partial* solution $(x_1, x_2 \ldots x_{i-1})$ to a larger partial solution $(x_1, x_2, \ldots x_{i-1}, x_i)$ by selecting a value for x_i.

```
1.  i := 1; mark all values in D_1 as untried;
2.  repeat
3.    if there are any untried values for x_i in D_i then
4.    begin
5.      set x_i to an untried value; mark that value of D_i as tried;
6.      if i = n then
7.      begin
8.        if (x_1, x_2, ..., x_n) is a solution {or is the best solution so far} then
9.            print it out {or take a note of it}
10.     end
11.     else begin i := i+1; mark all elements of D_i as untried end
12.   end
13.   else i := i-1
14. until i = 0
```

Figure 1. Structure of a simple backtrack algorithm

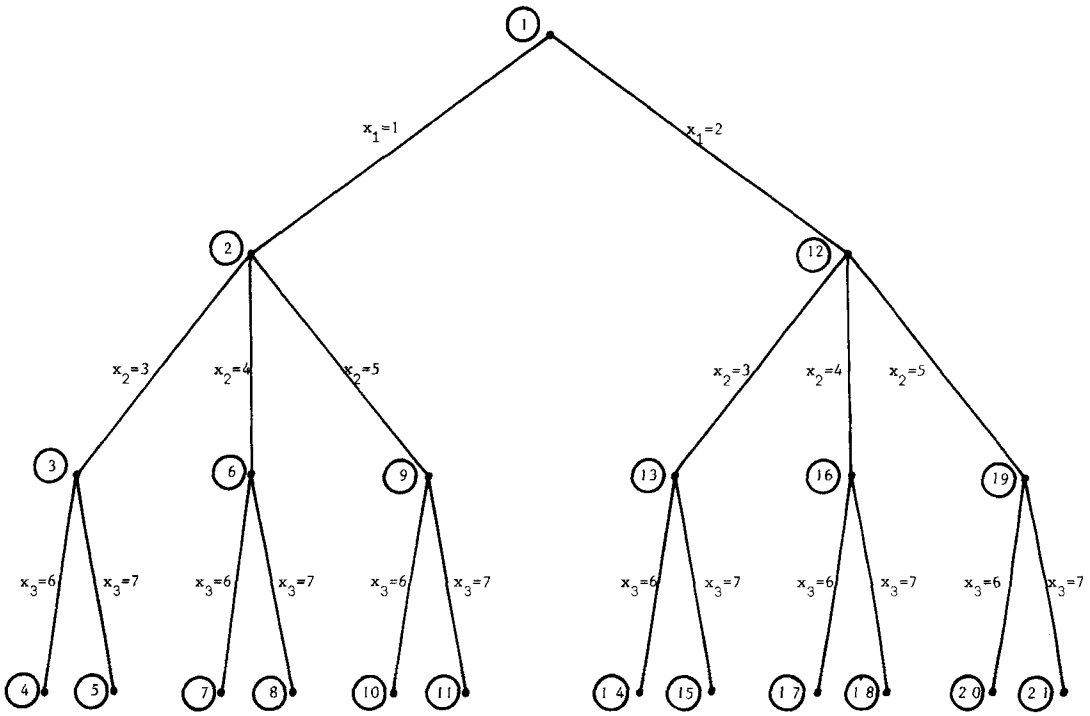

Figure 2. A tree search performed by the Figure 1 algorithm

An important feature of backtrack programming[8] is the ability to test these partial solutions and possibly eliminate large branches of the search tree from further consideration. If a suitable test can be formulated which can be used to recognize that a particular combination of values $(x_1, x_2, \ldots x_i)$ $i < n$ cannot form part of a complete solution $(x_1, x_2, \ldots, x_i, \ldots, x_n)$, then this test can be used after line 5 of Figure 1 to test whether the value x_i just selected is compatible with $x_1, x_2, \ldots, x_{i-1}$. If it is not, i can remain at its current value and a new value for the current variable x_i can be selected on the next execution of the main loop. (The same test can usually be used to recognize a solution when $i = n$.) The modified program structure required is illustrated in Figure 3. Consider the case where, in the simple example used above, we are seeking a solution (x_1, x_2, x_3) such that no two consecutive values have the same parity. A selected value of x_i, $i > 1$, can be compared with the value currently selected for x_{i-1} and if both are even or both are odd the current branch of the search tree need not be developed further. The structure of the search tree which would then be explored in this example is illustrated in Figure 4.

```
1.  i := 1; mark all values in D1 as untried.
2.  repeat
3.    if there are any untried values for xi in Di then
4.    begin
5.      set xi to an untried value; mark that value of Di as tried;
6.      if (x1, x2, ..., xi) is a valid partial solution then
7.        if i = n then print out {or take note of} (x1, x2, ..., xn)
8.        else begin i := i+1; mark all elements of Di as untried end
9.    end
10.   else i := i-1
11. until i = 0
```

Figure 3. Structure of a backtrack algorithm with tests on partial solutions

MAXIMAL COMMON SUBGRAPH—DEFINITIONS

A *graph* G consists of a set of points or *nodes* N, together with a set of *arcs* E connecting pairs of nodes ($E \subseteq N \times N$). Two nodes are referred to as *adjacent* if they are connected by an arc. Some simple graphs are illustrated in Figure 5. A *labelled* graph is one in which labels are associated with the nodes or the arcs. Figure 6 presents the formulae for two chemical molecules, before and after a reaction. These can clearly be interpreted as labelled graphs in which the nodes are labelled with atom names and the arcs are labelled with bond types. A *subgraph* of G is usually defined as a subset P of the nodes of G together with a subset F of the arcs connecting pairs of the nodes in P. ($P \subseteq N$ and $F \subseteq P \times P$).

Two graphs are *isomorphic* if they have the same structure: there is a correspondence or mapping between the nodes of one and the nodes of the other such that adjacent pairs of nodes in one graph are mapped to adjacent pairs of nodes in the other.

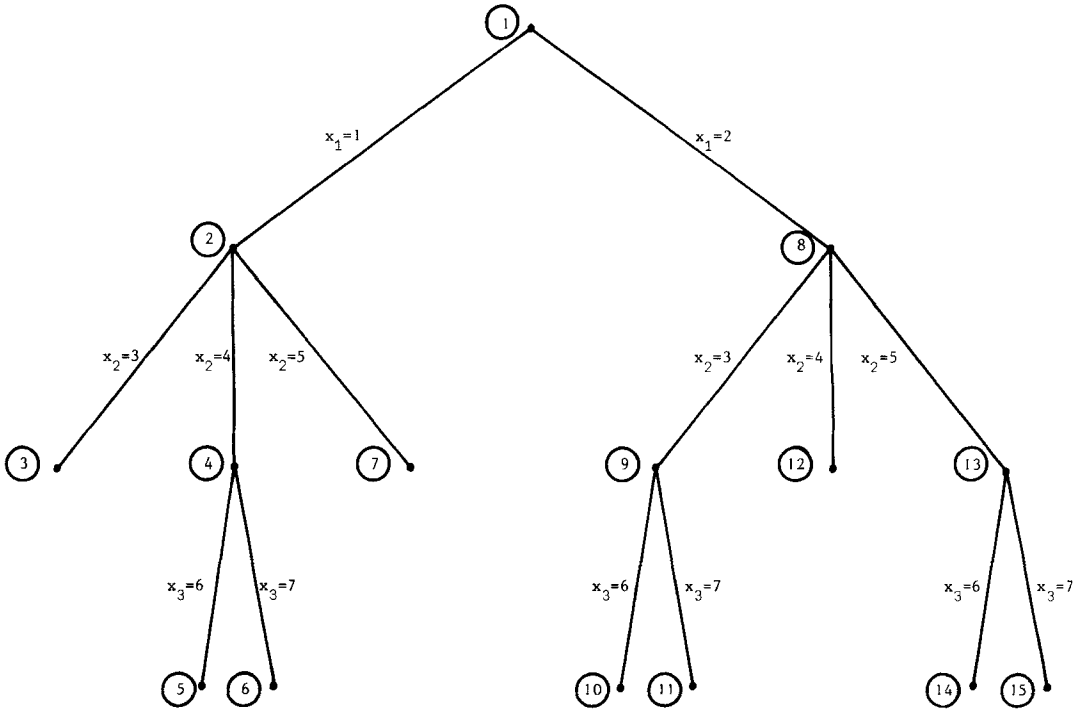

Figure 4. A tree search performed by the Figure 3 algorithm

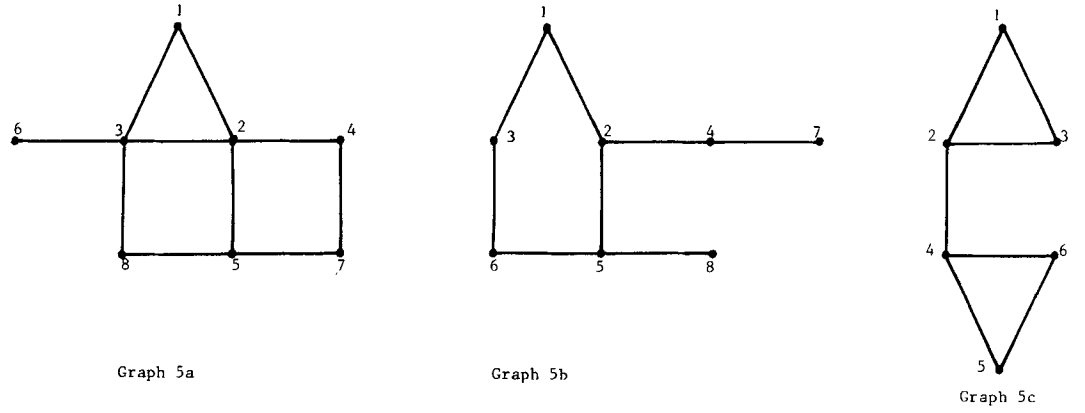

Figure 5. Simple graphs used to illustrate the notion of maximal common subgraph

A *common subgraph* of two graphs G_1 and G_2 consists of a subgraph H_1 of G_1 and a subgraph H_2 of G_2 such that H_1 is isomorphic to H_2.

Our definition of the term subgraph is the standard one used in graph theory[11] and in defining the *subgraph isomorphism* problem which has been widely discussed in the computing literature.[12, 13] The algorithm presented by Levi[6] for finding the maximal common subgraph of two graphs adopts a more restrictive definition of the term subgraph. His definition requires that, having selected the subset P of the nodes to be

included in the subgraph, all arcs connecting pairs of nodes in P are then included in F ($F = P \times P$). This makes the definition of the term *maximal* common subgraph easier and considerably restricts the number of possibilities which have to be considered in finding a maximal common subgraph; the method described by Levi is developed from the requirement that adjacent nodes in G_1 can correspond only to adjacent nodes in G_2 and non-adjacent nodes in G_1 can correspond only to non-adjacent nodes in G_2.

The weakness of Levi's definition when used as a measure of the similarity between two graphs can be illustrated by the graphs of Figure 5. Consider graphs 5a and 5b: using the above definition, the largest common subgraph which can be found consists of five nodes and four arcs from each graph, for example nodes (1, 3, 5, 6, 8) from 5a correspond to nodes (1, 2, 6, 4, 5) from 5b. If we now consider graphs 5a and 5c, we can for example make nodes (1, 2, 3, 4, 7) correspond to nodes (1, 2, 3, 4, 5) giving a common subgraph of 5 nodes and 5 arcs. This suggests that graph 5a is more similar to 5c than to 5b. We feel intuitively that the structure of graph 5a is more similar to that of 5b, and by using the more conventional definition of a subgraph, we can find a common subgraph of 5a and 5b which contains seven arcs.

Levi's definition suffers from similar disadvantages if used to compare the molecules of Figure 6. In order to detect the changes which have taken place during the reaction, we need to recognize that nodes (1, 2, 3, 4, 7, 8, 9) correspond to nodes (1, 2, 3, 4, 5, 6, 7) and therefore that bonds 3—5 and 6—7 in 6a have been broken and bond 3—5 in 6b has been formed. Levi's definition would not permit the inclusion of both nodes 3 and 5 from 6b in a common subgraph as there is no arc 3—7 in 6a to correspond to the arc 3—5 in 6b.

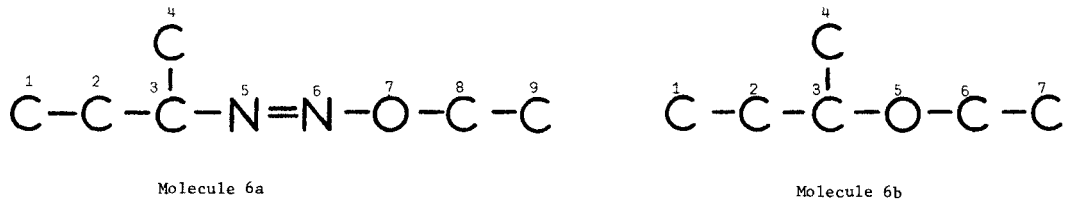

Figure 6. Structural formulae for the main molecules present on either side of the equation for a simple chemical reaction

Using the standard definition for the term subgraph, we can define a *maximal common subgraph* of two graphs to be the common subgraph which contains the largest possible number of arcs. A maximal common subgraph of graphs 5a and 5b is illustrated in Figure 7. The two subgraphs are isomorphic under the correspondence in which nodes (1, 2, 3, 4, 5, 6, 7, 8) are mapped to (1, 2, 3, 4, 5, 7, 8, 6). The isolated nodes 6 from graph 5a and 7 from graph 5b could be omitted without affecting the maximality of the common subgraph in terms of the number of arcs involved. However, it is convenient for such nodes to be included and our algorithm will find the maximal common subgraph according to the above definition which also contains the largest possible number of nodes.

The emphasis in the above definition on the number of arcs involved in the maximal common subgraph reflects the fact that it is the *structure* of the two graphs that is being compared. In the chemical reaction analysis problem, we wish to detect differences in structure.

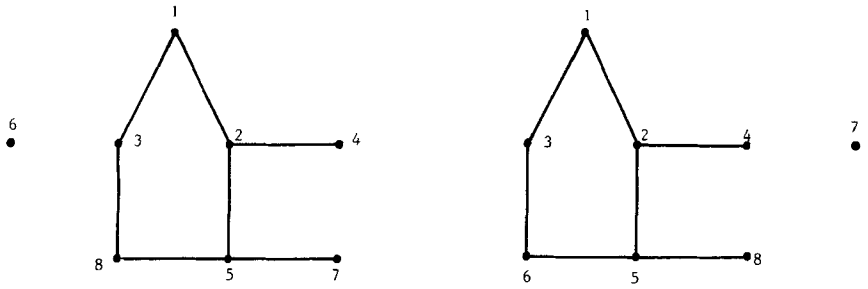

Figure 7. Maximal common subgraph of graphs 5a and 5b

A SIMPLE MAXIMAL COMMON SUBGRAPH ALGORITHM

The problem of finding a correspondence between the nodes of two graphs which satisfies certain conditions can easily be expressed as a backtrack programming problem of the type discussed earlier. The nodes of one graph are represented by the variables x_i, and a correspondence is defined by assigning to each of these variables a node of the second graph (or an indication that node i is to be omitted from the correspondence).

In this section we describe a simple backtrack algorithm for finding the maximal common subgraph of two *unlabelled* graphs G_1 and G_2. G_1 has p_1 nodes and q_1 arcs and G_2 has p_2 nodes and q_2 arcs. The basic algorithm described in this section is inefficient, and in subsequent sections we discuss the improvements which were required to make its use practicable on realistic problems. We shall assume throughout this section that $p_1 \leqslant p_2$ and that every node in G_1 must therefore be included in the correspondence.

Graph matching algorithms usually operate on the adjacency matrix representation of a graph: a graph with p nodes is represented by a $p \times p$ matrix of bits in which a one in row i and colum j indicates that node i is adjacent to node j. Such a bit matrix can be stored and manipulated efficiently by packing rows into computer words. A representation of the graphs in which the arcs appear more explicitly is required to enable our algorithm to keep track of the number of arc correspondences which result from a given node correspondence: the nodes and arcs are numbered independently and a graph of p nodes and q arcs is represented by a $p \times q$ bit matrix where a one in row i and column r indicates that *arc r* is connected to *node i*.

The arc correspondence resulting from a given node correspondence is constructed in a $q_1 \times q_2$ mapping matrix $MARCS$ which contains a bit in position (r,s) indicating whether or not arc r in G_1 is permitted to correspond to arc s in G_2. $MARCS$ initially contains all ones indicating that all possible arc correspondences are permitted. Each time a node in G_1 is tentatively paired with a node in G_2, $MARCS$ is refined on the basis of this node correspondence. Say node i in G_1 is tentatively paired with node j in G_2. Then any arc r connected to node i in G_1 can correspond only to arcs which are connected to node j in G_2. This is represented in $MARCS$ by setting to zero any bit (r,s) such that arc r is connected to node i in G_1 and arc s is not connected to node j in G_2. If, subsequently, an alternative correspondence for node i is tried, $MARCS$ will have to be restored to its state prior to the refinement mentioned above, in preparation for testing the effect of this alternative correspondence. The copying process required

to do this is a common feature of backtracking algorithms which modify data structures on the basis of tentative decisions, these modifications having to be undone when backtracking takes place.

We now present in Figure 8 the basic maximal common subgraph algorithm which finds the node correspondence which permits the correspondece of the maximal number of arcs. The variable *arcsleft* is used to keep track of the number of arcs which could still correspond in a node correspondence based on the current partial correspondence. Whenever refinement of *MARCS* results in a row being set to zero (this happens when two adjacent nodes in G_1 have been tentatively mapped to non-adjacent nodes in G_2) *arcsleft* is decremented by one. Whenever backtracking takes place, *arcsleft* has to be restored to its previous value.

The test at line 8 is used to recognize any partial correspondence which involves the elimination of more arcs from the set of possible arc correspondences than were eliminated in the best solution found so far. This comparison of partial solutions with complete solutions already found is a common feature of backtracking algorithms when used to find the 'best' of a set of possible solutions. Such tests will be more effective if branches of the backtrack search tree leading to 'good' solutions can be explored first and ways of doing this are discussed in a subsequent section.

```
1.  Set MARCS to contain all 1's; arcsleft := q1; bestarcsleft := 0;
2.  i := 1; mark all nodes of G2 as untried for node 1;
3.  repeat
4.    if there are any untried nodes in G2 to which node i of G1 may correspond then
5.    begin
6.      xi := one of these nodes; mark node xi as tried for node i;
7.      refine MARCS on the basis of this tentative correspondence for node i;
8.      if arcsleft > bestarcsleft then
9.        if i = p1 then
10.       begin take note of x1, x2, ..., xp1, MARCS; bestarcsleft := arcsleft end
11.       else
12.       begin
13.         store a copy of MARCS, arcsleft in the workspace associated with node i
14.         i := i+1;
15.         mark all nodes of G2 as untried for node i;
16.       end
17.   end
18.   else
19.   begin
20.     i := i-1
21.     restore MARCS, arcsleft from the workspace associated with node i
22.   end
23. until i = 0
```

Figure 8. The basic maximal common subgraph algorithm

LABELLED GRAPHS

In any realistic problem, the graphs to be analysed will be models of objects or relationships in some domain of interest. In our application area they represent chemical molecules. Graphs have also been used in picture processing to represent descriptions of scenes.[14,15] In such applications it is unlikely that the graphs will be unlabelled as was assumed in the previous section. When labelling information is available, it can be used to extensively reduce the number of possibilities which have to be considered by the backtrack algorithm.

Node labels can be used to perform a preliminary refinement on the set of nodes in G_2 to which each node in G_1 can correspond. Node i in G_1 can correspond only to nodes in G_2 whose labellings are identical to that of node i. This can result in enormous reduction on the potential size of the backtrack search tree.

One problem which now arises is that all nodes in G_1 will not necessarily be included in the correspondence even if $p_1 \leqslant p_2$. There may be more nodes in G_1 with label l than there are in G_2, and some of these nodes in G_1 will therefore have to be omitted from the correspondence. At each stage in the backtrack search, a further possibility available for node i is to omit it from the node correspondence. If there are p_1^l nodes in G_1 with label l and p_2^l nodes in G_2 with label l then at any stage in the backtrack search, a node in G_1 with label l can be tentatively omitted from the correspondence only if $p_1^l > p_2^l$ and then only if fewer than $p_1^l - p_2^l$ nodes from G_1 with label l have already been omitted. Thus a separate count of omitted nodes for each different label present must be maintained as backtracking proceeds.

Any arc labels present can be used to perform a preliminary refinement on *MARCS*. All bits (r,s) such that arc r in G_1 and arc s in G_2 have different labels, can be set to zero, thus indicating that these arcs cannot correspond in a solution.

ORDERING THE BACKTRACK SEARCH

We have already remarked that finding a good solution early on in the process of searching for a maximal common subgraph will strengthen the effects of the test at line 8 (Figure 8) and enable more cut-offs to take place during the search. Furthermore, in some applications such as pattern recognition it may be necessary to obtain only an approximate measure of the structural similarity of two graphs. In these applications, a good solution may be quite adequate as such a measure and if the search can be ordered so that good solutions tend to be found on the branches of the search tree which are explored first, then the backtracking process could be terminated before it runs to completion. In this section, we therefore discuss methods for guiding the backtrack search in such a way that good solutions will be found as early as possible in the search process.

Consider the stage in the backtrack search at which a partial correspondence for nodes $1, 2, \ldots, i$–1 of G_1 has just been constructed. The algorithm will now select one of the unpaired nodes from G_2 to correspond to node i and will then consider all possible combinations of correspondences for the remaining nodes in G_1 before backtracking and selecting a further node from G_2 to correspond to node i. The order in which alternatives are selected for node i will determine the overall order in which complete solutions are examined. Some information, which can be used for ordering the alternatives for node i at the above stage in the search, is contained in the partial solution already generated.

As an example, consider the two graphs of Figure 9. Say the search has tentatively paired nodes 1 and 2 of graph 9a with nodes 7 and 8 of graph 9b. Of the nodes in 9b which could now correspond to node 3 of 9a, node 6 will immediately create two corresponding arc pairs, node 9 will create one corresponding arc pair and the remaining nodes will not result immediately in any corresponding arc pairs. On the basis of the partial solution already generated, the most promising alternative for node 3 in 9a is to map it to node 6 of 8b.

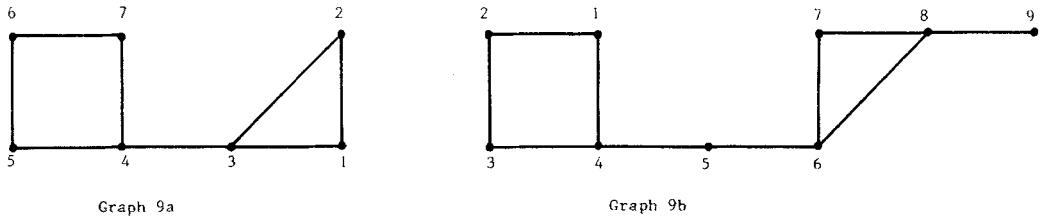

Figure 9. Graphs used to illustrate the effect of different orderings of the possible node pairing considered during the backtrack search

It should be noted that the choice resulting in the immediate creation of the largest number of arc correspondences will not always lead eventually to the best solution: consider the partial correspondence in which nodes 1, 2, 3 of graph 9a have been mapped to nodes 7, 8, 6 of graph 9b. Mapping node 4 in 9a to node 5 in 9b at that stage creates one new corresponding arc pair, whereas mapping node 4 in 9a to node 4 in 9b does not immediately create any corresponding arc pairs. It is however the latter mapping for node 4 of 9a which leads eventually to the maximal common subgraph for this pair of graphs. All possible correspondences must still be tried although the most promising ones should be tried first.

The above analysis suggests that, in theory, we could associate with each node in G_1 an ordered list of the nodes of G_2. The ordering on the list for node i of G_1 would indicate the order in which alternative mappings for node i should be considered. These ordered lists would have to be adjusted each time a tentative decision is made, and restored each time backtracking takes place. Performing such an operation at every stage in the backtrack search would be extremely expensive in terms of computing time required. There is, however, a useful compromise which can be adopted and which proved effective in our reaction analysis program.

During the backtrack search, each node i in G_1 has associated with it a 'priority subset' of the nodes in G_2 to which node i could correspond. Node j in G_2 is a member of this subset only if the nodes of the current partial correspondence which are adjacent to node i in G_1 have been tentatively mapped to nodes which are adjacent to node j in G_2. Pairing node i with one of these nodes will result in the immediate creation of the maximum possible number of corresponding arc pairs. Node i is always tentatively paired with each of the nodes in this subset before the remaining nodes from G_2 are tried.

The priority subsets can be represented by bit patterns and can be efficiently kept up to date as the search progresses by means of simple logical operations on these bit patterns. When a node i in G_1 is tentatively paired with a node j in G_2, the subsets associated with unpaired nodes in G_1 are updated as follows:

for *each node k of* G_1 *such that* $k > i$ *and k is adjacent to node i* **do**
priority subset $[K]$: $=$ *priority subset* $[l] \cap \{l : l$ *is adjacent to* j *in* $G_2\}$.

When backtracking subsequently takes place in order to try associating a new node of G_2 with node i, these subsets must be restored to their previous values and they therefore have to be copied into the workspace associated with node i before the above change is made.

Each priority subset could be initialized to include all the nodes of G_2. We would, however, prefer to have some better criterion to guide the backtrack search in its initial stages. An initial categorization of the nodes of G_1 and G_2 on the basis of the structure immediately surrounding each node could be used to identify promising correspondences, the priority subsets being initialized accordingly. A given node can be easily categorized according to, for example, the number of adjacent nodes. In the case of labelled graphs, the labels of the adjacent nodes and of the arcs connecting them to the given node could be taken into account. The analysis could be extended to include information about nodes connected to the given node by paths of length two or more. Such an analysis is described by Lynch and Willets[4] who used it to identify extensive isomorphic subgraphs of the molecules on either side of a reaction equation. In fact, in the reaction analysis problem, the set of correspondences suggested by their algorithm were used as an initial partial correspondence which our backtrack algorithm then extended to a full maximal common subgraph.[5]

The order in which nodes from G_1 are selected for consideration during the backtrack search can be chosen so as to optimize the effect of the above techniques for ordering the search. Once node i in G_1 has been selected and tentatively paired with a node in G_2, we have described how use is subsequently made of this pairing in ordering the selection of possibilities for the nodes which are connected to node i. We wish that optimum effect be obtained from the use of the information about the pairings which have been made on the current branch of the search tree. To do this, at each stage in the search the node in G_1 selected for consideration next should be the one which is adjacent in G_1 to a large number of the nodes which have already been selected and tentatively paired. The nodes in G_1 can be easily ordered so as to satisfy this requirement. This is done by selecting a node as a starting point, then selecting a node adjacent to the first node, and subsequently selecting at each step the node which is adjacent to the maximum number of previously selected nodes. In the reaction analysis problem, the set of nodes identified by the Lynch and Willets algorithm as a likely constituent of a maximal common subgraph was used as the starting point for the above ordering.

CONCLUSION

Backtrack programming techniques find application in many problem solving areas. These techniques can be effective if there is enough information available in the problem domain to guide and prune the search. Finding the maximal common subgraph of two graphs is in general a very difficult combinatorial problem. Despite this, the methods we have presented for applying a backtrack search algorithm to this problem proved effective in our particular application area—that of finding the maximal common subgraph of the two graphs representing the chemical molecules present before and after a reaction. The graphs involved were richly labelled and highly structured, and we feel that this is typical of many practical problem areas in which graphical representations can be used.

REFERENCES

1. M. F. Lynch, 'Storage and retrieval of information on chemical structures by computer', *Endeavour*, **27**(101), 68–73 (1968).
2. E. H. Sussenguth Jr., 'A graph-theoretical algorithm for matching chemical structures', *J. Chem. Doc.*, **5**, 36–43 (1965).
3. R. E. Tarjan, 'Graph algorithms in chemical computation', in R. E. Christofferson, (ed.), *Algorithms for Chemical Computations*, Am. Chem. Soc. Symposium Series, 46, 1–20 (1977).
4. M. F. Lynch and P. Willett, 'The automatic detection of chemical reaction sites', *J. Chem. Inf. Comp. Sci.*, **18**(3) (1978).
5. J. J. McGregor and P. Willett, 'Use of a maximal common subgraph algorithm in the automatic identification of the ostensible bond changes occurring in chemical reactions', *J. Chem. Inf. Comp. Sci.*, **21** (3) (1981).
6. G. Levi, 'A note on the derivation of maximal common subgraphs of two directed or undirected graphs', *Calcolo*, **9**, 1–12 (1972).
7. M. M. Cone, Rengachari Venkataraghaven, and F. W. McLafferty, 'Molecular structure comparison program for the identification of maximal common substructures', *J. Am. Chem. Soc.*, **99**(23), 7668–7671 (1977).
8. S. W. Golomb and L. D. Baumert, 'Backtrack programming', *J.ACM*, **12**(4) 516–524 (1965).
9. J. R. Slagle and J. K. Dixon, 'Experiments with some programs that search game trees', *J.ACM*, **16**(2), 189–207 (1969).
10. J. R. Slagle and R. C. T. Lee, 'Application of game tree searching techniques to sequential pattern recognition', *CACM*, **14**(2), 103–110 (1971).
11. F. Haray, *Graph Theory*, Addison-Wesley, 1969.
12. S. A. Cook, 'The complexity of theorem proving procedures', in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, 1971, pp. 151–158.
13. J. R. Ullmann, 'An algorithm for subgraph isomorphism', *J.ACM*, **23**(1), 31–42 (1976).
14. H. G. Barrow, A. P. Ambler and R. M. Burstall, 'Some techniques for recognising structures in pictures', in S. Watanabe (Ed.), *Frontiers of Pattern Recognition*, Academic Press, New York, 1972.
15. E. C. Freuder, 'Structural isomorphism of picture graphs', in C. H. Chen (Ed.), *Pattern Recognition and Artificial Intelligence*, Academic Press, New York, 1976.