(24) Hartigan, J. A.; Wong, M. A. "A K-Means Clustering Algorithm". *Appl. Stat.* **1979**, *28*, 100-108.
(25) Jarvis, R. A.; Patrick, E. A. "Clustering Using a Similarity Measure Based on Shared Nearest Neighbours". *IEEE Trans. Comput.* **1973**, *C-22*, 1025-1034.
(26) Willett, P. "Some Heuristics for Nearest-Neighbor Searching in Chemical Structure Files". *J. Chem. Inf. Comput. Sci.* **1983**, *23*, 22-25.
(27) Gowda, K. C.; Krishna, G. "Agglomerative Clustering Using the Concept of Mutual Nearest Neighbourhood". *Patt. Recognit.* **1978**, *10*, 105-112.
(28) Mizoguchi, R.; Shimura, M. "A Nonparametric Algorithm for Detecting Clusters Using Hierarchical Structure". *IEEE Trans. Patt. Anal. Mach. Intell.* **1980**, *PAMI-2*, 292-300.
(29) Murtagh, F. "A Review of Fast Techniques for Nearest Neighbour Searching". In International Association for Statistical Computing. "*COMPSTAT 1984*"; Physica-Verlag: Vienna, 1984.
(30) Olson, E. C.; Christoffersen, R. E. *Computer-Assisted Drug Design*; American Chemical Society: Washington, DC, 1979.
(31) Topliss, J. G. *Quantitative Structure-Activity Relationships of Drugs*; Academic: New York, 1983.
(32) Willett, P. "The Calculation of Intermolecular Similarity Coefficients Using an Inverted File Algorithm". *Anal. Chim. Acta* **1982**, *138*, 339-342.
(33) Croft, W. B. "Organizing and Searching Large Files of Document Descriptions". Ph.D. Thesis, University of Cambridge, 1978.

Computer Storage and Retrieval of Generic Chemical Structures in Patents. 7. Parallel Simulation of a Relaxation Algorithm for Chemical Substructure Search

VALERIE J. GILLET, STEPHEN M. WELFORD, MICHAEL F. LYNCH,* PETER WILLETT, JOHN M. BARNARD, and GEOFF M. DOWNS

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

GORDON MANSON and JON THOMPSON

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

Received February 7, 1986

A relaxation algorithm for chemical substructure search is simulated for implementation on general-purpose multiprocessors. An improved relaxation algorithm is described and the inherent parallelism detailed. The general-purpose simulation package PASSIM is described, and the methods used to simulate the algorithm are given. A variably sized pool of processors was assumed. The simulation was run on 71 structure/query pairs, and an average maximum speedup of 5.5 over a single processor was found, for approximately 20 processors. A great variation is found for individual structure/query pairs. The overall factor limiting the performance is the serial bottlenecks in the algorithm.

INTRODUCTION

This paper describes the results obtained with the simulation package PASSIM[1] to simulate the use of parallel processors in substructure matching of specific chemical structures by a relaxation technique. The study involved specific chemical structures only, as an initial step toward understanding the application of parallelism to the relaxation searching of files of generic chemical structures.[2]

Exact substructure matching for specific chemical structures is recognized as being very computationally demanding, involving establishing an atom-by-atom correspondence between the query and the file structure.[3] Screening systems have therefore been developed where the number of structures requiring an atom-by-atom matching is rapidly reduced by an approximate method.[4] The problems are much greater when a file of generic structures is searched. The variable nature of the structures is likely to result in less effective fragment screening and a more complicated and time-consuming atom-by-atom search. The relaxation algorithm developed at Sheffield by von Scholley[5] is more discriminating than fragment screening but not as computationally expensive, or as discriminating, as an atom-by-atom search. It was developed for screening generic structures after fragment screening.

Relaxation refers to a class of iterative methods for pattern matching. An initial mapping is made between two patterns by establishing a correspondence between the components of one pattern and the components of the second, according to some attributes of the components. The initial mapping is approximate and is refined by iterations that extend the range of local similarity and are repeated until no further refinement is possible, or until some components of one pattern can no longer be mapped onto the other pattern. In the past, relaxation methods have been used in areas such as image segmentation[6] and breaking of substitution ciphers.[7] Relaxation was suggested as a method of chemical substructure searching by Kitchen and Krishnamurthy[8] and Kitchen and Rosenfeld.[9] Here, the problem is one of subgraph isomorphism, where the patterns to be matched are connected graphs and the components used for establishing a mapping between the graphs may be the nodes and edges of each graph (corresponding to the atoms and bonds of a chemical structure). von Scholley's algorithm is based on ideas from Kitchen and Krishnamurthy.

Relaxation searching is well suited to matching patterns characterized by areas of variability, e.g., generic structures, and the inherent parallelism of the algorithm can be exploited to make this a viable search method for the large number of candidates passing fragment screening. The parallelism of this class of algorithms has already been noted[8] although neither tested nor simulated for chemical substructure searching.

Parallelism has already been investigated for some areas of nonnumerical chemistry. The CAS ONLINE search system[4] uses parallelism whereby the database of over 7 million compounds is divided equally among a series of pairs of PDP-11 minicomputers. A query is broadcast to each pair, a section of the database is searched serially by one of the PDP-11's, and candidates are passed to the second for atom-by-atom searching. A constant search speed can be maintained by adding further pairs of computers as the file increases in size.

Ullman[10] has proposed a parallel asynchronous solution for a subgraph isomorphism problem by placing a vital part of

0095-2338/86/1626-0118$01.50/0 © 1986 American Chemical Society

the algorithm into hardware for parallel implementation, but the hardware has not been built or simulated.

Wipke and Rogers[11] have described an investigation of parallelism in subgraph isomorphism by an atom-by-atom search. Normally, this is a backtracking process where atoms of the query are matched against atoms of the file structure; when a mismatch is found, the algorithm backtracks to its state when the last choice was made. A basis for parallelism is established by eliminating the need for backtracking. A task is generated for each possible match of an atom in the query against an atom in the file structure; i.e., M tasks are generated for mapping the first atom of the query onto each atom in the file structure, where M is the number of atoms in the file structure. Subtasks can be generated for mapping the second atom in a query and so on. If a task cannot map its assigned atom, it dies. The problem now becomes NP complete with respect to tasks, but each task is independent of the others and can therefore be freely allocated among processors.

The parallelism to be exploited here, in contrast to the methods used by CAS ONLINE and by Wipke and Rogers, is within the relaxation method itself, although suggestions are also given on how to tackle the macroparallelism contained within the entire file search.

There are two major classes of parallel algorithms.[12] One requires a SIMD (Single Instruction Multiple Data) machine such as the DAP,[13] where a single instruction is carried out on multiple data items simultaneously. This technique has been investigated in text retrieval by Pogue and Willett.[14] The other major class of parallel algorithms requires a MIMD (Multiple Instruction Multiple Data) machine such as Intel's Hypercube[15] or a machine based on Inmos' Transputers.[16] It was with Transputers in mind that the parallelism within the relaxation algorithm was investigated. Since Transputers were not yet available in quantities, and it is still unclear how these should be connected, a simulation of the parallel version of the relaxation algorithm was undertaken to determine the effect of introducing parallelism.

The aim of the simulation reported here was to demonstrate the feasibility of implementing relaxation in parallel, without being constrained by particular hardware configurations, and to give insight into possible optimization of the number of processors with respect to the execution time of the algorithm. A variably sized pool of processors was assumed, and each processor was assumed to be general-purpose with its own local memory. Although no particular processor was favored, these properties are those possessed by a pool of Transputers and by a number of other multiprocessor systems, e.g., Intel's Hypercube. Estimates were made of the communications costs during execution of the algorithm, and synchronization was achieved simply, by processes waiting in queues until they are able to continue.

The next section describes an improved serial implementation of von Scholley's relaxation algorithm and is followed by a section describing the simulation of the parallel algorithm. The results of the simulation are discussed in a final section.

IMPROVED SERIAL IMPLEMENTATION OF RELAXATION ALGORITHM

The algorithm as considered here compares a query structure with a single structure taken from a database, and the following discussion refers to the query structure as "query" and the file or database structure as "structure".

Although von Scholley's algorithm was developed for generic structures, the relaxation method applies equally well to specific structures. The algorithm was restructured and procedurized for this purpose. This produced greater insights into the functioning of the algorithm, leading to increased efficiency and making the inherent parallelism more apparent.

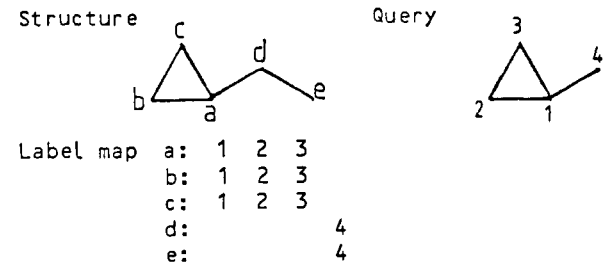


Figure 1. Initial label map formed by assignment of query nodes to each structure node according to two attributes: (1) node type (all nodes are carbon in this example); (2) bond type, i.e., ring or chain node.

In the implementation described here, the initial assignments of the query nodes to each of the structure nodes are made according to two attributes of the nodes, these are as follows: (1) node type, the nodes must have the same elemental values; (2) bond type, nodes forming part of a ring are ring nodes, with the remaining nodes as chain nodes. For correspondence, query and structure nodes must have the same bond type, a strict condition for substructure search. The assignments are stored in a set associated with each structure node, known as the "label set". The label sets of all structure nodes combine to form the "label map" of the structure. An initial label map is shown in Figure 1, and as an example, the label set associated with structure node b is {1 2 3}. The initial assignments are then refined iteratively until no further refinement is possible or until one or more of the query nodes can no longer be mapped onto a structure node. In the latter case, the result of the search is a mismatch; the former case indicates either a match or that the presence or absence of a mapping remains unresolved. The iterations attempt to eliminate labels from the label map of the structure, and three different processes are employed. (1) Labels can be eliminated from a structure node label set by examining the immediate neighbors of the nodes for each structure node:query node correspondence. For example, if for each neighbor of a corresponding query node there does not exist a matching neighbor of the structure node, this label is removed. (2) Labels can also be eliminated if the degree of the structure node is less than the degree of the corresponding query node. The degree of a structure node need not remain constant during processing. If during the iterations a neighbor of a structure node no longer has any query node correspondences, it cannot be included in a final mapping and can be removed, a process referred to as *pruning*. The degree of the structure node is then decremented. (3) If a query label occurs only once in the label map, then it can be assigned to one structure node, and any other labels that may be present in the label set of that structure node are eliminated.

A summary description of the algorithm is given in Figure 2 and is detailed below. The attributes for determining local similarity were as follows: node value, e.g., carbon, nitrogen, oxygen, etc; bond type, i.e., whether the node forms part of a ring; the degree of a node.

Initialization is carried out by InitialMapping, and assignments are made according to atom type and bond type. On exit from InitialMapping, the Flag is set to either Mismatch or to NowCheckDegrees. The Flag can be set to Mismatch at initialization if the query contains more nodes than the structure or if not all the query nodes have been assigned. If the Flag is set to Mismatch, the algorithm terminates; otherwise, the main part of the relaxation algorithm is entered in which iterative refinement of the mapping is made. The procedures that cause eliminations of query atoms from the label map are CheckNeighbours, CheckDegrees, and CheckAssignments.

CheckNeighbours is the most complex of these procedures and also the most computationally expensive. Each corre-

```
BEGIN
  Flag := NowCheckDegrees;
  InitialMapping;
  IF Flag = Mismatch
   THEN {no possible match}
    ELSE BEGIN
          REPEAT
            REPEAT
              EliminationsMade := false;
              CASE Flag OF
                NowCheckDegrees    : CheckDegrees;
                NowCheckNeighbours : CheckNeighbours;
              END;

              IF Eliminationsmade THEN CheckAllQatomsActive;

              CASE Flag OF
                Mismatch              : {no possible match}
                NowCheckNeighbours,
                NowCheckDegrees       : CASE EliminationsMade OF
                                          true  : BEGIN
                                                    PruneSmatrix;
                                                    CASE SmatrixPruned OF
                                                      true  : CASE Flag OF
                                                                NowCheckDegrees    : Flag := NowCheckNeighbours;
                                                                NowCheckNeighbours : Flag := NowCheckDegrees
                                                              END;
                                                      false : Flag := NowCheckNeighbours
                                                    END
                                                  END;
                                          false : CASE Flag OF
                                                    NowCheckDegrees    : Flag := NowCheckNeighbours;
                                                    NowCheckNeighbours : Flag := NowCheckAssignments
                                                  END
                                        END
              END;

            UNTIL not (Flag in [NowCheckDegrees, NowCheckNeighbours]);

            CASE Flag OF
              Mismatch           :;
              NowCheckAssignments : BEGIN
                                      CheckAssigments;
                                      CASE EliminationsMade OF
                                        true  : Flag := NowCheckNeighbours;
                                        false : Flag := Match
                                      END
                                    END
            END;

          UNTIL Flag in [Match, Mismatch]
        END
END.
```

Figure 2. Summary description of relaxation algorithm.

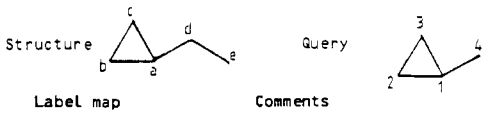
spondence between a structure node and a query node is examined, and the range of local similarity is extended to include immediate neighbors. The requirement for correspondence is that each immediate neighbor of the query node is associated with at least one immediate neighbor of the structure node and that each of the query neighbors can be associated with a different structure neighbor.

CheckDegrees takes each structure node/query node correspondence and eliminates a query node if it is of a higher degree than the corresponding structure node.

CheckAssignments examines the label map for each query node. If a query node is present only once in the label map, then it is uniquely assigned to that structure node: if this structure node contains any other query node correspondences, then these other correspondences are eliminated. The structure node and the query node can then both be flagged as Assigned and removed from subsequent processing.

CheckAllQatomsActive determines if all query nodes are present at least once in the label map. Any structure node with no query node correspondences is labeled as Pruned by PruneSmatrix and removed from further processing. Pruning also results in that structure node being removed as a neighbor of any other node to which it is adjacent, the degree of these nodes being decremented accordingly. Neither of these procedures causes eliminations from the label map, but CheckAllQatomsActive can result in the termination of the algorithm if a Mismatch is determined.

The algorithm consists of an outer loop and an inner loop. The inner loop repeats calls to either CheckNeighbours or CheckDegrees until no further eliminations are made from the label map. The outer loop then calls CheckAssignments: if eliminations are made here, then the outer loop is repeated; otherwise, termination results. The algorithm is terminated on one of the following conditions: (1) One or more of the query nodes has no correspondence with any structure nodes; i.e., a mismatch occurs. (2) No further refinements can be made. This can occur if there is a unique correspondence of each query node with a structure node; i.e., all query nodes are Assigned and the result of the search is a match. This can also occur if there is some remaining ambiguity where not all of the query nodes can be Assigned. If a structure node bears more than one query label, or if a label from the query is present more than once in the label map, the presence or absence of a mapping remains unresolved. Such conditions




```
   Label map                Comments

a: 1  2  3                  The Initial label map is made up of
b: 1  2  3                  five label sets; one for each structure
c: 1  2  3                  node.
d:          4
e:          4

                            [Flag := NowCheckDegrees]
a: 1  2  3
b:    2  3                  CheckDegrees: eliminates
c:    2  3                  (b:1, c:1)
d:          4               CheckAllQatomsActive: Yes
e:          4               PruneSmatrix: no effect.

                            [Flag := NowCheckNeighbours]
a: 1
b:    2  3                  CheckNeighbours: eliminates
c:    2  3                  (a:2, a:3, e:4)
d:       4                  CheckAllQatomsActive: Yes
                            PruneSmatrix: prunes e

                            [Flag := NowCheckDegrees]
a: 1
b:    2  3                  CheckDegrees: no
c:    2  3                  eliminations made
d:          4

                            [Flag := NowCheckNeighbours]
a: 1
b:    2  3                  CheckNeighbours: no
c:    2  3                  eliminations made
d:          4

                            [Flag := NowCheckAssignments]
a: 1
b:    2  3                  CheckAssignments:
c:    2  3                  (a:1, d:4)
d:          4               no eliminations made

                            [Flag := Match]

                Result = Match
```

Figure 3. A worked example of the relaxation algorithm for substructure matching, showing refinements made to the label map at each stage of the algorithm.

can arise for multiple occurrences of a query substructure or in some cases when no match exists, e.g., for very similar database structures and queries. Thus, cyclobutane will be retrieved by substructure searching with cyclopropane since the molecules are identical at a local level. Further computation is required to resolve these; additional characteristics of the graph could be included, the range of similarities could be extended to second- and third-order neighbors, or the structure could be passed as a candidate to a further searching technique.

To illustrate this, a worked example is given in Figure 3. The object is to attempt to find a mapping of the labels {1 2 3 4} of the nodes of the query into the labels {a b c d e} of the nodes of the structure. In the given example, such a mapping is nonunique because of symmetry.

CheckNeighbours and CheckDegrees are the procedures most appropriate for parallel implementation. This is demonstrated by the slightly expanded versions of the procedures shown in Figure 4. In each procedure, the outer loop processes each structure node in turn while the inner loop is repeated for each label contained in the label set of that structure node and examines each structure node/query node correspondence in turn. The processing of any one iteration of the outer loop is independent of the processing of any other iteration; i.e., each structure node can be processed independently and simultaneously. Also, the processing of any one iteration of the inner loop is independent of the processing of any other iteration, i.e., each structure node/query node correspondence can also be processed independently and simultaneously. Only the outer level of parallelism was investigated here for reasons explained under Results. As noted previously, CheckNeighbours is the most complex of the procedures and the most computationally expensive but results in the greatest number

```
PROCEDURE CheckNeighbours;

  FOR each structure node DO

    FOR each query node in the label set DO

      Check for consistency of neighbours for
      structure node : query node correspondence;


PROCEDURE CheckDegrees;

  FOR each structure node DO

    FOR each query node in the label set DO

      Check the degrees for
       structure node : query node correspondence;
```

Figure 4. Summary versions of procedures CheckNeighbours and CheckDegrees to show where parallelism exists.

of eliminations. Each process created from CheckNeighbours requires as its data the label set for the node being processed and the query structure record and entire current label map for reference.

CheckDegrees can also be implemented in parallel in a similar manner to CheckNeighbours with each structure node being processed independently. Each process here requires as its data the label set for the node, the degree of the node, and the degree of each query node contained in the label set.

Each procedure operates on the label map and so cannot begin until all processes generated from the previous procedure have terminated and the label map has been re-formed. The label map is re-formed after completion of both CheckNeighbours and CheckDegrees to determine if a mapping between the query and structure is still possible. This is performed by CheckAllQatomsActive, which looks for the presence of each query label in the label map.

SIMULATION OF PARALLEL RELAXATION ALGORITHM

PASSIM[1] is a Pascal system designed for discrete event simulation, which involves the analysis of queuing systems. Typical applications include analyzing a doctor's waiting room or ships arriving at a port. The system to be simulated is coded in a model specification. The model then describes the system in terms of its components, i.e., the objects of the system and the events to take place during the simulation. A Pascal program can be generated from the model and executed. The model coding system and the program system are related in appearance; this adds to the flexibility of the system, enabling additional features to be incorporated directly into the program if necessary. The three main components of a PASSIM simulation model are entities, queues, and activities. The entities are the objects of the system, e.g., doctors and patients, or nodes and processors in the context of substructure search by relaxation. Queues are groups of ordered entities awaiting some activity, e.g., the processor pool. Activities are what make things happen in the system; i.e., activities result in entities changing queues and may also cause the attributes of entities to be altered. Activities have a clearly defined beginning and end and take a definite period of time. The activities in this simulation are the processes that are generated from the procedures of the algorithm. One process is generated from each of the serial procedures, i.e., CheckAllQatomsActive, CheckAssignments, and PruneSmatrix. Several processes are generated from the procedures to be implemented in parallel, i.e., CheckDegrees and CheckNeighbours.

The Relaxation Model. The data flow chart of the relaxation model, given in Figure 5, resembles a PASSIM composite life

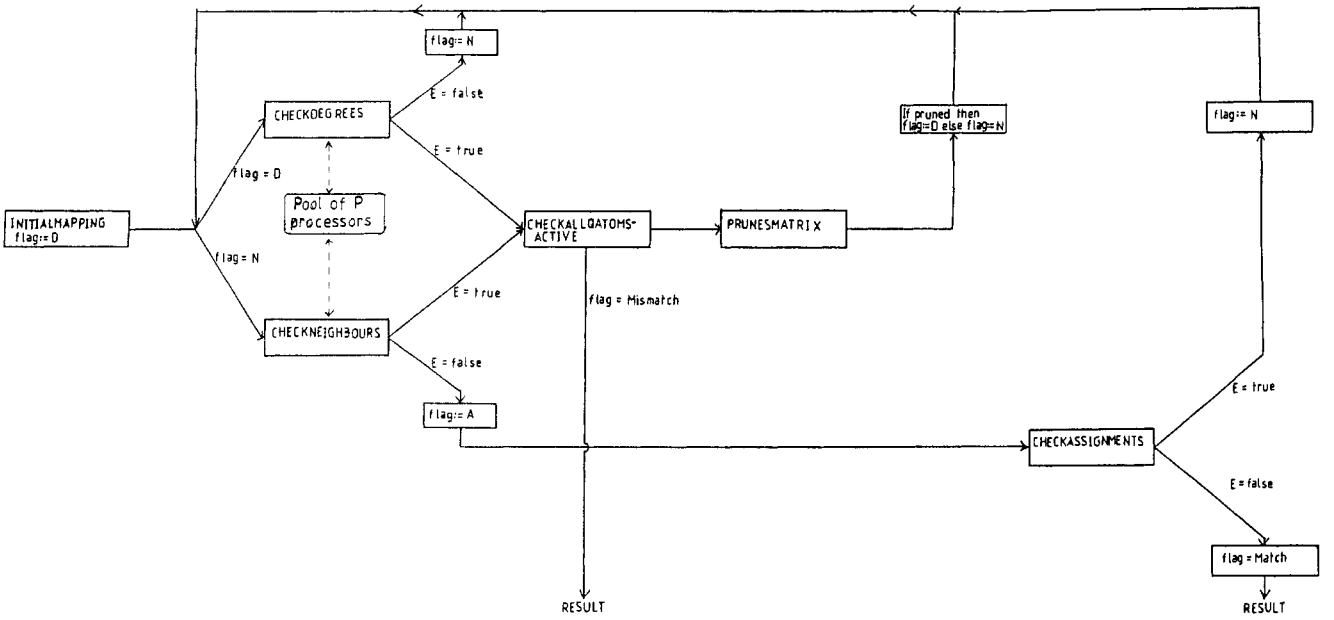


Figure 5. Model of relaxation algorithm. E represents the Boolean EliminationsMade. The values of the Flag are given as D representing NowCheckDegrees, N representing NowCheckNeighbours, and A representing NowCheckAssignments, Match and Mismatch. CheckNeighbours and CheckDegrees are multiple activities; each activity requires a processor from the pool.

cycle diagram with the activities enclosed in boxes, representing the procedures of the algorithm. The pool of P processors is shown in the diagram, but the queues of nodes awaiting some process are omitted for clarity. The entities of the system are the nodes of the file structure. CheckDegrees and CheckNeighbours are the procedures from which multiple processes are derived for parallel implementation and are included in PASSIM as multiple activities. Each activity requires one processor and one structure node. The remaining procedures are implemented serially and require a block of N_s nodes (where N_s is the number of nodes in the structure; this is also equivalent to the number of entities in the system) and one processor. In this simulation, the activities merely take entities from queues as specified, hold them for a duration, and then release them to destination queues. The duration of each activity is determined by analysis of the time spent in various components of the serial algorithm as described below.

The destination of each entity is determined by the value of the Flag and of the Boolean EliminationsMade. The Flag can take the value NowCheckDegrees, NowCheckNeighbours, NowCheckAssignments, Mismatch, or Match. Either or both of these variables may be altered by each procedure, as can be seen from the summary of the relaxation algorithm in Figure 2. Each activity must have been completed before the next can begin; i.e., all N_s nodes must have been acted upon before the simulation can continue. This condition holds because the current label map must be recombined before the execution of the next procedure can begin.

In order to simulate the algorithm, a number of structure/query pairs were selected. The serial algorithm was run with each pair, and timings were taken as described below. The path taken through the algorithm was determined and a model written. Where possible, the models were written in a general form with parameters controlling the destination queues of certain activities. The values of these parameters could then be specified in the data file so that recoding of the model was not always necessary for different structure/query pairs. Each simulation was run, incrementing the number of processors from 1 until the number of processors was equal to N_s. By monitoring of the processor pool, statistics on precursor utilization for each run were collected when all nodes had been placed on the last queue, i.e., when the algorithm had terminated.

Timings. All timings used in the simulation were based on running the serial algorithm on a Prime 2250 minicomputer. This has a 500-Hz interval clock generating a timing pulse every 2 ms. The duration of each activity was determined by summing the process duration together with an estimated communication cost. In order to determine the durations of processes, clocks were inserted at the beginning and end of each procedure, and the algorithm was run serially. To improve on the accuracy of the Prime 2250 clock, each call to a procedure was repeated 1000 times with a clock set to zero time before the first call and the time recorded following the last call. The recorded time was then divided by 1000. This also had the effect of negating the clock overheads. In determining a process duration for the procedures to be implemented in parallel, i.e., CheckNeighbours and CheckDegrees, which generate several processes, counters were included in the serial algorithm, and the number of processes generated was recorded. An average duration per process was then calculated. A large variance in process duration may have an effect on the simulation results, but this was not determined.

Estimates were made of the communication costs for each process. A data-transfer rate of 1 Mbyte/s inclusive of overheads was taken, based on an actual hardware device, namely, the Transputer.[16] Communication costs were estimated from the amount of data required by the processors for the different tasks. It was assumed that all the processors were preloaded with the program code and the query record, which remains invariant during processing. The relevant communication time was added to the duration calculated for each process and inserted in the model as the duration of the relevant activity.

RESULTS

A database of over 1000 specific structures, each exemplifying a particular patent, had been compiled earlier. Each structure, in turn, was used as a query against the entire database and searched by means of a prototype screening system. This screening system did not include any ring-screening information. Over 6000 candidate hits were obtained, and a random sample comprising 71 of these query/structure pairs was studied. Some characteristics of the structures are given in Table I.

Table I. Some Characteristics of Sample Query/Structure Pairs Studied, Showing Average Number of Atoms per Query and Structure and Their Elemental Composition

	no. of atoms per structure	
	av	SD
queries	15.4	10.1
structures	26.1	15.4

distribution of elements in sample (%)				
C	O	N	S	others
69.8	18.8	7.3	1.6	2.7

Table II. Mean and Standard Deviation from Mean (SD) for Speedup Obtained for Varying Numbers of Processors

no. of processors	mean speedup	SD
2	1.67	0.17
4	2.54	0.55
6	3.20	0.89
10	3.96	1.40
20	4.79	1.90
65	5.51	2.56

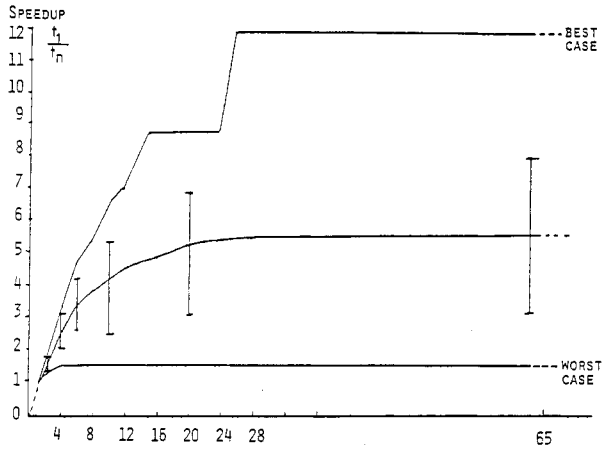


Figure 6. Graph of speedup against number of processors. Standard deviations are given as bars.

Performance Measures. A speedup ratio was calculated for each of the structure/query pairs for different numbers of processors. The speedup ratio can be defined as t_1/t_n, where t_1 is the time for the algorithm to execute with one processor and t_n the time to execute with n processors. Average processor utilization $[t_1/(nt_n)]$ was also calculated for each structure/query pair, for different numbers of processors. It is a measure of the proportion of time the processors spend doing useful work. t_1 was taken as the execution time of the parallel algorithm running with one processor, and so any overheads due to recoding the algorithm in a form appropriate for parallel implementation are ignored.

Table II shows the speedup calculated as above. A mean and standard deviation from the mean are calculated for different numbers of processors. These values are plotted in Figure 6, with the standard deviations shown as bars. Figure 7 shows a similar graph of mean processor utilization for varying numbers of processors. A general trend can be seen where the average speedup increases to a maximum of 5.5. The increase is rapid initially, with the addition of more processors resulting in significant speedup. The speedup then begins to level off at about 26 processors, with the addition of further processors having little effect. A 100% increase in the number of processors from 5 to 10 results in a 25% increase in performance, on average, whereas a 100% increase in the number of processors from 25 to 50 results in only a 6% increase in performance. The trend is amplified in Figure 7

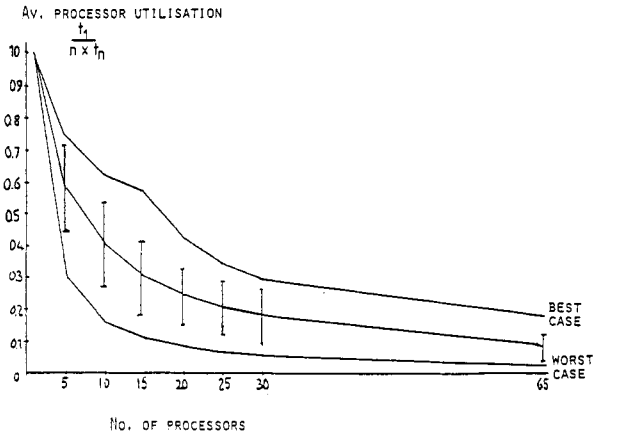


Figure 7. Graph of average processor utilization against number of processors. Standard deviations are given as bars.

with the addition of further processors, resulting in a decrease in average processor utilization. The rate of decrease is greater initially and then begins to level off at about 26 processors. At 10 processors, a 4-fold speedup is found at an average processor utilization of 0.4.

This result seems to be consistent with other studies of parallel algorithms. M. F. Deering[17] suggests that for most existing programs written in conventional computer languages the maximum parallelism seems to be about a 4-fold increase, the low figure being due to the style of programming forced by these languages. Wipke and Rogers quote a 24-fold speedup over a single processor for 25 processors, with very efficient processor utilization. The main difference would appear to be the nature of the parallelism involved. With the atom-by-atom search considered by Wipke and Rogers and described earlier, each of the initial $M \times N$ mappings from the structure nodes to query nodes initiates a separate task. Each task may subsequently give rise to a number of further tasks, while others may die. However, no synchronization or communication between tasks is necessary. In contrast, during relaxation, the parallelism is inherent in the algorithm, and the synchronization of processes and communication between them is essential. This synchronization results in a low processor utilization, and the serial nature of parts of the algorithm restricts the amount of parallelism possible. In both cases a further speedup could be calculated relative to the execution time of the best serial algorithm, rather than the parallel algorithm run with one processor, for a more concrete result. A comparison with the best serial algorithm will then take into account the overheads involved in recoding each algorithm in a parallel form.

Some optimization point can be found from the two graphs in Figures 6 and 7, its position based on speedup and numbers of processors and the cost of the latter. As the cost of processing elements is reduced and as MIMD machines are built with larger numbers of processors, a low processor utilization may then be more acceptable despite small increases in speedup.

No account was taken here of the scheduling of processors since it was assumed that each processor is dedicated to one process. Communication costs are treated at a simple level, without consideration of topologies of processors, limitations on the number of point-to-point connections between processors, or the routing of messages through the network.

Serial Bottlenecks. One significant factor affecting the performance of the parallel algorithm is the synchronization of processes. This often requires processors to sit idle waiting for other processes to finish their current activity. The synchronization is a requirement of the serial bottlenecks in the relaxation algorithm and arises from having to recombine the

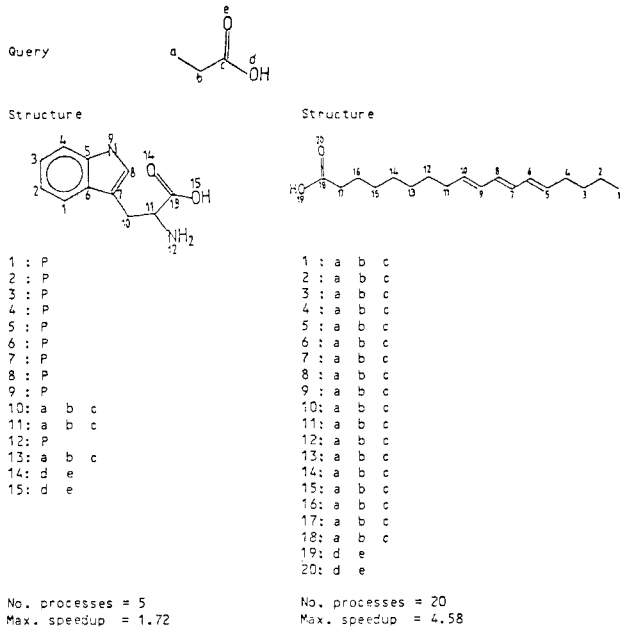


Figure 8. Effect on speedup of number of processes generated during processing. P represents a structure node that has been pruned from the label map.

label map from the updated label sets following each call to procedures CheckNeighbours and CheckDegrees. In the simulation, the maximum speedup for structure/query pairs is achieved when the number of processors is equal to the number of processes created initially by the parallel procedures, i.e., the amount of parallelism. In this limit all processors will be active during this initial call to the parallel procedure, although all but one of the processors will be inactive during the serial procedures, resulting in low processor utilization. In the limit of a single processor, it is active 100% of the time with the average processor utilization degrading with the addition of each further processor.

The inefficiency resulting from the serial bottlenecks could be offset by pipelining[12] searches where processing of a second structure/query pair begins before the first has been completed. This more efficient use of processors should lead to an increase in the average speedup per structure/query pair, although the control involved in pipelining structures may introduce additional overheads, the effects of which would need to be investigated.

The serial bottlenecks in the algorithm seem to be the most significant factor resulting in a low average speedup. The great variation in the behavior of individual structure/query pairs, evident from the standard deviations given, suggests that other factors are also important. Unfortunately, there does not appear to be a simple explanation for the magnitude of this variation, and it is attributed to several influencing factors, which are described below with illustrative examples. In the choosing of examples to demonstrate each factor, an attempt has been made to ensure that all other factors remain reasonably constant, although this is not always possible.

Number of Processes. The number of processes generated by the parallel procedures (i.e., the maximum degree of parallelism obtainable from each procedure) affects the maximum speedup achieved. In this implementation, the maximum number of processes possible is equal to the number of nodes in the structure, as each node is operated on independently. Pruning of structure nodes decreases the number of processes generated; this can occur at initialization and throughout processing. The rate at which the number of processes decreases contributes to speedup; as structure nodes are pruned or assigned, the absolute time for execution decreases, but the amount of parallelism that can be exploited also decreases. Examples demonstrating the effect are shown in Figure 8, in which the initial mappings only are given.

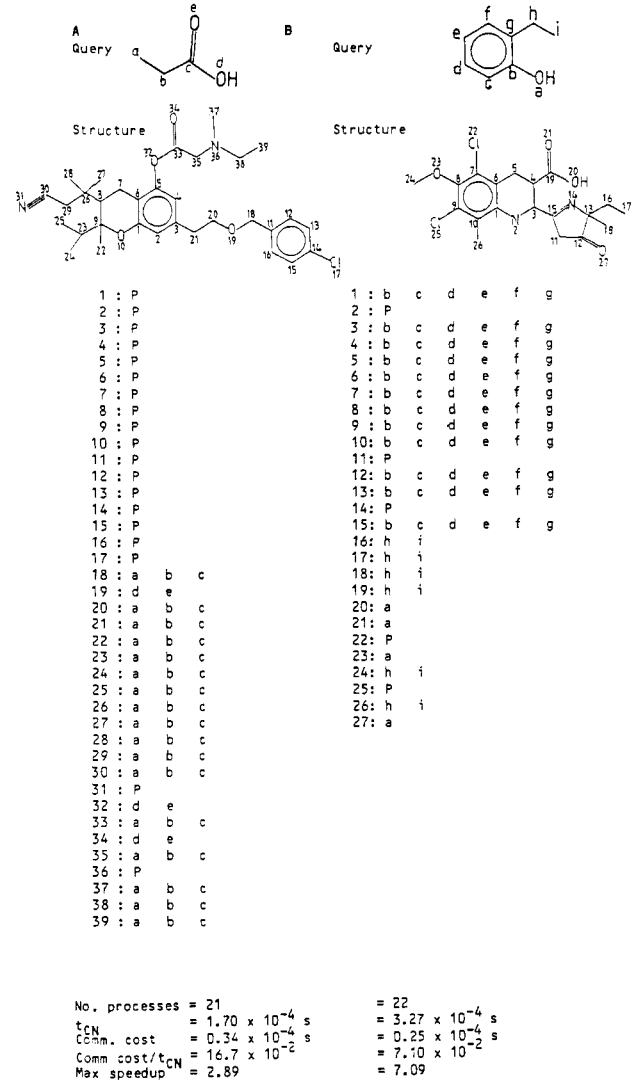
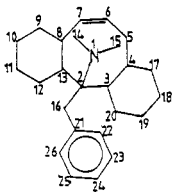


Figure 9. Larger granularity of processes for the Match B, combined with the smaller label map to be transferred between processors, tends toward a higher speedup.

Granularity. The granularity of a process is the amount of computation to be performed before the process has to communicate with other processes. The amount of computation performed by each process contributes to speedup. The larger the granularity, the less significant the communication costs become, relative to the execution time of the process. The execution time of each CheckNeighbours process was found to be several times larger than the execution time of each CheckDegrees process; therefore, only CheckNeighbours will be considered here. In addition, the communication costs for each CheckNeighbours process are greater than those for processes created by CheckDegrees. The entire current label map is required by each CheckNeighbours process, as well as information about the node being considered.

t_{CN} gives the average execution time of each CheckNeighbours process, and a ratio is calculated of communication time for each process, divided by the average execution time. Figure 9 shows that, in general, other factors remaining constant, the lower the ratio the greater the maximum speedup. This suggests that a large granularity for each process is desirable for good performance and that the processes should not be further subdivided, e.g., additional processes generated for each structure node/query node correspondence. The granularity



```
1:  a
2:  b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
3:  b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
4:  b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
5:  b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
6:  b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
7:  b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
8:  b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
9:  b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
10: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
11: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
12: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
13: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
14: n  o  p
15: n  o  p
16: n  o  p
17: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
18: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
19: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
20: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
21: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
22: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
23: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
24: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
25: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
26: b  c  d  e  f  g  h  i  j  k  l  m  q  r  s  t  u  v  w  x  y  z
```

Figure 10. Initial map only is shown to demonstrate number of labels in label map. Query and structure are the same molecule, and the labeling of the structure only is shown. "1" in the structure corresponds to "a" in the query, "2" to "b", and so on.

of each process generated from CheckNeighbours could be increased by introducing additional processing, e.g., checking the degrees of neighbors or checking for compatible bonds.

An average time per process was taken for each structure/query pair. This was calculated by accumulating the total time spent in CheckNeighbours and CheckDegrees, respectively, and then dividing by the total number of processes generated from each of the procedures. As the iterations proceed, the label maps are refined, and in general, the number of structure node/query node correspondences is reduced. This has the effect of reducing the granularity of processes as the iterations proceed. This was not taken into account in the simulation. A further simplification was to assume that the entire label map is always transferred between processors, although during processing the size of the label map may be reduced by pruning. The reduced granularity during processing tends toward a reduced speedup whereas the reduced communications overheads tend toward an increased speedup.

Path. It is not immediately clear how the path that the structure/query pairs take through the algorithm (i.e., the number of iterations) affects the maximum speedup attainable. The path generally depends on the degree of similarity between the structure and query; i.e., the more similar the pair, in terms of functional groups and bonding patterns (only cyclic and acyclic bonds are significant at present although more discrimination will be possible when further bond information is incorporated into the algorithm), the greater is the number of iterations required. Less common atoms are effective in leading to a rapid solution as are atoms of high degree, but a huge variation in speedup is found for structures following the same path. The similarity between structure and query is known to play a large part in determining the execution time of the serial algorithm. The similarity between the query and structure, in general, will depend on the effectiveness of the screening system. A prototype screening system without ring screens was used here, and better results may be obtained following a more effective screenout.

Result of Search. No correlation was found between speedup and whether the result of the search is a Match or Mismatch; although this does affect the path through the algorithm, a mismatch is generally found with fewer iterations than a match.

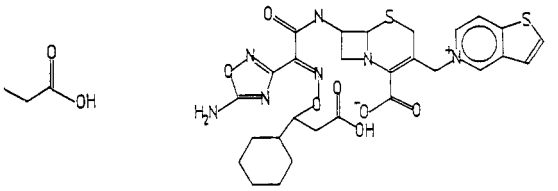


Figure 11. This query/structure pair achieved a maximum speedup of 1.6.

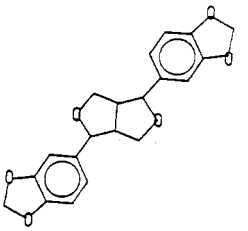


Figure 12. In this example, the query was searched against itself with a maximum speedup of 9.9.

Some Case Studies. Some interesting cases are discussed.

(i) See Figure 10; Maximum Speedup 11.9. Analysis of the path taken through the relaxation algorithm reveals that there were 12 iterations of the inner loop and two iterations of the outer loop. A total of 26 processes was created from the parallel procedures in the first iteration, one for each structure node. Also, the granularity of the processes is high with an average of 19 structure node/query node correspondences. This results in a low communication cost to the t_{CN} ratio (2.79×10^{-2}). The elimination of labels from the label map occurs relatively slowly and explains the many iterations necessary to arrive at a solution and that lead to a large absolute execution time. However, none of the nodes are pruned, and few are assigned in the early iterations, and so the total number of processes generated from CheckNeighbours and CheckDegrees remains large throughout. The amount of parallelism that can be exploited in each iteration remains correspondingly large, and hence, a large speedup is achieved.

(ii) See Figure 11; Maximum Speedup 1.63. The path taken through the algorithm shows seven iterations of the inner loop and two iterations of the outer loop. The poor results obtained from this pair seems to be due the size of the structure (45 nodes) and the small number of processes created, 14 in the first instance, although this is rapidly reduced. Each of the structure nodes that form part of a ring is pruned initially as all the query nodes are chain nodes. The labels are eliminated rapidly from the label map during processing. The result is a Match.

(iii) See Figure 12; Maximum Speedup 9.92. The molecule has a single axis of rotation, and all nodes are ring nodes. This means that an unambiguous assignment of query nodes is not possible, and iterations must continue until no further changes are made to the label map. In general, the more iterations to be performed the longer the time the serial algorithm takes to execute. As nodes can never be definitely assigned in this example, and because all bond types are ring, the number of processes generated (related to the amount of parallelism possible) and the granularity of the processes remains large. These factors tend toward a large speedup. Node type (i.e., whether the node is part of a ring or not) is a factor in determining initial assignments, and no query nodes can therefore be eliminated on this basis. The number of labels in the initial label map is therefore high and the ratio of communication costs to t_{CN} low (2.72×10^{-2}).

(iv) See Figure 13; Maximum Speedup 1.93. The file structure is characterized by a large number of nodes, but many of these are pruned initially, having no query node correspondences, leaving a small number of processes. This

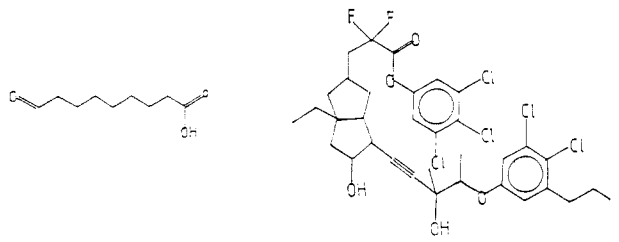


Figure 13. Maximum speedup of 1.9.

example can be compared with example ii. Here, the granularity is larger, perhaps accounting for a small improvement in performance although fewer iterations are required to obtain a result: one iteration of the outer loop and three iterations of the inner loop. The result is Mismatch.

CONCLUSIONS

For substructure searching of specific structures with the relaxation algorithm, the simulation demonstrates that, on average, a 5-fold increase in computational speed can be expected for approximately 20 processors. In some cases, the speedup is as much as 12-fold over the serial algorithm, and some of the factors responsible for this variation have been discussed. The degree of variation found for the small sample of specific structures investigated here makes projections of the performance as applied to searching a file of generic structures difficult, although some general observations can be made. Generic structures are more variable in nature and, taking into account all substituent groups, are generally much larger, consisting of many more nodes. The amount of computation to be performed is much greater, especially with generic queries, suggesting an improved performance with respect to speedup, although the larger volume of information associated with generic structures will result in higher communication costs. The simulation was designed as a feasibility study and, therefore, some assumptions were necessary in modeling the algorithm for generalized multiprocessor systems, for instance, the estimating of communication overheads. Further enhancements could be made to the simulation, but we are now working toward an actual implementation of the relaxation algorithm on networked Transputers.

ACKNOWLEDGMENT

The support of the Department of Education and Science through the award of an Information Science Studentship to V.J.G. is acknowledged. L. Kitchen is acknowledged for his useful comments.

REFERENCES AND NOTES

(1) Shearn, D. C. S. *PASSIM—A Pascal Simulation System*; Division of Economic Studies, University of Sheffield: Sheffield, U.K., 1982.
(2) Lynch, M. F.; Barnard, J. M.; Welford, S. M. "Computer Storage and Retrieval of Generic Chemical Structures in Patents. 1. Introduction and General Strategy". *J. Chem. Inf. Comput. Sci.* **1980**, *21*, 148-150.
(3) Ray, L. C.; Kirsch, R. A. "Finding Chemical Records by Digital Computers". *Science (Washington, D.C.)* **1957**, *126*, 814-819.
(4) Ash, J.; Chubb, P.; Ward, S.; Welford, S.; Willett, P. *Communication, Storage and Retrieval of Chemical Information*; Horwood: Chichester, 1985.
(5) von Scholley, A. "A Relaxation Algorithm for Generic Chemical Structure Screening". *J. Chem. Inf. Comput. Sci.* **1984**, *24*, 235-241.
(6) Kuschel, S. A.; Page, C. V. "Augmented Relaxation Labelling and Dynamic Relaxation Labelling". *IEEE Trans. Pattern Anal. Machine Intell.* **1982**, *PAMI-4*(6), 676-682.
(7) Peleg, S.; Rosenfeld, A. "Breaking Substitution Ciphers Using a Relaxation Algorithm". *Commun. ACM* **1979**, *22*(11), 598.
(8) Kitchen, L.; Krishnamurthy, E. V. "Fast, Parallel Relaxation Screening for Chemical Patent Database Search". *Chem. Inf. Comput. Sci.* **1982**, *22*, 44-48.
(9) Kitchen, L.; Rosenfeld, A. "Discrete Relaxation for Matching Relational Structures". *IEEE Trans. Syst. Man Cybernet.* **1979**, *SMC-9*, 896-874.
(10) Ullman, J. R. "An Algorithm for Subgraph Isomorphism". *J. ACM* **1976**, *23*(1), 31-42.
(11) Wipke, W. T.; Rogers, D. "Rapid Subgraph Search Using Parallelism". *J. Chem. Inf. Comput. Sci.* **1984**, *24*(4), 255-262.
(12) Kung, H. T. "The Structure of Parallel Algorithms". *Adv. Comput.* **1980**, *19*, 65-112.
(13) Gostick, R. W. "Software and Hardware Technology for the ICL DAP". *Aust. Comput. J.* **1981**, *13*(1), 1-6.
(14) Pogue, C.; Willett, P. "An Evaluation of Document Retrieval from Serial Files Using the ICL DAP". *Online Rev.* **1984**, *8*(6), 569-584.
(15) Wain, C. "Concurrent Computing: A New Age in Supercomputer Architecture". *Solutions (Intel Corp. Eur. Ed.)* **1985**, *May/June*, 6-8.
(16) Barron, I.; Cavill, P.; May, D. "Transputer Does 10 or More MIPS Even When Not Used in Parallel". *Electronics* **1983**, *Nov.*, 109-115.
(17) Deering, M. F. "Architectures for AI". *Byte* **1985**, *10*(5), 193-206.

Logical Extension of an Isomeric Pseudoconversion of Polycyclic Aromatic Hydrocarbons into Acyclic Polyenynes

SEYMOUR B. ELK
Elk Technical Associates, New Milford, New Jersey 07646

Received October 16, 1985

An isomerism between polycyclic aromatic hydrocarbons and ring assembly compounds formed by replacing selected rings with acetylenic linkages was noted, but not developed, by Dias. A far more complete isomerism, which is the logical extension of the referenced isomerism, is mathematically possible and is based on a reaction pathway using Dewar benzene intermediates. This technique is applied first to benzene and then to the general class of polycyclic aromatic hydrocarbons formed from six-membered rings, with the formulation of a concomitant nomenclature. Applicability of this method to the entire class of cata-condensed polycyclic aromatic hydrocarbons formed from six-membered rings is shown; also, some problems associated with trying to apply this technique to peri-condensed polycyclic aromatic hydrocarbons are described.

In the process of developing his "periodic table" for polycyclic aromatic hydrocarbons, Dias[1] used an isomerism of polycyclic aromatic hydrocarbons with a related set of alkynes.[2] This isomerism was only a minor part of Dias's focus, and no further development of this idea was presented. Nevertheless, this is the seed for the creation of a pseudoconversion scheme that uniquely correlates a given polycyclic aromatic hydrocarbon with a corresponding acyclic polyenyne. Furthermore, in order for such an isomerism to have significance, it is imperative that a chemically viable pathway between "reactants" and "products" exists—irrespective how energetically unlikely such a pathway may be. Consequently, ignoring the chemical

0095-2338/86/1626-0126$01.50/0 © 1986 American Chemical Society