Matching Algorithms For Handling Three Dimensional Molecular Co-ordinate Data

A thesis submitted for the degree of Doctor of Philosophy at Sheffield University

By

Andrew Timothy Brint

Department Of Information Studies July 1987

Abstract

Various techniques for efficiently handling three dimensional co-ordinate data are reviewed. In particular, four algorithms for substructure searching are compared, as are two methods for finding the largest common substructures between a set of molecules. A simulation followed by an implementation of a version of one of these common substructure finding algorithms on a parallel processor (transputer) system is also reported, with up to eleven transputers being used. Carrying on from this, a brief attempt at a transputer implementation of distance geometry is mentioned. Finally, a system for searching a file of one thousand molecular co-ordinates (taken from the Cambridge Crystallographic Data Bank) in order to find similar structures to a pattern molecule, is described. This system incorporates a screening stage using screens previously used for three dimensional substructure searching, before going on to a full comparison stage using one of the algorithms mentioned above. Throughout the above work, the emphasis was placed on the efficiency of the algorithms rather than on developing an integrated operational system.

Acknowledgements

The author received a research studentship from the Science And Engineering Research Council and is grateful to many people for their help during the project, including:-

my supervisor, Dr. Peter Willett,

Sue Jakes, for the use of programs to read the Cambridge Crystallographic Database and for details of the 3D screening system,

Dr. David Bawden, Jeremy Fisher, for providing ideas and data throughout the project. (Also Frank Woodward for writing the chemical structure drawing program used to illustrate Chapter 8),

Dr. Gordon Manson, George Wilson, for their help and use of their facilities when working with transputers.

CONTENTS

Note On Alterations

Several alterations were requested to be carried out by the examiners. Some of these entailed changing odd sentences or phrases so as to try to improve the readability, clarity and ease of comprehension of the thesis. As the original word processing facilities were no longer available, it was decided for reasons of neatness to mark the relevant sentences and paragraphs in the body of the thesis and to give the new versions in a section headed "Alterations" at the very end of the thesis.

For similar reasons, the requested extension to the final chapter was carried out by adding extra pages numbered 196a, 196b and 196c before the original last page of this chapter.

Andrew Brint October 1987

CHAPTER 1

CHEMICAL INFORMATION SYSTEMS IN DRUG DESIGN

Over the years, drug design has become increasingly more expensive with the trial and error methods involving the synthesis and testing of large numbers of compounds, becoming less likely to succeed. This is partly due to the more rigorous standards demanded from modern drugs with emphasis being put on their lack of side effects, and also probably because the more easily discoverable drugs (such as those like morphine which are extracted from plants) have already been found [Gund80, Aust84, Wyke87]. Therefore increasing emphasis has been put on trying to reduce the chance element in drug design and complex computer systems have been developed to aid in this task.

Typically the launching of a new medicinal product will have involved the synthesis and testing of about ten thousand compounds, have cost between 40 and 100 million dollars and have taken 10 years (with at least 2 years in the pre-clinical test tube and animal trials, and 5 in the clinical trials on test groups of humans) [Wyke87, Wool84]. Consequently, chemical information systems have been designed to try to lower this by using knowledge of some of the millions of compounds already known to the chemical community and also of the tens of thousands a pharmaceutical company has access to. This is achieved in a variety of ways, including

1) Novelty checking -the comparing of a structure against a database to find out whether any information is known about it.

2) Synthesis planning -trying to find a pathway from chemicals held or easily obtainable by a company, to a specified compound.

3) Identification of compounds containing substructures of interest.

4) Prediction of compounds' properties.

The net effect has been a growing use of computers to assist in the search for new drugs with the development of sophisticated retrieval systems [Will87a], of quantitative structure-activity relationships, and of synthesis design techniques [Hend86] inter alia. This chapter briefly reviews several of these areas as a precursor to the more detailed description of techniques for processing three dimensional (3D) chemical structure data that forms the basis of this thesis.

1.1 STRUCTURE REPRESENTATION

In addition to the great demand for information systems in the pharmaceutical industry, information systems in chemistry have a large advantage over other scientific fields in that chemical structures provide a very convenient index to the data. A variety of different structure representations have been developed for chemical compounds partly for historical reasons, and partly for use in specific applications, for instance structure-activity oriented representations [Avid82]. The three main unambiguous structure representations (that is where each characterization defines a single compound) [Ash85, Will87a] are described very briefly below.

1.1.1 Systematic Nomenclature

Systematic nomenclature methods [Cahn79] are algorithmic ways of assigning names which are of the familiar chemical style to chemical compounds (for example, calling $SnCl4$ tin (4) chloride). However, computers cannot easily manipulate data in this form, and so it has very little use in chemical retrieval systems other than for printed indexes. When it does occur it is usually for historical reasons and software is used to convert the systematic name into another structure representation [Vand74].

1.1.2 Linear Notations

Linear notations represent chemicals as a string of alphanumeric characters and the main linear notation is the Wiswesser Line-formula (WLN) [Voll83]. In WLN, commonly occurring features such as certain rings, are represented by only a few characters (eg. the hydroxyl pair O-H is represented by Q) and saturated, branchless carbon chains are denoted by the number of carbons in the chain. Examples of the naming of structures using WLN are given in figure 1.1 (along with their intermediate steps).

The use of linear notations in present day chemical information systems owes a great deal to history in that before the general use of computers, linear notations had the advantages of being relatively easy to produce from a chemical structure diagram, not requiring much storage and being easy to sort (to allow indexing). These advantages were also important when computers started to be used in chemical information, but more recently the increased power of computers has led to widespread use of connection tables.

1.1.3 Connection Tables

Connection tables [Ash75], as their name suggests, indicate which atoms are connected together and the order of each bond. Normally, hydrogen atoms are not included in connection tables since their presence can be deduced from a knowledge of the connectivities, ie. the number of attached non-hydrogen atoms, for each of the atoms in the table. An example of a connection table is given in figure 1.2.

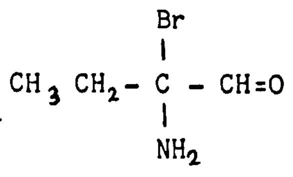
The connection table of figure 1.2 includes some redundancy in it in that each bond occurs in two places in the table. This can be removed if storage space is short, however this would have an adverse effect on the speed of obtaining information from the table. Connection tables can

$CH_3\ CH_2\ CH_2 - NH - CH_2\ CH_2\ CH_3$

```
Intermediate Step  3      M       3
```

Final WLN Form 3M3

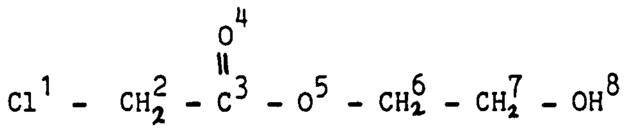
Figure 1.1(a) A Symmetric Example Of Wiswesser Line Notation



```
                          E
Intermediate Step    2    X    VH
                          Z
```

Final WLN Form ZXE2&VH

Figure 1.1(b) A More Complex Example Of Wiswesser-Line Notation



	Connection	Bond	Connection	Bond	Connection	Bond
1 Cl	2	1				
2 C	1	1	3	1		
3 C	2	1	4	2	5	1
4 O	3	2				
5 O	3	1	6	1		
6 C	5	1	7	1		
7 C	6	1	8	1		
8 O	7	1				

Figure 1.2 A Structure Diagram And Its Corresponding Connection Table

be made unique by using algorithms which specify the order in which the atoms should occur in the table. Probably the best known ordering algorithm is that described in [Morg65] which assigns a number to an atom depending on its connectivity, the sum of the connectivities of its neighbours, and so on.

The chemical structures are not usually input directly as connection tables but are more likely to be input as two dimensional structure diagrams or in another representation such as WLN; conversion software is then used to produce the internal connection table representation.

As was alluded to above, over the last 15 to 20 years connection tables have become the more attractive means for structure representation in chemical information systems because they allow more flexibility in substructure searching as described in the next section [Bawd83, Ash85, Will87a]. (The user can choose to search for any fragment he wants whereas WLN systems often place restrictions on the type of fragments which can be searched for.) On account of this, connection tables are the only type of structural representation to be considered in the rest of this thesis.

1.2 CHEMICAL RETRIEVAL SYSTEMS

All major chemical information systems include powerful searching facilities with the two main types of searches [Almo82] being:-

1) Structure searching where a database is analysed to see whether it contains a molecule which is the same as the query molecule (and it is used for the gathering of general information on the molecule).

2) Substructure searching where molecules are looked for which contain the query as a substructure. This type of searching is used, amongst other reasons, for the retrieval of chemicals containing substructures thought to cause a particular activity, the analysis of rival companies products, obtaining information about reactions and planning possible synthesis paths.

Recently there has been interest in generic structure searching [Lync81] where classes of related compounds are retrieved which differ in having, for example, variant substituent groups and patterns. This is of use in patent work where whole classes of related compounds need to be retrieved.

Before considering the two main types of searches in detail, the closely related mathematical field of graph theory [Deo74] is introduced. In addition to its close relationship with chemical searching and connection tables, several of the algorithms used in future chapters will have their foundations in graph theory. Therefore the next section gives a brief introduction to graphs and the following section introduces the related idea of NP-completeness (which is of importance when discussing the complexity of graph theory algorithms).

1.2.1 Graph Theory

The representation of a chemical structure as a connection table is closely related to the concept of a graph with the problems of structure and substructure searching becoming those of graph and subgraph isomorphism. Graph matching is also widely used in pattern recognition for tasks such as identifying a machine part [Boll79] and finger print identification [Isen86].

1.2.1.1 Introductory Graph Theory

A graph consists of a set of points (nodes) and a set of arcs between pairs of these points. If each node has a descriptor associated with it, then the graph is labelled and if a descriptor is associated with each arc, then the graph is weighted. A directed graph is when each arc has a direction associated with it. A subgraph of a graph is a subset of the nodes of a graph along with all the arcs of the graph connecting these nodes. (However, some definitions of a subgraph allow a subset of these arcs.) Therefore a connection table can be regarded as a labelled, undirected graph where the labels on the nodes are the atomic numbers and the labels on the arcs are the bond orders.

A graph isomorphism [Read77, Gati79] is a one to one mapping between the arcs and nodes of one graph and the arcs and nodes of another graph such that if an arc with end points a, b maps onto an arc with end points x, y then {a,b} maps onto {x,y}. A subgraph isomorphism is a graph isomorphism between a graph and a subgraph of another graph. Consequently, the problem of chemical structure searching (exact matching) for purposes such as registration is analogous to that of graph isomorphism, and that of substructure searching (partial matching) is analogous to subgraph isomorphism. Unfortunately, whereas quick methods for structure searching exist (see Section 1.2.2) and it can be shown that graph isomorphism problems can be solved in a time which is a polynomial function of the size of the graph for graphs of bounded valence [Luks80], subgraph isomorphism is known to belong to the NP-complete class of problems.

1.2.1.2 NP-Completeness

The concept of NP-completeness [Gare79, Papa82] is concerned with the performance of algorithms in the worst possible case. More specifically, the label NP-complete identifies a class of problems for which no known algorithm exists which can solve all cases of any of the problems in a time which is a polynomial function of the "size" of the problem, but for which no-one has proved that such an algorithm does not exist. (The "size" is a polynomial function of such things as the number of nodes in the graphs.) Moreover, the NP-complete class is defined so that if a polynomial algorithm is found for one member of the class, then the algorithm can be modified to solve any other member of the class in polynomial time. Hence, in one sense, all NP-complete problems are of the same degree of complexity.

The most well known NP-complete problem is that of the Travelling Salesman Problem [Lawl85] which involves finding a path between a set of cities which is of the shortest possible length, which visits each city once and which ends at the starting point, and the problem has been extensively studied over many years. However, the "intractability" of these problems only occurs in a very low percentage of cases, and so the problem really becomes that of finding an effective heuristic for the cases of interest. But the NP-complete concept does indicate that the chosen algorithm could have a poor performance in adverse conditions.

1.2.2 Structure Searching

With structure searching [Will87a], if the structure representation being used is a unique form such as WLN, then the problem becomes the straightforward one of string matching. However, if connection tables are being used they can either be put into a unique form using Morgan's algorithm or some form of hash coding can be applied. With this last method, the few hits from the hashing stage can be passed on to the computationally more expensive isomorphism examination [Bawd81]. A system which incorporates stereochemical features, such as chiral centres in the matching process has also been described [Wipk74].

1.2.3 Substructure Searching

Another important method in computer assisted drug design is chemical substructure searching (determining whether a pattern of atoms is present in a chemical compound) [Will87a]. This can be either two dimensional searching which uses the connectivity relations (or topology) of the atoms, or three dimensional which uses their 3D co-ordinates (or topography). Three dimensional substructure searching is more directly related to drug design because the binding of a drug to its target is very heavily dependent on the drug's three dimensional shape, and it will be considered in detail in Chapter 3.

Due to the computational expense of examining a molecular structure to see whether it contains a particular substructure, most substructure search systems have two stages. The first (or screening) stage [Lync75] uses a computationally inexpensive method to try to rule out from consideration most of those structures in a machine readable file which do not contain the substructure. The structures which survive stage one, then pass on to an atom-by-atom search to discover whether they actually contain the required pattern. This involves atoms from the query and the structure being compared with each other to determine whether they have the same attributes (such as having neighbours of the same atomic types).

Whilst screening systems have traditionally been used for carrying out substructure searches on large files of compounds, other techniques are now being developed although their exact details are not clear. In particular, [Bruc87] describes a method for entering the molecules in a database into a tree structure by examining each atom, its neighbours and so on, and using these to determine which branch to take when building the tree structure. Similarly, a query substructure descends down the tree until it comes to its match and the molecules containing it can be read out. The advantage of this method is that as the number of molecules in the file increases, the search time goes up sub-linearly. Another alternative has been described by Vladutz [Vlad87] and involves superimposing all the molecules in a file on to a grid. A substructure search is effected by finding the grid sites corresponding to it and intersecting these with those assigned to each compound. However, neither of these techniques will be considered any further in this thesis.

1.2.3.1 Screening

When the search system is set up, the relevant screens are assigned to every structure in the database. Then each substructure search involves determining the screens "contained" in the substructure and extracting the structures which contain all of these screens in their own screen lists. These structures are then passed on to the atom-by-atom search, with one of the primary aims of the screening system being to minimise the number of incorrect structures (or false drops) passed on to this step.

Ideally, each screen should be assigned to 50% of the structures while being independent of all of the other screens [Adam73a]. This cannot be achieved in practice due to the uneven distribution of substructural features [Adam71] and the interdependence of screens; while another complicating factor is that the queries might not make an even use of the screen set [Adam73c]. However, it leads to the important principle that the screens should have as even a distribution as possible.

In practice, how specific the screens should be depends on how homogeneous the chemical structures are and the probable nature of the queries. There are two main methods of screen generation (although in reality some combination of the two is often used). The first method was developed at Sheffield University in the early 1970's [Adam71, Adam73a, Adam73b] and uses bond centered fragments (although atom and ring centered fragments were also investigated). The screens are assigned with approximately similar frequencies by having the commoner features associated with larger bond centered fragments.

The second method [Feld75] was developed at the Walter Reed Army Institute of Research (WRAIR). This technique produces screens by "growing" atom/bond fragments in a tree, each branch having a screen associated with it. A branch is "pruned" back one stage when the relevant screen has an assignment frequency below a specified level. Additionally, Feldman and Hodes [Feld79] have described a way of arranging the WRAIR screening system to deal with queries of a varying degree of specificity.

An example of screen development in practice is the design of the screens for the CAS ONLINE system and this is described in [Ditt83].

1.2.3.2 Atom-By-Atom Searching

When the structure representation is a systematic nomenclature or a linear notation, the atom-by-atom search is similar to text searches. However, when a connection table is used, the problem becomes that of subgraph isomorphism. In the chemical field several algorithms have been described for this task [Suss65, Figu72, Kitc82, VonS84] and they all iteratively refine a mapping between the structure and the substructure by considering whether each atom in the structure which matches a particular pattern atom, has neighbours which match the pattern atom's neighbours. Further details of subgraph isomorphism algorithms in the chemical field can be found in Chapter 3 whilst the relaxation class of algorithms [Pric85] which includes [Kitc82, VonS84] is outlined in Section 5.3.1.

1.2.3.3 Two Dimensional Substructure Search Systems

Two dimensional (2D) substructure searching is now a major feature of all large chemical information systems and probably the most important searching system is CAS ONLINE which is reviewed in [Ditt83, Stob85]. One of the main features of this system is that it uses pairs of microcomputers, which are connected to a larger computer via a network, to do the search. Each pair is allocated a section of the database to search and one of the pair does the screening stage while the other does the atom-by-atom search. Consequently, many of the microcomputers can be active at the same time ("parallel computation"), leading to the substructure search being completed quicker.

As CAS ONLINE is not available as an in-house system, other important search systems such as the MACCS software package [Polt82], have been developed. With all these major search systems, sophisticated input facilities are available for drawing structure diagrams of query substructures on v.d.u. terminals.

To try to partially overcome the problem that a drug's activity is dependent on its three dimensional shape, another system [Elde84] allows three dimensional features, such as atom-plane distance, to be added to the substructure being searched for. The search first uses the connectivity relations and, if a possible match is found, it then goes on to the three dimensional constraints.

1.3 QUANTITATIVE STRUCTURE-ACTIVITY RELATIONSHIPS

In the past, the molecules passed on to the synthesis and testing stage of the drug design process were selected fairly randomly and if the tests showed that they were active, structurally similar molecules were also synthesised so as to try to optimise the activity [Fran84]. However, as was mentioned at the start of this chapter, this method is no longer adequate with the chance of finding a new agent being estimated at one in ten thousand and the cost at more than 40 million dollars. Therefore there has been an increasing use of computers to select which compounds it is thought worthwhile to synthesise and the major technique for doing this is quantitative structure-activity relationship (QSAR) methods [Mart81, Topl83, Hopf85]. These aim to correlate the structural properties of the compounds under investigation in a quantitative manner with the compounds' respective biological properties. This is achieved by having a set of chemicals whose structural properties and activities are already known, to calibrate the methods. (The structural properties which are used in QSAR's include physicochemical properties, spectral characteristics and two or three dimensional substructural features [Bawd83]. While the activities can be classified as being either quantitative that is where a measurement is taken and so the range of values is continuous, or qualitative where the range of values is discrete such as active or inactive [Will87b]).

The estimates of biological activity produced by QSAR techniques, are used to reduce the number of compounds needing to be synthesised and tested. There are two ways of doing this:-

a) lead generation -where compounds are searched for which have the required biological activity but are from different structural classes from the structures currently under investigation.

b) lead optimisation -where a potential new drug has been found (maybe from lead generation) and small structural modifications are made to it so as to increase its activity.

There are three main classes of QSAR methods [Ash85] :- Hansch Analysis, the Free-Wilson Method and Pattern Recognition.

1.3.1 Hansch Analysis

This uses physicochemical properties such as hydrophobic and electronic components for the structural features. Normally, the equation used is

activity = k1 + k2*pi + k3*sigma + k4*Es + k5*MR

where [Mart81] pi is the hydrophobic component (the effect on the logarithm of the octanol-water partition coefficient), sigma is the electronic component (the logarithm of the effect on the acid dissociation constant of benzoic acid), Es is the steric component (the relative rates of hydrolysis of esters) and MR is the molar refractivity (derived from the refractive index) providing the dispersion contribution.

The parameter values k1,..,k5 are obtained using multiple regression on the set of compounds whose activity is already known. In some cases this approach has proved unsatisfactory and more complex (non-linear) equations have been used [Hans73, Mart81].

Hansch analysis has been widely applied and has had a fair degree of success [Ash85] with it usually being used for lead optimisation rather than lead generation. Because of this interest in Hansch analysis, there has been a corresponding interest in how to calculate the physicochemical components (for example [Iwas85]) and this need for knowledge of the physicochemical properties is one of the main drawbacks with Hansch analysis [Crai75].

1.3.2 The Free-Wilson Method

The Free-Wilson method is based upon whether certain groups are present or absent from specified ring substituent positions in the compound and is described by the equation

$$\text{activity} = K_1 + \sum_{i,j} K_{ij} * X_{ij}$$

where X_{ij} is an indicator variable which takes the value 1 if the ith group is present at the jth position but which is otherwise 0, K_{ij} is the contribution to the activity of the ith group being at the jth position and K_1 is a constant (being the mean of the activities in the data set).

One of the drawbacks with this method is the large number of compounds whose activity must be known in order to be able to derive the values of K_{ij} from a statistical analysis. Other disadvantages are the need for multiple substituent positions and the fact that all molecules must belong to the same structural class [Crai75]. Also all the compounds being considered need to be similar, and so the method is unsuitable for lead generation.

[Ash85] reports that Free-Wilson analysis has not had that many successful applications cited in the literature (maybe because the added computational complexity makes it less attractive than Hansch analysis), but that it has exported the idea of indicator variables into Hansch analysis. Additionally, it has led to the important technique of substructural analysis [Ash85, Cram74, Almo82, Adam74, Adam77, Hode77, Hode81] in which the activity of a compound is regarded as being correlated to the substructural features it contains but with no account being taken of where these features occur. For reasons of expediency, the fragments used have often been drawn from existing retrieval systems, and so substructural analysis has been a very appropriate method of QSAR for use within computerised chemical information systems (where it can be used with collections of thousands of molecules). However, [Adam77] describes a more advanced system which is basically Free-Wilson analysis without the substituent positions.

Some of the factors which influence the decision whether to use Hansch or Free-Wilson analysis are [Crai75]:-

1) If there are only one or two substituent positions, then Free-Wilson analysis will probably yield nothing more than a chemist's intuition is likely to produce, and so Hansch analysis should be preferred.

2) Hansch analysis can deal with more disparate structural classes. However, both methods have problems trying to extrapolate to less similar compounds.

3) If both methods are applicable, then [Crai75] suggests that Free-Wilson analysis should probably be used first.

1.3.3 Pattern Recognition

The third major class of QSAR methods is pattern recognition techniques [Redl74] and these are useful for dealing with qualitative property data where parametric statistical methods cannot be applied (and they are well suited to deal with cases where there are discontinuous relationships). An example of their use has been in problems where the number of substructural fragments was very large in order to reduce the this number to a total where Free-Wilson analysis could be used safely. However, [Wold83] criticises many applications of this idea as having an unsound statistical basis (see below).

Pattern recognition methods [Stup79] are made up of three stages:-

1) A group of compounds whose activities have been tested, are analysed and a set of structural attributes is extracted which can be used to discriminate between the activity classes.

2) A methodology is developed for assigning a new compound to one or more of the activity classes present in the original group of compounds, on the basis of the new compound's structural attributes. The classification method is usually based on either splitting the data up by using hyper-planes which mark the boundaries of the data classes or assigning a new data element to an activity class by considering which activity class its nearest neighbours belong to.

3) The appropriate compounds whose activities are unknown, are assigned to the relevant activity classes by the decision making process of step (2).

The structural attributes that can be used in pattern recognition (which range from single atoms to 3D patterns of atoms) are discussed in [Bawd83] along with some of the criticisms of the results obtained using pattern recognition [Matt75, Wold83]. These criticisms are centered on when it is justified to extract features from a data set and then to assign new data elements to classes on the basis of a statistical analysis of these features. An example of one of the problems is how large the ratio of the number of elements in the data set divided by the number of extracted features should be. However, it has been successful [Ash85] and Jurs and Stuper [Stup76] give details of a software package implementing it.

1.3.4 An Overview Of QSAR

If QSAR methods are used, then they are only one step along the path of developing a drug; QSAR analysis aims to give some indication of the biological activity of a compound without having to synthesise it. If the predicted activity is low, then the expense of synthesising and testing the chemical is probably not worthwhile. However, QSAR methods cannot help in avoiding the pitfalls which occur at the clinical stage, and so the value of QSAR analysis should be judged by the number of compounds derived from QSAR techniques which reach the development stage. Hopfinger [Hopf85] reports that at Searle the required activity level is achieved from 42% of the drugs developed with computer assistance, which is several times the figure which would be obtained by chance. Furthermore, in the last few years examples of drugs designed by QSAR methods have begun to emerge [Hans84] (with the two quoted examples arising from Hansch analysis being applied to a series of molecules obtained by methodically varying the substituents at each position).

One of the main drawbacks with QSAR methods is that a molecule's three dimensional shape can be very important in determining its biological activity. Consequently, there has recently been interest in using molecular shape indices as an extra parameter in Hansch analysis [Hopf80, Walt84, Kier85] (although there is some implicit 3D information in the steric component [Mart81]). Additionally, the splitting up of QSAR methods into the above three classes slightly over simplifies the issue as in practice hybrids are quite likely to be used [Mart81] (eg. the use of position dependent terms in Hansch analysis).

[Wold83] has criticised many (about 50%) of the papers on QSAR which were examined in a study, for using incorrect statistical techniques which invalidated the results. However, this is less of a problem now as people are more aware of the dangers. Also the criticisms are not likely to prove a disincentive to using QSAR's because of the huge number of compounds which need to be examined. For example, [Fran84] discusses a molecule with a varying number of ring substitution positions and (only) 50 substituents, and so the total number of compounds needing to be examined is 50 to the power of the number of ring positions.

1.4 THE NEED FOR THREE DIMENSIONAL METHODS

The very fast rate of increase of interest in QSAR techniques has now slowed down [Aust84, Hopf85, Cohe79] due to the fact that the amount of success that they have achieved has not fulfilled the very high expectations for them, a particular problem being in predicting compounds from other chemical families which are likely to be biologically active. This is partly due to the geometric nature of drugs' interactions with their hosts, and so it has led to a rapid increase in the use of three dimensional computer graphics in drug design.

Another very important factor in the rise of molecular graphics is that computers are now powerful enough to allow real time modification of three dimensional molecules on the screen. Thus, because of technological developments, the interest in chemical representation in information systems has moved from printed indexes and WLN, through connection tables, and graphics based 2D systems to 3D co-ordinates.

A short introduction to these graphics based techniques is given in the next chapter along with a description of other methods for analysing the 3D nature of the drug-receptor binding. After which, the rest of the thesis considers the problem of developing efficient algorithms for dealing with 3D co-ordinate data. In more detail, Chapter 3 describes a screening system for 3D substructure searching before comparing several algorithms for the partial matching stage. Chapter 4 examines two algorithms (along with various extensions) for finding the 3D substructures in common between molecules. Following on from this, the next chapter reports the results of a simulation of a parallel processor executing one of these algorithms and Chapter 6 is concerned with an actual parallel processor implementation of the algorithm. Chapter 7 describes the use of such a processor on one of the drug-receptor examining algorithms of Chapter 2, and Chapter 8 describes a system for searching the Cambridge Crystallographic Database for patterns similar to the provided one using the screening system of Chapter 3 and one of the algorithms of Chapter 4. As further background material for all of this work, [Cohe85] gives an extensive review of the use of 3D information in drug design.

CHAPTER 2
COMPUTERS IN THREE DIMENSIONAL DRUG DESIGN

The last chapter gave a brief, introductory review of some of the two dimensional methods used in computer assisted drug design. The current chapter describes the (three dimensional) binding of a drug to its receptor before summarising the main sources of availability of a molecule's co-ordinates, and looking at two computer based methods for investigating the drug-receptor interaction.

2.1 THE DRUG-RECEPTOR INTERACTION

The recognition of a drug by its receptor and their subsequent interaction is dependent on the three dimensional geometry of the two molecules, as can be seen from the fact that different stereoisomers of the same molecule may or may not have any effect [DeRa84]. The process is generally regarded as being similar to that of a key fitting a lock [Gund79] with the forces which cause the attraction and subsequent binding to occur being, in order of decreasing energy, electrostatic, hydrophobic and van der Waal's [Gund77, Koll84]. The attraction is a two way process with the receptor being attracted to the ligand as well as the ligand to the receptor. The pattern of the drug's atoms which are attracted to the receptor, is called a pharmacophore [Toll84].

However, this somewhat over simplifies the situation as it is thought that the receptor and the ligand undergo conformational changes during the binding [Gund79]. Burgen et al. [Burg75] have proposed a model of the interaction where only parts of the pharmacophore initially bind to the receptor and the ligand then assumes a different conformation before the rest of the molecule attaches itself to the ligand. (Even in this case though, the interaction will occur very quickly). [Will77] whilst discussing in detail the dynamic nature of the interaction, points out that "static matching has a very important and proven role to play".

When a drug binds to a receptor and produces a normal biological response, it is called an agonist. However, a drug may adhere to a receptor in a way that prevents agonists binding to the receptor, and in this case, the drug is called an antagonist [Gund77]. A molecule may have the right pharmacophoric pattern of atoms but still not produce the right effect because of, amongst other factors, transport problems in arriving at the receptor and having other atoms which are in positions which prevent binding taking place [Gund77, Gund80]. On the other hand, the pharmacophoric pattern may allow some of the atoms to be in a range of positions or for some of the positions to be occupied by atoms from a choice of types [Gund77].

2.2 THE AVAILABILITY OF 3D CO-ORDINATES

One of the prime factors behind the increase in the use of 3D methods in drug design has been the more general availability of molecules' 3D co-ordinates. Before describing computer methods for examining drug-receptor binding, the sources of these co-ordinates (which form the input for these methods) will be considered.

2.2.1 Obtaining 3D Co-ordinates

The main experimental method of obtaining the 3D co-ordinates of molecules is X-ray crystal structure analysis [Duch79]. This obtains the atomic co-ordinates from an analysis of the X-ray diffraction patterns of various orientations of the crystal. The regular structure of single crystals acts like a diffraction grating, thus providing information on the spacing of atoms. The increasing power of computers has meant that the analysis step has become less of an obstacle, and there has been an increase in interest in polycrystalline materials and protein crystallography [Town85]. However, the main drawbacks of X-ray crystallography are that it only determines the co-ordinates for the conformation the molecule adopts in its solid state, and that the co-ordinates of hydrogen atoms are difficult to determine accurately. The problem with the conformation which is to a greater or lesser extent a difficulty with all the techniques for obtaining co-ordinates, will be discussed in Section 2.2.3.

The other major experimental technique is Nuclear Magnetic Resonance (NMR) spectroscopy [Jame75] which determines the atomic positions for the liquid conformations of molecules. This method works by putting the compound in a varying magnetic field and measuring the frequencies that the nuclei resonate at by way of the photons which are emitted. The main drawback of NMR spectroscopy is that it is difficult to provide the co-ordinates with enough precision.

Although quantum mechanics can provide molecular co-ordinates, its high computational cost means that molecular mechanics [Duch79, Boyd82] is the main calculational technique for deriving the atomic positions. It works by trying to minimise the strain energy of the molecule and differs from quantum mechanics primarily in that the electrons are not considered as a separate entity in the calculations. By finding local minima, the co-ordinates for the different conformations of the molecule are obtained. The initial co-ordinates that the molecular mechanics technique works on, can be obtained by using the standard bond lengths and angles or by using a simpler optimising method such as distance geometry (which is described in Section 2.3.2).

2.2.2 Databases Of Co-ordinates

Various databases [Murr84] of molecular co-ordinates exist of which the most important are the Cambridge Crystallographic Database [Alle79] and the Brookhaven Protein Databank [Bern77, Abol85]. The Cambridge system provides the crystal co-ordinates of about 40,000 substances along with their connection tables and references to relevant papers. Various searching facilities are available for the connection table and bibliographic files, with the latter being able to be searched for key words. The structural data which is retrieved can be displayed using a molecular plotting program. The Brookhaven database contains the co-ordinates of more than 300 macromolecules along with literature citations and details of secondary structures.

The data used in this thesis will be of the form given in figure 2.1 which just gives the number of atoms, their atomic numbers and their co-ordinates in units of Angstroms (and in some instances the 6 letter reference code used to identify the molecule in the Cambridge Database). Additionally, the connectivities were also used when screens were being assigned.

2.2.3 Relevance To The Ligand's Conformation

The above methods and databases for providing molecular co-ordinates can be criticised when they are used

```
PHYLOC
           16
            6    3.55198765          3.53722382          5.30371952
            6    4.55479240          2.71890545          4.69650745
            6    4.19013596          2.51220417          3.41784763
            6    4.90155602          1.88468552          2.31977463
            6    4.26426029          1.99492455          1.11868191
            6    1.77641964          2.12530136          2.82881069
            6    1.85309982         0.770622551          3.50753021
            6   0.794064224        -0.204578936          3.00818348
            6   0.581916094        -0.635992289E-01     1.52469444
            6    1.70655537         0.654022157         0.859311163
            6    3.01352310          2.77190304         0.925975025
            6    2.96155167          3.96864319          1.88587952
            6    2.89679909          3.14184284          3.15484238
            7    1.85309982          1.99810123          1.36471558
            8    2.58667183          3.84462261          4.36563110
            8    3.43015194          3.96864414          6.40906429
```

Figure 2.1 The Usual Form Of The Co-ordinate Data Used In This Thesis

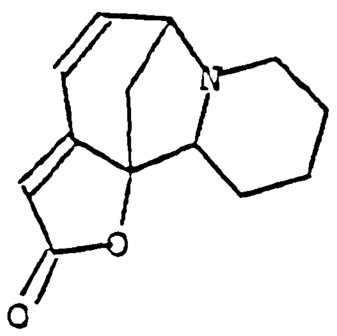


Figure 2.2 The Structure Diagram For The Molecule Of Figure 2.1

to study the receptor-ligand interaction because there is no reason to believe that the conformation adopted by the ligand is that of its least energy. This arises from the presence of the (usually much larger) receptor which provides distorting forces [Toll84, Mars79, Humb80, Mars84]. Marshall [Mars84] suggests that all the conformations within a certain energy range from the conformation of least energy should be considered, but there may well be many such conformations.

Gund [Gund77, Gund79] argues that there is likely to be some attraction, if only a weak one, between one of the major conformations of the drug and its receptor. He also suggests that any interaction which takes place will happen quicker if it occurs when the drug is in its ground state conformation as there will be a higher concentration of molecules in this state. However, consideration of the various low energy conformations, or the use of conformationally restricted analogues [Horn84] which try to imitate the original drug but which have less conformational flexibility, is still required.

Recently, the technique of radioligand binding [Gour84] which involves using a radioactive ligand, has provided an additional method for investigating the receptor-ligand binding.

2.3 COMPUTER EXAMINATION OF THE RECEPTOR-LIGAND INTERACTION

This section discusses two methods for analysing the receptor-ligand interaction which try to overcome the conformational flexibility mentioned above. In the first (which is the much more widely used and important of the two), the two sets of co-ordinates are displayed on a graphics terminal and the user (a skilled chemist) manipulates the two molecules into a docking position. Whilst in the second method, the upper and lower bounds on the inter-atomic distances of each molecule are compared so as to try to find a common region (the pharmacophore). However, the two methods should not be regarded as alternatives but rather as two elements in the computer assisted drug design field, some of whose other members were described in Chapter 1 (and another one of which will be described in the next chapter).

2.3.1 Three Dimensional Computer Graphics

The increasing availability of molecular co-ordinates coupled with the decreasing cost and greater power of computer graphics hardware, has led to interactive computer graphics playing a very important role in drug design [Vint85, Hass85]. Instead of using the traditional wire frame models, molecules can now be built and displayed on graphics terminals [Toll84, Will77, Kao85, Humb81]. The 3D shape of the molecule can be examined by rotating it or by using depth cueing (in which points at a greater distance from the user have a reduced intensity). Sections of the molecule can also be examined more closely by zooming in on portions of it. The image displayed may take other forms rather than the traditional stick diagram, for instance the electron densities can be displayed.

Besides simply viewing a molecule, two molecules can be superimposed so as to examine their degree of similarity. Alternatively, by rotating parts of the molecule about various bonds, different conformations can be produced and examined. Software can provide an indication of the energy level of each of the new positions. Dynamic docking of molecules can be simulated by bringing the molecules closer together and then examining the various conformations which they can take on [Buse83], thus overcoming the criticisms of rigid pharmacophores met in Sections 2.1 and 2.2.3.

The use of computer graphics techniques for the examination of binding has been widely reported. This use can either be independent of other computer assisted drug design techniques, for instance [Feld78, Palm83], or in conjunction with them. The main example of this latter case being the combining of Hansch analysis, X-ray crystallography and computer graphics [Hans82, Caro84]. Here computer graphics aids in the understanding of how the steric and hydrophobic coefficients are acting and thus enables better values to be calculated for these coefficients.

At present, because of the large numbers of atoms which can need to be rotated, most molecular graphics systems use vector-refresh (or line drawing) displays. However, in the future, the decreasing cost and increasing speed of raster graphics systems should lead to increasingly more detailed molecular images being able to be manipulated interactively by the user [Lang81].

2.3.2 Distance Geometry

Distance geometry [Crip81, Have83] is a technique for finding a set of possible atomic co-ordinates given the maximum and minimum bounds on every inter-atomic distance. As the method employs random numbers to choose "trial" distances from the allowed ranges, a search of conformation space can be carried out by repeatedly applying the method. A full description of the basic algorithm can be found in Chapter 7, the present section is only concerned with possible applications to receptor-ligand analysis.

Distance geometry was originally developed as a means of determining macromolecular conformation [Crip79b, Have79] but it has been used to generate a series of possible conformations of the ligand whose elements are then compared with the receptor's binding site so as to try to find a match [Crip79a, Crip80]. The algorithm used for this comparison [Levi72, Barr76, Kuhl84] will be considered in more detail in Chapters 3 and 4 where it is used for finding the maximum substructure in common between two or more molecules.

The use of only one upper and one lower bound matrix is inefficient as a large number of inter-atomic distances will be correlated and consequently, [Ghos85] has suggested a way of searching conformation space by discrete rotations about bonds. Upper and lower bound matrices are produced from the co-ordinates of the atoms before and after each rotation, and the method avoids missing conformations through having too large an angle of rotation.

An alternative use of distance geometry [Sher86] tries to find pharmacophores by combining the upper and lower bound matrices of several ligands, with the distances between atoms in different ligands but which are thought to correspond to the same atom in the pharmacophore, being set to zero. If suitable co-ordinates can be found which satisfy the combined bound matrices, then these give a possible pharmacophoric pattern.

Whilst they do not use distance geometry, it seems appropriate because of their similarities to the above methods to mention several algorithms developed by Motoc et al.. [Moto86] describes a search of conformation space using increments of rotation angles; the search incorporates a quick check to see whether van der Waal's radii are infringed. Pharmacophores can be looked for by picking functional groups from a set of molecules with a pharmacophore existing if the intersection (over the set of molecules) of the distance ranges between the functional groups, is non-empty. The intersection ranges from molecules considered first can be used to constrain the conformational search of the later molecules. [Laba86] details a molecular mechanics program where specified geometric relationships can be maintained. This allows flexible molecules to be compared with a rigid pattern or for specified atoms from two molecules to be correlated and the possible conformations examined.

2.4 DISCUSSION

This chapter has considered the increasing importance of 3D co-ordinate methods, especially computer graphics, in computer assisted drug design. However, the two methods reviewed in Section 2.3 are computationally expensive and can only deal with small numbers of molecules. Therefore the next chapter describes a system for searching the Cambridge Crystallographic Database for user specified pharmacophoric patterns (with the retrieved compounds then being passed on for more detailed analysis to the above methods). This use of a cruder method to screen large collections of molecules is somewhat analogous to the use of substructural analysis as opposed to Hansch or pattern analysis when working with 2D data (see Chapter 1).

Before moving on to describe the work carried out for this dissertation, it is perhaps best to summarise the computer-assisted drug design tools which have been discussed above. QSAR techniques are useful when dealing with large numbers of similar structures typically formed using different ring substituents when trying to optimise the activity of a drug, but they are less helpful in generating new "lead" compounds. The 3D graphics methods can provide valuable insights into inter-molecular binding, and so can indicate which atoms are the active ones in a drug. However, they can only deal with a handful of molecules at a time and they require a large amount of interaction from the user. The non-graphics approaches described in this chapter are much more recent and less widely used but are of use in the same sort of context as the computer graphics approach. In any pharmaceutical design setting, all of the above methods are likely to be available (with the possible exception of the distance geometry related approaches) and used at different points in the design process. An example of such an integrated system is described in [Klei86].

CHAPTER 3

THREE DIMENSIONAL SUBSTRUCTURE SEARCHING

As was mentioned in Chapter 2, the increased availability of molecules' 3D co-ordinates along with increased computer power and the 3D nature of drug receptor interactions has led to widespread use of computer graphics systems for docking molecules. As this involves the study of pharmacophoric patterns, interest has also been shown in determining whether a particular pharmacophore is present in a molecule. [Gund77] has described a system for searching a given molecule for a specified pharmacophore and this program has been extended by Esaki [Esak82, Esak83] to allow a comparison of electronic states. Work has been carried out at Sheffield by Jakes to allow the Cambridge Crystallographic Database of molecular co-ordinates to be searched for user specified pharmacophoric patterns. Like the 2D substructure searches described in Chapter 1, this system is composed of a screening stage followed by a more computationally expensive (per molecule) partial matching stage for compounds which pass the first stage. A short description of the screening stage is given (a fuller one can be found in [Jake87b]) before a comparison of several partial matching algorithms is reported.

3.1 INTER-ATOMIC DISTANCE SCREENS

3.1.1 Basic Implementation

The 3D screening system has a similar role to a 2D screening system, however, whereas in the 2D case there are many possible features on which a screen set may be based, in the 3D case there is an obvious candidate in the distances between pairs of atoms (although torsion angles could also be considered). Therefore the screening system was based on distances between the atoms, and only atoms which were of types B, Br, C, Cl, F, I, N, O, P or S were used in the atom pairs as usually only these occur in pharmacophoric patterns [Watt84]. Following an analysis of the numbers of each type of atom pair present in the database, Jakes decided to split each atom pair distance range (that is the frequency distribution of the distances between each possible pair of atomic types -figure 3.1 shows the distance versus frequency graph for the carbon-oxygen pair) into blocks containing approximately 1000 occurrences of the atom pair and these blocks then made up the screen set. Additional screens were assigned for use when the type of one of the original atoms in the atom pair is not specified.

Connectivity information was also incorporated into the description of the atoms but it is not considered in this chapter as it is essentially a topological factor.

Molecules which are retrieved from the database by the screening system are then subjected to a test to

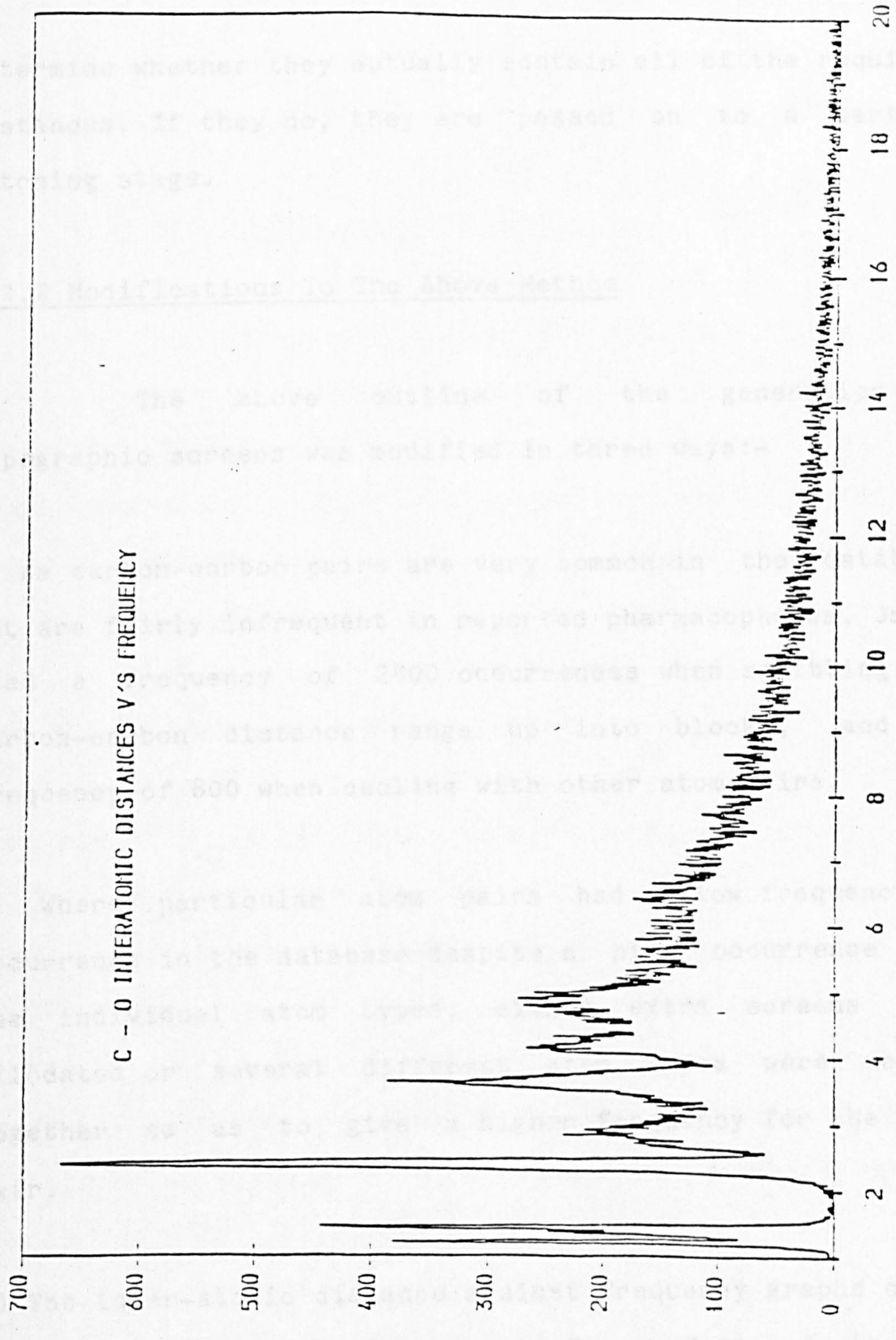


Figure 3.1 The Frequency Of Occurrence Of The Oxygen-Carbon Pair When Plotted Against Distance For A Sample Of The Cambridge Crystallographic Database (Taken From [Jake86])

determine whether they actually contain all of the required distances. If they do, they are passed on to a partial matching stage.

3.1.2 Modifications To The Above Method

The above outline of the generation of topographic screens was modified in three ways:-

1) As carbon-carbon pairs are very common in the database but are fairly infrequent in reported pharmacophores, Jakes used a frequency of 2400 occurrences when splitting the carbon-carbon distance range up into blocks, and a frequency of 800 when dealing with other atom pairs.

2) Where particular atom pairs had a low frequency of occurrence in the database despite a high occurrence for the individual atom types, either extra screens were allocated or several different atom types were merged together so as to give a higher frequency for the atom pair.

3) The inter-atomic distance against frequency graphs often show peaks (an example is shown in figure 3.1) and it was felt undesirable to have different screens allocated for different parts of a peak. Therefore a threshold value was introduced and a screen's distance range could only end at a point on the atom pair's graph where the frequency was

below the threshold value. Of course, where the allowed distance ranges in the query enclosed a screen boundary, molecules having either of the screens set were retrieved.

3.1.3 Analysis Of The Screen Performance

The screening system was analysed in [Jake87a] where ten pharmacophoric patterns from [Watt84] were searched against 12728 of the molecules in the Cambridge Crystallographic Database and the performance is shown in table 3.1 (taken from [Jake87a]). For some of the patterns, not all of the distances between the atoms were specified.

Although the screens are efficient in that they do "screen out" a substantial proportion of the molecules in the database which do not contain the pattern, they are less efficient in this sense than 2D substructure searching screens. However, this can be partly explained by the fact that the query patterns used for 2D searching are considerably larger than those for 3D searching (which are generally composed of between 3 and 6 atoms), and the large amount of time and effort which has been put into designing 2D screening systems. More details of the screening system can be found in [Jake87b].

Having established an appropriate methodology for the implementation of the screening component of a 3D substructure search system, the question then arises as to how the second-level search, the 3D equivalent of atom-by-atom searching, should be carried out. This type of search, which Jakes et al. refer to as geometric searching, is considered in detail in this chapter.

3.2 PARTIAL MATCHING OF TOPOGRAPHIC PATTERNS

3.2.1 Reported Algorithms

Various 3D matching methods for determining whether a topographic pattern of atoms is present in a molecule have been reported by Sundaram et al. [Sund74], Gund et al. [Gund74, Gund77, Gund79], Lesk [Lesk79], Kuntz et al. [Kunt82], Golender and Rozenblit [Gole83], and Danziger and Dean [Danz85]. ([Kuhl84] also discusses the method of Golender and Rozenblit but in the slightly different context of determining how similar two molecules are, and this will be considered in more detail in Chapter 4). However, [Sund74] is not very relevant in the present context as it assumes that the position of one of the pharmacophoric atoms is already known in the molecule under investigation and then the dihedral angles are varied so as to try to match other atoms with the rest of the query. Once a correspondence is known, numerous algorithms have been described for rotating and translating pattern atoms on to specified structure atoms (including [Bari81] which allows the molecules to be flexible by rotating about single bonds). However these algorithms can only be used as a final stage in the search because of the need for a knowledge of which structure atoms match which pattern atoms.

Lesk has described an algorithm primarily for use in searching for patterns in proteins but which can also be used with smaller molecules. The algorithm assigns candidate matches to each pattern atom on the basis of whether an atom has other atoms at all the same distances as the pattern atom. All of the molecule's atoms which are not matched with any pattern atom are removed from consideration and the candidate matches are checked again to make sure that the removals have not led to candidates no longer having other atoms at the required distances. This process is repeated until no more eliminations can be made. All the possible combinations produced from the candidate/pattern atom groups are then tested to see whether they match the pharmacophore by trying to rotate the combination of atoms onto the pattern [McLa82].

The algorithm of Kuntz et al. tries to find an optimal match between a ligand and a receptor by successively matching atoms from the two structures which have the highest number of distances to other atoms in common. When four atoms have been matched, the molecules are then compared by being rotated onto each other. Hence the algorithm finds substructural features in common between the two molecules rather than simply determining whether one molecule is contained in the other. [Kunt82] also mentioned that this algorithm can take "a few hours of computer time", but in this example, macromolecules were being used as receptors.

Danziger and Dean's method is a best match search in that it determines the best geometric fit between specified sets of points rather than checking whether part of one molecule is the same as the other molecule. It uses a tree search to match points from each step and pruning is carried out by calculating a dissimilarity measure for each branch. The depth of the tree search is the number of atoms in the smaller molecule, although the number of atoms which are attempted to be matched can be reduced by using null correspondences.

Besides their use in conjunction with screening systems which underlies the work reported in this chapter, partial matching algorithms are also useful in fields such as the steric difference QSAR method [Moto81]. Here a series of biologically active compounds is superimposed on to the most active compound's pharmacophoric pattern and a weighting scheme is subsequently produced.

3.2.2 Comparison Of Partial Matching Algorithms

Four partial matching algorithms were coded in FORTRAN 77 and their performances were compared. The four methods were:

3.2.2.1 Lesk's Algorithm

This was chosen as it was designed specifically to detect whether a 3D pattern occurs in a molecule or not, and it was described in outline in Section 3.2.1. In more detail, the algorithm consists of a series of steps:

1) Form an array of triples where each triple consists of the distance between a pair of atoms and the two atom types.

2) Associate two bit strings with each pattern atom, the entries in the first string corresponding to the atom types present in the pattern, and those in the second string to the distances between atoms in the pattern.

3) For each pattern atom, set the bit in string one associated with its atom type.

4) For each pair of pattern atoms, set the bits in the second strings which correspond to the elements in the "triple" array of step one which have the same atom types as the pair and where the distance equals that between the pair of atoms within the specified tolerances.

5) Associate the above two bit strings with all the structure atoms under consideration.

6) For each of these structure atoms, set the bit in string one associated with its type (if one exists).

7) For all pairs of these structure atoms, set the bits in the second string in a similar manner to step 4.

8) Check whether each of the structure atoms being considered has all the attributes (shown by ones in the bit strings) of at least one pattern atom. If it does not, remove it from the set of relevant structure atoms. If any atom has been eliminated from the structure, return to step 5.

9) For each pattern atom form a list of the structure atoms which are possible matches for it.

10) Form all possible combinations from step 9 and test to see whether they are a match (by using a rotation if necessary).

The coded version of the algorithm used arrays of integers rather than bit strings so as to avoid the system overheads which manipulating bits often cause. (The use of bit strings in the reported version of the algorithm [Lesk79] stems from the fact that it was developed to deal with macromolecules and this will be considered in Section 3.2.6).

3.2.2.2 A Set Reduction Algorithm

Set reduction [Suss65, Figu72] involves the successive elimination of atoms from sets corresponding to each pattern atom on the basis of an analysis of the atom's neighbours and higher order connectivities. Lesk's algorithm can be regarded as a variant of this technique along with the algorithm described in this section, which is that in use with the screening system of Section 3.1.

The n pattern atoms are labelled from 1 to n and for each of the n*(n-1)/2 distances between atoms in the pattern, a list of pairs of atoms from the query molecule is produced. The distance between the atoms in these pairs is equal to that between the pattern atoms to the allowed tolerances, and the atom type of the first atom corresponds with the type of the first pattern atom and similarly for the second atom. Thus if the query atoms are both carbons, two entries will be made in the list (the latter having the atoms in an opposite order to the former).

The main stage consists of taking each pattern atom in turn and finding the smallest list of pairs associated with this atom. For each pair in this list, checking that the atom in correspondence with the pattern atom corresponds with the pattern atom in the pattern atom's other (n-2) lists. If it does not, the pair is removed from the list. When the list has been processed, the pairs in the pattern atom's other lists are checked to see whether the atom which corresponds to the pattern atom does so in the list which was processed first. If it does not, then again the pair is removed.

The main stage is repeated until no further eliminations can be made. A final stage checks the possible combinations which can be produced from the pair lists. This is done by using a depth first search to try and find a successful combination as follows (where Pi is the i^{th}

pattern atom):-

1) (Initialisation) Set the level, L, of the search to the value one, INDEX(1), the index into the pair list P1-P2, to one and COMBIN(1), the structure atom currently matching pattern atom 1, to the atom corresponding to P1 in the first atom pair of P1-P2.

2) Set L=L+1 and COMBIN(L), the current structure atom under consideration, to the atom corresponding to PL in the INDEX(L-1)th atom pair in P1-PL.

3) Check the Pi-PL (i=2,..,L-1) pair lists to ensure that the pairs (COMBIN(i), COMBIN(L))are present in the relevant lists. If so then go to step 6 (the next level of the search).

4) (Backtrack) Find the first pair in the P1-PL list which is greater than INDEX(L-1) and whose first atom is COMBIN(1). Set INDEX(L-1) to the number of this pair, COMBIN(L) to the second atom and go to step 3. If no pair is found, then go to step 5.

5) Set L=L-1. If L=1, then set INDEX(1) to INDEX(1)+1 and go to step 2 (unless INDEX(1) is greater than the number of pairs in the P1-P2 list when the program terminates as the pattern is not contained in the structure), otherwise go to step 4.

6) Set L=L+1. If L is greater than the pattern size, then the pattern has been found and the program terminates, otherwise find the first pair in P1-PL with the first atom equalling COMBIN(1) and set INDEX(L-1) to be the number of this pair and COMBIN(L) to be the second atom of the pair. If no pair is found go to step 4, otherwise go to step 3.

Where only k of the distances in the pattern are specified, the method is the same as that above but only k lists are used. In fact, Gund [Gund79] has pointed out that for n greater than 3 only 4*(n-3)+2 lists need to be used as then not all of the n*(n-1)/2 distances in the pattern are independent, with a similar observation applying to Lesk's algorithm. However, a slight drawback is that, whereas a molecule can be reconstructed from 4*(n-3)+2 suitably chosen exact inter-atomic distances, if the distances are only specified as ranges of values, then two "reconstructions" of the molecule using different values in these ranges can magnify these differences. Also, this economy only begins to have a significant effect when the pattern is of size 9 or greater, and it was not employed for the algorithms under test.

3.2.2.3 A Clique Detection Method

Graphs were briefly mentioned in Chapters one and two and any 3D molecule can be regarded as being a labelled, weighted graph, that is one where the arcs between nodes are associated with real numbers (in this case the distance between the atoms which the nodes represent). A way of finding the largest subgraph in common between two graphs has been described by Levi [Levi72], and Barrow et al. [Barr76, Barr81]. A graph can be transformed into a totally connected, weighted graph where the values on arcs between nodes which were originally unconnected, have the number zero (and otherwise have the value one, or their original weight if the graph was weighted).

A correspondence graph can be formed from the transformed graphs of the two original graphs, by

1) creating the set of all pairs of nodes from the two graphs such that the nodes of each pair are of the same type.

2) forming a graph whose nodes are the pairs from (1). Two nodes (A1,B1), (A2,B2) are connected if the values of the arcs from A1 to A2 and B1 to B2 in the transformed graphs are the same.

Maximal common subgraphs then correspond to cliques (subgraphs where every node is connected to every other node and which are not contained in any larger subgraph with this property) of the correspondence graph, and finding cliques in graphs is a problem which has been widely studied. The efficiency of this method for finding maximal common subgraphs stems from the fact that tests which would need to be made several times in a naive tree search are only carried out once in setting up the correspondence graph.

As an illustration of this method, consider the unlabelled graphs A and B shown in figure 3.2. As the nodes are all of the same type, the nodes of the correspondence graph, C, are all the pairs (Ai,Bj) (i=1,..,3;j=1,..,4). If these nodes are enumerated as C1=(A1,B1), C2=(A1,B2), C3=(A1,B3), C4=(A1,B4), C5=(A2,B1), ..., then the connectivity matrix for the correspondence graph is given in figure 3.3. The subgraph isomorphisms now correspond to subsets of nodes of C of size three where all the nodes are connected to each other, ie. to the cliques that are present. One example of such a clique is C1=(A1,B1), C7=(A2,B3) and C12=(A3,B4).

This method has been applied in the chemical context by Kuhl et al. [Kuhl84] and Golender and Rozenblit [Gole83]. The latter have used it to find out if a pattern occurs in a molecule by looking for cliques in the correspondence graph whose size is the same as that of the pattern and it was this method which was coded. The clique detection was carried out by the algorithm of Bron and Kerbosch [Bron73] which is one of the quickest of the clique finding algorithms (others will be considered in the context of finding the maximal common substructure between molecules in Chapter 4).

3.2.2.4 Ullman's Subgraph Isomorphism Algorithm

As was mentioned above, 3D chemical structures can be regarded as being weighted graphs, and so the

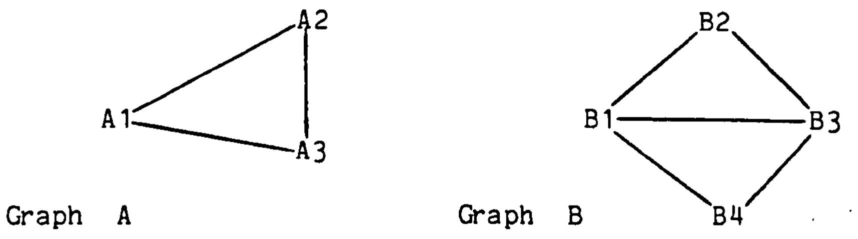


Figure 3.2 Unlabelled Graphs Used To Illustrate The Clique Finding Algorithm

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
C1	1	0	0	0	0	1	1	1	0	1	1	1
C2	0	1	0	0	1	0	1	0	1	0	1	0
C3	0	0	1	0	1	1	0	1	1	1	0	1
C4	0	0	0	1	1	0	1	0	1	0	1	0
C5	0	1	1	1	1	0	0	0	0	1	1	1
C6	1	0	1	0	0	1	0	0	1	0	1	0
C7	1	1	0	1	0	0	1	0	1	1	0	1
C8	1	0	1	0	0	0	0	1	1	0	1	0
C9	0	1	1	1	0	1	1	1	1	0	0	0
C10	1	0	1	0	1	0	1	0	0	1	0	0
C11	1	1	0	1	1	1	0	1	0	0	1	0
C12	1	0	1	0	1	0	1	0	0	0	0	1

Figure 3.3 The Connectivity Matrix For The Correspondence Graph Formed From The Graphs Of Figure 3.1

where C1=(A1,B1), C2=(A1,B2),..., C5=(A2,B1),...,C12=(A3,B4)

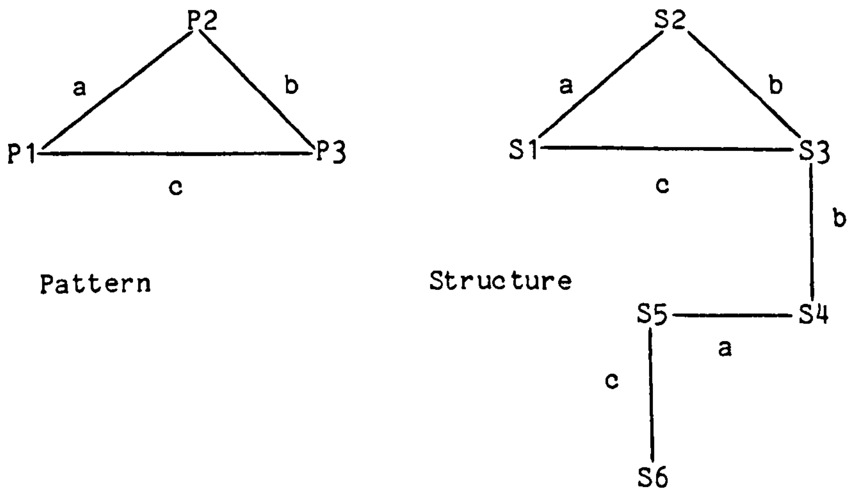


Figure 3.4 The Pattern And Structure Used To Illustrate The Four 3D Substructure Searching Algorithms

problem of finding a pattern in a molecule becomes that of subgraph isomorphism. Therefore it was decided to compare the above methods of pattern detection with a standard subgraph isomorphism algorithm reported by Ullman [Ullm76].

The algorithm begins with three main arrays A, B and M0 of sizes m*m, n*n and m*n respectively, where m is the number of nodes in the pattern and n is the number in the structure's graph. A and B are the connectivity matrices while the elements of M0 have the value one if the relevant pattern and structure nodes could match each other, and zero otherwise. The algorithm uses a tree search algorithm to try to alter M0 into a matrix M where each row contains a single one and each column contains no more than one one, by changing ones into zeros. M represents a permutation of the nodes of the structure's graph, and so, if C is the matrix M*Btranspose*Mtranspose, then M specifies a subgraph isomorphism if

C1 $\underset{1\leq i\leq m;}{\forall i}\ \underset{1\leq j\leq m}{\forall j} \quad (a_{ij}=1) \Rightarrow (c_{ij}=1)$

The basic algorithm works by forming a series of matrices Md (d=1,..,m) each one being created from its predecessor M(d-1) by systematically changing all but one of the ones in a row to zero. The final matrix is checked to see whether it satisfies the conditions imposed on M, if it does not, then backtracking occurs.

Ullman modifies this naive tree search by adding a refinement procedure. This procedure stems from the fact that, for a subgraph isomorphism, if ax is a neighbour of aw in the pattern and bz in the structure matches with aw, then there must exist a neighbour, by, of bz which matches with ax (and the relevant entry for ax-by in M must be one). Therefore for any subgraph isomorphism, if aw corresponds with by, then

$$C2 \quad (\forall_{1 \leqslant x \leqslant m} x) \quad ((a_{ij}=1) \Rightarrow (\exists_{1 \leqslant y \leqslant n} y) \quad (m_{xy} \cdot b_{yj}=1))$$

. The refinement procedure tests each one in Md to see whether the condition is satisfied, changing the one to zero if it is not. If any change took place, the procedure is repeated. If Mm is left unchanged by condition 2, then Mm represents a subgraph isomorphism.

The algorithm's steps can now be stated:-

1) Form matrices A, B and M0. Set D, the depth of the tree search to 1. Set M equal to M0 and then refine M. If the new M has one row of all zeros, then go to step 5.

2) If there is no node in the structure's graph which could match pattern node D and which has not already been provisionally matched with an earlier pattern node, then go to step 7.

3) Find, from M, the next potential match for pattern node D. Set all other entries in the D^{th} row of M to zero and refine M. If the new M has one row of all zeros, then go to step 5.

4) If D is equal to the pattern size, then a subgraph isomorphism has been found otherwise go to step 6 (the next level of the search tree).

5) If there are no more potential matches for pattern node D (compare with step 2) go to step 7. Otherwise set M equal to MD and go to step 3 (to try this new potential match).

6) Increase D by one (the next level of the tree search) and go to step 2.

7) No match has been found at this point in the tree search. If D=1 then terminate else subtract one from D, set M equal to MD and backtrack to step 5.

An implementation of the refinement procedure in hardware which allows a degree of parallel computation, is also suggested, but this has not been constructed.

[Ullm76]'s statement of the method was modified so as to deal with labelled, weighted graphs by changing condition 2 to condition 3.

C3 $\underset{1 \leq x \leq m}{(\forall x)} \quad \underset{1 \leq y \leq n}{(\exists y)} \quad (| m_{xy} . b_{yj} - a_{ix} | < e)$

where e is the allowed tolerance for two distances to be "equal".

The various improvements to the algorithm discussed in [McGr79, Chen81] were not coded.

3.2.3 A Worked Example

To illustrate the four algorithms, their operation on the (very artificial) pattern and structure shown in figure 3.4 will be examined. The nodes are all taken to be of the same type and the inter-atomic distances which are not marked are assumed to be different from those present in the pattern.

Figure 3.5 shows the bit strings for Lesk's algorithm which contain the information about the distances each atom is from the other atoms. The first iteration removes structure atom 6 as its bit string does not contain any of the pattern's bit strings. The structure's bit strings are then recalculated and the second iteration removes structure atom 5 because it no longer has an atom at a distance c from it. Finally, the third iteration removes structure atom 4, and, as the fourth iteration does not remove any structure atoms, the remaining three structure atoms are passed on to the final stage of Lesk's algorithm.

Figure 3.6 gives the pair lists produced by the set reduction algorithm. The first atom and atom pair examined by the algorithm are P1 and P1-P2. Structure atoms 2 and 4 are not a match for P1 because they are not present in P1's column of the structure atoms in the P1-P3 pair list. Therefore the S2-S1 and S4-S5 atom pairs can be eliminated from the P1-P2 pair list. After processing the P1-P2 list, the algorithm forms the set of possible matches

	Distance		
Atom	a	b	c
P1	1	0	1
P2	1	1	0
P3	0	1	1

The bit strings associated with the pattern by Lesk's Algorithm.

	Distance		
Atom	a	b	c
S1	1	0	1
S2	1	1	0
S3	0	1	1
S4	1	1	0
S5	1	0	1
S6	0	0	1

The bit strings initially associated with the structure by Lesk's Algorithm.

	Distance		
Atom	a	b	c
S1	1	0	1
S2	1	1	0
S3	0	1	1
S4	1	1	0
S5	1	0	0

The structure's bit strings after the removal from consideration of atom S6.

	Distance		
Atom	a	b	c
S1	1	0	1
S2	1	1	0
S3	0	1	1
S4	0	1	0

The structure's bit strings after the removal of S5.

	Distance		
Atom	a	b	c
S1	1	0	1
S2	1	1	0
S3	0	1	1

The structure's bit strings before entry to the final stage of Lesk's algorithm.

Figure 3.5 Applying Lesk's Algorithm To The Pattern And Structure Of Figure 3.3

Pattern Pair	P1	P2	P2	P3	P1	P3
	S1	S2	S2	S3	S1	S3
Structure	S2	S1	S3	S2	S3	S1
Pairs	S5	S4	S4	S3	S5	S6
	S4	S5	S3	S4	S6	S5

The initial pair lists for the set reduction algorithm.

Pattern Pair	P1	P2	P2	P3	P1	P3
	S1	S2	S2	S3	S1	S3
Structure	S5	S4	S3	S2	S3	S1
Pairs			S4	S3	S5	S6
			S3	S4	S6	S5

The pair lists after the elimination of pairs from P1-P2 which did not have a first element which could match P1.

Pattern Pair	P1	P2	P2	P3	P1	P3
	S1	S2	S2	S3	S1	S3
Structure	S5	S4	S3	S2	S5	S6
Pairs			S4	S3		
			S3	S4		

The pair lists after the elimination of pairs from P1-P3 which did not have a first element which could match P1.

Pattern Pair	P1	P2	P2	P3	P1	P3
	S1	S2	S2	S3	S1	S3
Structure	S5	S4	S3	S2		
Pairs			S4	S3		
			S3	S4		

The pair lists after the elimination of pairs from P1-P3 which did not have a second element which could match P3.

Pattern Pair	P1	P2	P2	P3	P1	P3
Structure	S1	S2	S2	S3	S1	S3
Pairs	S5	S4	S4	S3		

The pair lists after the elimination of pairs from P2-P3 which did not have a second element which could match P3.

Figure 3.6(a) The Set Reduction Algorithm Applied To The Pattern And Structure Shown In Figure 3.3

Pattern Pair	P1	P2	P2	P3	P1	P3
Structure	S1	S2	S2	S3	S1	S3
Pairs			S4	S3		

The pair lists after the elimination of pairs from P1-P2 which did not have a first element which could match P1.

Pattern Pair	P1	P2	P2	P3	P1	P3
Structure Pairs	S1	S2	S2	S3	S1	S3

The pair lists before the final stage of the algorithm.

Figure 3.6(b) The Set Reduction Algorithm Applied To The Pattern And Structure Shown In Figure 3.3

for P1, {S1,S2}, from this list. The pairs in P1-P3 whose first atom is not in this set are then eliminated (these being S3-S1 and S6-S5). The algorithm then proceeds by applying a similar procedure in turn to the atoms P3 and P1. When no more eliminations can be made from the pair lists, they are passed on to the final stage of the algorithm.

The clique finding algorithm produces the correspondence graph from the graphs of the pattern and the structure in exactly the same way as the illustration of figure 3.2. For example, consider the nodes of the correspondence graph C5=(P1,S5), C10=(P2,S4) and C18=(P3,S6), then

1) as the distances between P1-P2 and S4-S5 are both a, C5 is connected to C10 in the correspondence graph.

2) as the distances between P1-P3 and S5-S6 are both c, C5 is connected to C18.

3) the distances between P2-P3 and S4-S6 are not the same, so C10 is not connected to C18.

After setting up the correspondence graph, the problem then becomes one of determining whether the correspondence graph contains a clique of size 3.

For Ullman's algorithm the structure atoms were re-ordered as (S5, S4, S2, S1, S3, S6) so as to avoid the method immediately finding the match (S1, S2, S3) in the first 3 rows and columns of the matrix M0. The distance tables for this new ordering are shown in figure 3.7. As all the atoms are of the same type, any structure atom

	1	2	3
1	0	a	c
2	a	0	b
3	c	b	0

Distance
Table A

	1	2	3	4	5	6
1	0	a	X	X	X	c
2	a	0	X	X	b	X
3	X	X	0	a	b	X
4	X	X	a	0	c	X
5	X	b	b	c	0	X
6	c	X	X	X	X	0

Distance Table B

Figure 3.7 Ullman's Distance Tables For The Example Of Figure 3.3

where X indicates that the distance is not one of those contained in the pattern.

	1	2	3	4	5	6
1	1	0	0	1	0	0
2	0	1	1	0	0	0
3	0	0	0	0	1	0

Figure 3.8 Matrix M After Its First Refinement

i	j	x	distance	y	Mxy	Action
1	1	2	a	2	1	
1	1	3	c	6	0	M(1,1):=0
2	3	1	a	4	1	
2	3	3	b	5	1	No change
2	2	1	a	1	0	
2	2	3	b	5	1	M(2,2):=0
1	4	2	a	3	1	
1	4	3	c	5	1	No change
3	5	1	c	3	1	
3	5	2	b	2, 4	0, 1	No change

Figure 3.9 Refining Matrix M Of Figure 3.8 Using Condition 3 Of Section 3.2.2.4

	1	2	3	4	5	6
1	0	0	0	1	0	0
2	0	0	1	0	0	0
3	0	0	0	0	1	0

Figure 3.10 Matrix M After The Refinement Illustrated In Figure 3.8

could possibly match any pattern atom, and so, M0 is initially a 3*6 matrix of ones. (A more sophisticated approach might be to check that the structure atoms had neighbours at the right distances.)

The first step in the algorithm requires M to be set equal to M0 and then refined. The refinement involves finding each element (Mij) which has the value one and then checking whether, for these values of i and j, whether condition 3 holds. If it does not, (Mij) is set equal to zero. The matrix M after the first application of the refinement procedure is shown in figure 3.8.

As the refinement procedure changed some elements of M, it is reapplied to M. In more detail, figure 3.9 shows the working out of condition 3 for each non-zero element of M. First i and j are assigned to be the row and column numbers respectively of the non-zero element. Next x and the relevant pattern distance are found from matrix A before y is found from matrix B. Finally, the element (Mxy) from M is examined and if it is zero, then (Mij) is set equal to zero. Figure 3.10 shows M after the second application of the refinement procedure and a match for the pattern has been found in the structure without any recourse to backtracking (and this was found to be the usual case in the searches which were undertaken).

3.2.4 Results Of The Comparison

The first three algorithms were compared with a small sample of the data output from the queries of Section 3.1.3 using the VAX 8600 system at Pfizer (U.K.). For convenience reasons, patterns where all the distances were specified were chosen. A further restriction was the fact that Lesk's method uses the same error margin when comparing a distance in the query molecule with one in the pattern for all the pattern distances (although this problem can be overcome by using an array containing the error margins). The results are given in table 3.2, the missing entry for Lesk's algorithm being caused by the fact that one of the molecules contained multiple occurrences of the pattern. When this occurs, the algorithm calls its transformation stage at least n to the power r times where n is the number of atoms in the pattern and r is the number of disjoint (that is no atoms in common) occurrences of the pattern.

Although the sample of data is very small, there is a slight indication that the clique finding algorithm is the quickest followed by the set reduction method with Lesk's algorithm third. However, it was decided that a more meaningful analysis would not be worthwhile because

1) the performances of the algorithms of Sections 3.2.2.2 and 3.2.2.3 were pretty similar

2) the time taken for the partial matching stage is much smaller than that for the two screening stages which

typically take about 10 minutes of real time on the VAX 8600 [Jake87a]. (However, most of this time is taken up by the database retrieval operations rather than the calculations associated with the screening program.)
3) where there are a large number of structures to be checked by the partial matching stage in table 3.1, there is also a high success rate on the partial matching stage due to the effectiveness of the screening system.

Therefore the algorithms were analysed in a less specific setting than that of having small patterns, and molecules which contain all the distances in the pattern. This was done by taking molecules of various sizes and extracting "patterns" of atoms of different sizes from these molecules. These atoms were chosen to be mainly carbons so as to make the problems more computationally demanding due to the fact that most atoms in the molecules are carbons. (Hence, it was a "worst case" test since pharmacophoric patterns normally involve heteroatoms.) The patterns were then slightly distorted so that the algorithms would no longer find them in the molecules. The distance error margin for two distances to be regarded as matching was set at 0.25 A in all the runs.

These comparisons were run on a Prime 9950 and the results are given in tables 3.3 to 3.11. To improve the accuracy of the timing, each run involved 20 searches for the pattern in the query molecule with the recorded time being the time taken divided by 20. Some of the table entries for Lesk's algorithm are not monotonically increasing with respect to the pattern size because of the combinatorial problem mentioned above and this also caused the searching of the molecule of size 25 using Lesk's algorithm to be prohibitively expensive. Additionally, only Ullman's algorithm was used to search the molecule of size 106 atoms because of the computational cost.

A further comparison was carried out using 250 molecules from the start of the Cambridge Crystallographic Database. 10 molecules which were evenly spaced throughout the 250, had patterns selected from them by using random numbers to select 3, 5 and 7 carbons. The algorithms were then run to see how many times these patterns occurred in the 250 molecules and the means and standard deviations of the times for these runs are given in tables 3.12. Unfortunately, Lesk's algorithm suffered from its combinatorial problem and no times were obtained for it. Also only 9 patterns of size 5 and 8 of size 7 were used because one of the molecules contained only 4 carbons and another 5.

3.2.5 Discussion Of The Comparisons

While the patterns of atoms which were chosen were very artificial, they do allow the various algorithms to be compared in computationally expensive circumstances. Bearing this in mind, the resulting comparison can only be regarded as a fairly rough, general indication of their performances. Clearly, in specific circumstances such as being used in conjunction with the screening system of Section 3.1 or to find whether a substructure common to two molecules occurs in a third [Gole83], a detailed analysis in that setting would be required. However, the following points can be made:

1) Ullman's algorithm is the quickest with it giving comparatively better performances as the pattern size increases (although Gund's comment in Section 3.2.2.2 could probably offset this to a certain extent). It is also noticeable that it is quicker at unsuccessful searches than successful ones. However, because the time taken for each search is very low, it is hard to envisage any use for the hardware proposed in Section 3.2.2.3 in this context.

2) The problem under investigation was to determine whether a specific 3D pattern of atoms was present in a molecule, and so, when a clique of the same size as the pattern was found by the method of Section 3.2.2.3, the search was successful and could terminate. If no clique of sufficient size was present in the correspondence graph, then all the cliques were generated by the algorithm so as to establish this fact. Therefore this method tended to perform better than the set reduction algorithm on successful searches and worse on unsuccessful ones.

3) When it did not suffer from its combinatorial problem, Lesk's algorithm was in the same performance range as the

clique finding and set reduction approaches, with it doing relatively better with large patterns. The combinatorial problem was exacerbated to some extent in the comparisons by using mainly carbon atoms in the pattern and a distance error range of 0.25 A. Steps 1 to 4 of the algorithm (see Section 3.2.2.1) only need to be executed once for each pattern which is a slight advantage if a large number of structures are being searched for the same pattern.

3.2.5.1 The Combinatorial Problem Suffered By The Reduction Methods

The simplest approach to the substructure search problem is to test all the possible combinations of structure atoms against the pattern, but this is hopelessly expensive in practice because of the factorial nature of the method. Both Lesk's and the set reduction algorithms operate by trying to reduce the number of structure atoms that are passed on to a final stage which is similar to this simple approach. (However, the set reduction method manages to avoid generating most of the possible combinations by using the depth first search described in Section 3.2.2.2 as a final stage.) Unfortunately, these methods are not always able to reduce the number of structure atoms to a number which the generating all combinations approach can handle, and so Lesk's algorithm can run into problems.

Consider the (pathological) pattern and structure shown in figure 3.11 where the unmarked distances are assumed not to be relevant and all the atoms are of the same type. Lesk's algorithm is unable to eliminate any atoms from the structure as each one has neighbours at the required distances, while the performance of the set reduction method is shown in figure 3.12 and again no progress has been made at eliminating any of the atoms (although the structure does not contain the pattern). In this case, only a few atoms are passed to the final stage of each algorithm, and so the fact that they have not been able to eliminate some "unmatchable" structure atoms is not significant. However, where there is a large amount of symmetry in the structure, this failing to be able to remove structure atoms can swamp the final stage of the algorithm. An illustration of this was searching for patterns of size 7 amongst the 250 molecules in Section 3.2.4, where one of the searches using the set reduction method produced a final stage with 15 possible matches for the first pattern atom, 16 for the second, 18 for the third, 16 for the fourth, 14 for the fifth, 14 for the sixth and 15 for the seventh. The simple approach for the final stage was swamped by the number of possible combinations and was aborted after it had used over 40 minutes of c.p.u. time. However, the depth first search ran to completion in a fraction of second.

Lesk's algorithm can never eliminate more structure atoms from consideration than the set reduction approach as it is only interested in whether an atom has

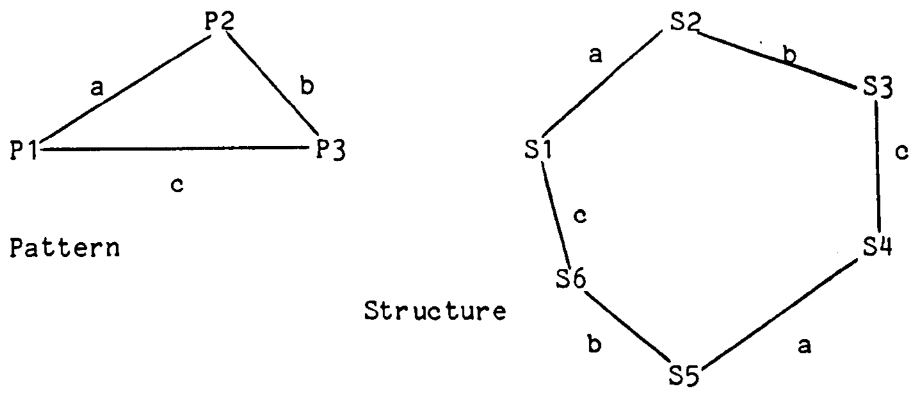


Figure 3.11 A Pattern And Structure Which Cause Problems For The Reduction Techniques

Pattern Pair	P1	P2	P2	P3	P1	P3
	S1	S2	S2	S3	S1	S6
Structure	S2	S1	S3	S2	S6	S1
Pairs	S4	S5	S5	S6	S3	S4
	S5	S4	S6	S5	S4	S3

The initial pair lists for the set reduction algorithm.

Pattern Pair	P1	P2	P2	P3	P1	P3
	S1	S2	S2	S3	S1	S6
Structure	S4	S5	S3	S2	S4	S3
Pairs			S5	S6		
			S6	S5		

The pair lists after eliminating atoms which could not match P1.

Pattern Pair	P1	P2	P2	P3	P1	P3
Structure	S1	S2	S2	S3	S1	S6
Pairs	S4	S5	S5	S6	S4	S3

The final pair lists which the reduction techniques cannot make any smaller.

Figure 3.12 The Set Reduction Algorithm Applied To The Pattern And Structure Of Figure 3.10

neighbours at the right distances, rather than whether these neighbours are also potential matches for the relevant pattern atom. Unfortunately, the simple approach has to be employed with Lesk's algorithm, and so, in cases like the above, it cannot cope.

3.2.6 Searching Macromolecules

3.2.6.1 Modifications To Lesk's Algorithm

The molecules contained in the Cambridge Crystallographic Database are all relatively small when compared with proteins, and so, it was decided to search a molecule containing 1807 atoms. Unfortunately, the storage space required by the clique finding method and Ullman's subgraph isomorphism algorithm was beyond the limits of the Prime 9950. In addition, to save storage space, the code for Lesk's algorithm was amended to use the bit strings described in the statement of the algorithm instead of the arrays of integers which were used when searching the Cambridge Crystallographic Database. The manipulation of bits on the Prime carries a considerable overhead when compared with the corresponding manipulation of integers; so as to get some idea of the extra cost the version of Lesk's algorithm using bit strings was run to (successfully) find the pattern of size 9 in the structure of size 42 (see table 3.7). The time taken was 2.41 cpu seconds compared with the original time of 0.99 cpu seconds.

A further modification of the version of Lesk's algorithm used above, was that the region occupied by the molecule was split up into non-overlapping cubes of side length equal to the maximum inter-atomic distance in the pattern plus the error limit [Levi66, Katz72]. Step 7 of Section 3.2.2.1 was altered so that additionally each structure atom was associated with its relevant cube. The purpose of the modification is that in step 7, when distances from a structure atom to other structure atoms are being considered to see if they match an inter-atomic pattern distance, only structure atoms from the original atom's cube and the cubes adjacent to this need to be considered. To see the effect of this splitting up of the molecule, two versions of Lesk's algorithm were used in the search of the macromolecule, one not using cubes and the other having 6*6*6 cubes arranged to make a larger cube. (Any structure atoms not present in one of the 216 cubes were assigned to the nearest cube.) The version of Lesk used on the Cambridge Crystallographic Database did not use cubes because when the maximum inter-atomic pattern distance is reasonably large when compared with the distances in the molecule, the extra processing involved in the cubes method leads to a significant degradation in the performance of the algorithm.

Whereas with the searching of the molecules in tables 3.3 to 3.12 the pattern was chosen so as to make the search time consuming, it was decided that the searches of the macromolecule should use less demanding patterns. Also in order to investigate the use of the cubes under very favourable conditions, consecutive atoms were extracted from the molecule (thus keeping the maximum inter-atomic distance in the pattern small). One set of twelve atoms was taken from the "start" of the molecule and the other set from the "middle". The error margin of 0.1 A when comparing distances was lower than before so as again to cut down on the computational cost. For the same reason, each search was only run once as opposed to the twenty times used in Section 3.2.4, but otherwise the method was the same as that of the above section.

3.2.6.2 Results And Overview

The results of the searches are given in tables 3.13 to 3.16 and these indicate that Lesk's algorithm with the addition of cubes performed best and the set reduction algorithm worst. Again Lesk's algorithm suffered from combinatorial problems and where these occurred, the time used upto the final step in the algorithm is given and is marked with a $ sign. However, a way round this problem might be to use Ullman's or one of the other algorithms as the final stage of Lesk's algorithm rather than using a generate all possible combinations of atoms approach (that is to use Lesk's method to reduce the number of structure atoms under consideration to a level where one of the other algorithms could be used). The number of structure atoms under consideration at step 5 of the algorithm on each iteration is also given.

Tables 3.13 to 3.16 show that it is possible to search for fairly distinctive patterns in a macromolecule without being devastatingly expensive in terms of the cpu time used, especially when the fact that bit strings were used is kept in mind. A way of reducing this cost could be to split the molecule up into blocks and use a screening system similar to that of section 3.1, treating each block as though it was a separate molecule. Alternatively, work is being carried out in Sheffield to try to use Lesk's algorithm to reduce the number of atoms under consideration to a level where Ullman's algorithm can be applied [Davi87].

3.3 COMMENTS

A 3D substructure searching system used for finding pharmacophoric patterns in the Cambridge Crystallographic Database has been described and various partial matching algorithms have been compared. Although the various tests indicate that Ullman's subgraph isomorphism algorithm is the quickest, the highly artificial nature of the patterns which were searched for and the fact that the partial matching stage takes relatively little time when compared with the screening stage must be emphasised. Also using the algorithms to search macromolecules was found to be expensive in terms of the c.p.u. time used. As with the results in the rest of this thesis, the performance figures of the algorithms can only be regarded qualitatively because of possible inefficiencies in the coding, the performance of different computers, compilers and languages, and most importantly of all, the lack of use made of "parallel" bit handling facilities. However, overall, 3D substructure searching can be regarded as being easier than 2D substructure searching in that it deals with weighted graphs. Additionally, it seems that regarding the structures as graphs and then using a standard subgraph isomorphism algorithm leads to a better performance then the algorithms developed from a chemical standpoint. The subgraph isomorphism used was Ullman's standard one but any of several others [McGr79, Chen81], some of which are claimed to be substantially quicker, could have been used instead and might have also given good results in this application.

A closely related problem to that of determining whether a pattern is present in a molecule (subgraph isomorphism) is that of determining what structure two or more molecules have in common (maximal common subgraph) and two algorithms for this will be considered in the next chapter. Later in this thesis (Chapter 8), a description of a program combining the screening system of Section 3.1 and one of these algorithms will be given in an attempt to provide a way of searching the Cambridge Crystallographic Database for a molecule which has a similar 3D structure to the pattern molecule.

PATTERN	N	D	Stage 1	Stage 2	Partial Match
Anti-cholinergic	4	5	368	80	4
ab-adrenergic	5	6	325	24	0
Anti-leukemic	3	3	485	370	171
Anti-malarial	6	9	379	78	0
Anti-neoplastic	3	3	283	56	0
Hallucinogenic	3	3	542	191	69
Serotoninergic	3	3	519	51	0
Prostaglandin-like	3	3	690	55	4
Steroid hormonal	4	3	1106	183	102
Analgesic	4	4	666	414	259

Table 3.1 Results Of The Screening System Of Section 3.1 (Taken From [Jake87a])

N is the number of atoms in the pattern
D is the number of distances in the pattern
Stage 1 is the initial screening stage
Stage 2 is the check to see whether the distances are present

PATTERN	Number Of Molecules	Number Of Matches	Time For Lesk	SR	Clique
Anti-neoplastic	9	0	1.7	1.7	1.5
Hallucinogenic	19	1	4.4	4.1	3.5
Serotoninergic	24	3	***	3.5	2.9

Table 3.2 Comparison Of The Partial Matching Algorithms

Showing: the number of molecules considered
: the number of molecules in which a match was found
The time is the cpu time (in seconds) averaged over three runs
SR is the set reduction algorithm of section 3.2.2.2
Clique is the clique finding algorithm of section 3.2.2.3

NB. Different distance error limits were used in tables 3.1 and 3.2.

Algorithm	Pattern Sizes 3	5	7	9
Lesk	17	7	13	23
Set reduction	3	7	9	13
Clique finding	3	6	10	15
Ullman	3	4	6	8

Table 3.3 Times(*) For Successfully Finding A Pattern In A Structure Of Size 14

Algorithm	Pattern Sizes 3	5	7	9
Lesk	3	4	9	16
Set reduction	2	4	7	10
Clique finding	3	6	11	16
Ullman	2	2	2	2

Table 3.4 Times(*) For Unsuccessfully Finding A Pattern In A Structure Of Size 14

Algorithm	Pattern Sizes 3	5	7	9	11
Lesk	**	**	***	***	***
Set reduction	77	88	137	129	154
Clique finding	16	39	75	128	195
Ullman	12	19	26	28	38

Table 3.5 Times(*) For Successfully Finding A Pattern In A Structure Of Size 25

Algorithm	Pattern Sizes 3	5	7	9	11
Lesk	**	**	**	***	***
Set reduction	11	22	61	83	113
Clique finding	18	62	148	263	446
Ullman	5	6	17	13	13

Table 3.6 Times(*) For Unsuccessfully Finding A Pattern In a Structure Of Size 25

* The cpu times are in hundredths of a second.

Algorithm	Pattern Sizes 3	5	7	9	11	13
Lesk	148	26	54	99	143	205
Set reduction	15	33	59	92	134	168
Clique finding	10	30	54	89	131	160
Ullman	10	13	18	25	35	44

Table 3.7 Times(*) For Successfully Finding A Pattern In A Structure Of Size 42

Algorithm	Pattern Sizes 3	5	7	9	11	13
Lesk	31	22	46	83	125	169
Set reduction	14	35	63	97	137	181
Clique finding	10	34	69	115	166	200
Ullman	9	10	8	9	9	9

Table 3.8 Times(*) For Unsuccessfully Finding A Pattern In A Structure Of Size 42

Algorithm	Pattern Sizes 3	5	7	9	11	13	15
Lesk	45	136	88	131	198	291	454
Set reduction	24	59	89	129	194	277	407
Clique finding	36	121	173	294	431	564	903
Ullman	20	27	32	43	58	73	92

Table 3.9 Times(*) For Successfully Finding A Pattern In A Structure Of Size 60

Algorithm	Pattern Sizes 3	5	7	9	11	13	15
Lesk	29	139	83	126	188	274	412
Set reduction	19	51	94	138	196	269	409
Clique finding	38	164	221	439	720	945	1693
Ullman	13	13	17	17	17	18	18

Table 3.10 Times(*) For Unsuccessfully Finding A Pattern In A Structure Of Size 60

* The cpu times are in hundredths of a second.

	Pattern Size	
	3	5
Pattern present	178	136
Pattern absent	76	86

Table 3.11 Times(*) For Ullman's Algorithm To Search A Molecule Of Size 106 Atoms

* The cpu times are in hundredths of a second.

	Pattern Size					
	3		5		7	
Algorithm	Mean	S.D.	Mean	S.D.	Mean	S.D.
Set reduction	64	17	120	63	125	66
Clique finding	44	1	109	15	235	24
Ullman	31	2	37	10	33	10

Table 3.12 Times(+) For Searching 250 Molecules For Patterns Consisting Of Three, Five And Seven Carbons

+ The cpu times are in seconds.
S.D. is the standard deviation

FOR TABLES 3.13 TO 3.16:-
The cpu times are in seconds
$ indicates that this time is the time upto step 10
Mk1 is the version which doesn't use cubes
Mk2 is the version which does use cubes

Algorithm	Pattern Sizes 6	8	10	12
Set reduction	73	103	107	189
Lesk Mk1	$85	100	146	292
Lesk Mk2	$33	53	104	221

Table 3.13(a) Times For Successfully Finding Atoms From The "Start" Of A Molecule Of Size 1807

Number Of Iteration	Pattern Sizes 6	8	10	12
First	1807	1807	1807	1807
Second	715	589	346	150
Third	258	103	11	12
Fourth	128	8	10	
Fifth	56			
Sixth	42			
Seventh	35			

Table 3.13(b) The Corresponding Number Of Structure Atoms Under Consideration at Step 5 Of Lesk's Algorithm On Each Iteration

Algorithm	Pattern Sizes 6	8	10	12
Set Reduction	41	65	91	191
Lesk Mk1	$78	97	148	273
Lesk Mk2	$26	56	107	240

Table 3.14(a) Times For Unsuccessfully Finding Atoms From The "Start" Of A Molecule Of Size 1807

Number of iteration	Pattern Sizes 6	8	10	12
First	1807	1807	1807	1807
Second	71	144	136	48

Table 3.14(b) The Corresponding Number Of Structure Atoms Under Consideration At Step 5 Of Lesk's Algorithm On Each Iteration

Algorithm	Pattern Sizes 6	8	10	12
Set reduction	105	187	tmsr	tmsr
Lesk Mk1	92	146	260	360
Lesk Mk2	46	107	231	330

Table 3.15(a) Times For Successfully Finding Atoms From The "Middle" Of A Molecule Of Size 1807

Number Of Iteration	Pattern Sizes 6	8	10	12
First	1807	1807	1807	1807
Second	701	665	400	319
Third	158	78	14	12
Fourth	19	9	10	
Fifth	6	8		

Table 3.15(b) The Corresponding Number Of Structure Atoms Under Consideration At Step 5 Of Lesk's Algorithm On Each Iteration

Algorithm	Pattern Sizes 6	8	10	12
Set reduction	167	294	tmsr	tmsr
Lesk Mk1	106	171	307	421
Lesk Mk2	62	136	283	395

Table 3.16(a) Times For Unsuccessfully Finding Atoms From The "Middle" Of a Molecule Of Size 1807

Number Of Iteration	Pattern Sizes 6	8	10	12
First	1807	1807	1807	1807
Second	806	740	469	360
Third	259	94	12	8
Fourth	60	3		
Fifth	2			

Table 3.16(b) The Corresponding Number Of Structure Atoms Under Consideration At Step 5 Of Lesk's Algorithm On Each Iteration

CHAPTER 4

COMMON 3D SUBSTRUCTURES

4.1 INTRODUCTION

Finding the 3D substructure in common between several molecules can be loosely regarded as being a generalization of 3D substructure searching. This problem is of interest if several molecules which are biologically active are known as large common regions may contain the active site, ie. the part of the molecule which is responsible for the activity. [Moto86] states that using an algorithm for solving this problem in conjunction with the molecular mechanics program described in his paper, could lead to "a powerful, computationally integrated approach to pharmacophore identification, validation, and assessment of uniqueness".

This chapter describes and compares two algorithms for the determination of common substructures. The first of these is the method of Crandell and Smith [Cran83a, Cran83b] which works by finding all common substructures of size n and then "grows" these so as to produce all those of size n+1. The other method is very closely related to the clique finding algorithm of Section 3.2.2.3 and treats the molecules as weighted graphs (see Section 3.2.2.3). The problem of finding maximal common substructures becomes that of finding maximal common subgraphs [Levi72, Barr76]. Before moving on to describe the algorithms, it should perhaps be pointed out that the problem of whether a graph contains a clique of size k or greater is NP-complete (see Section 1.2.1.2) and the problem of listing all the cliques of a graph has a running time which may in "bad cases" increase exponentially with the size of the graph as there can be an exponential growth in the number of cliques [Das78].

4.2 CRANDELL AND SMITH'S ALGORITHM

The method Crandell and Smith described [Cran83a, Cran83b] for finding the 3D substructures in common between a set of molecules, involves taking all the common substructures of size n associated with each molecule and adding an extra atom to each of them. These enlarged substructures are then canonically named so as to allow them to be compared with the enlarged substructures associated with other molecules. If a substructure is not found in all of the other molecules' lists, it is deleted from consideration. The surviving substructures form the common substructures of size n+1. This type of growing and comparing algorithm has also been used to compare 2D molecular data [Vark79].

The selection of an atom to add to a substructure in the "growing" step is done by consulting a distance matrix associated with each molecule. This contains the distances between all atoms in the molecule and distances which are not present in the current set of common substructures are indicated by a minus sign (this amendment of the distance tables being carried out after the comparison step). Atoms for addition to a substructure are those whose distances have not been negated.

The algorithm can be summarised as consisting of the following steps:-

1) Setting up the distance tables
2) Growing the common substructures
3) Naming the substructures
4) Comparing the substructures
5) Amending the distance tables and returning to step 2.

[Cran83a] describes modifications to the method to allow for a common starting substructure to be specified which must be contained in any substructures produced by the algorithm, and to cater for stereochemistry, but these will not be considered here.

4.2.1 Setting Up The Distance Tables

So as to make the comparison between substructures in step 4 simpler, the inter-atomic distances in each molecule have an integer associated with them. This is done by forming a list of inter-atomic distances present in the molecules for each atom type pair. These lists are then sorted into ascending order and the distances in them are grouped together so that a distance belongs to the same group as its predecessor if the difference in their values is less than the tolerance value (which is usually taken to be 0.09 A), otherwise a new group is formed. The groups are then numbered starting from one and groups which do not contain atom pairs from every molecule, have their numbers negated. A distance table is associated with each molecule and the (i,j)th entry (i<>j) is the group number for the inter-atomic distance between this molecule's ith and jth atoms.

4.2.2 Growing

Each substructure associated with a molecule is grown by enlarging its atom set by one by adding an atom which is greater than any of the atoms in the atom set (where greater just refers to the "natural" ordering of the atoms resulting from their input) and whose "distances" in the distance table to these atoms are non-negative. Where it is possible to add several different atoms, a new substructure is produced for each of them.

On the first iteration, the "grown" substructures are taken to be the individual atoms in each molecule.

4.2.3 Naming

Each substructure node set produced from step 2 is given a canonical name by taking each of the $(n-1)*n/2$ atom pairs in the substructure (where n is the size of the substructure) and forming a triple consisting of the two atom types (with the larger coming first) and the relevant "distance" entry in the molecule's distance table. These (n-1)*n/2 triples are then sorted so that if X=(a,b,c) and Y=(d,e,f) are two triples, then X occurs before Y if

1) a>d

2) a=d and b>e

or

3) a=d, b=e and c<f.

Once the triples have been sorted, each triple need only be represented by its "distance" element as this implicitly contains the atom type information. Hence each substructure can be uniquely named (up to isomorphism) by the list of its "distances" (ordered as above).

4.2.4 Comparing

For each of the molecules, its "named" substructures are compared with those of the other molecules. If another molecule is found which does not have this substructure amongst its substructures, then the substructure is deleted along with its node set. Hence, the substructures surviving this step are the substructures in common for this size.

4.2.5 Amending The Distance Tables

After the comparison stage, each non-negative entry in the distance tables has its atom pair checked to see whether it still occurs in the relevant molecule's list of node sets. If it does not, the distance entry is negated so as to avoid growing substructures which contain this atom pair at some future moment in time.

4.3 USING GRAPH THEORY TO FIND COMMON SUBSTRUCTURES

The maximal common 3D substructures between two molecules can be found by treating the molecules as weighted graphs and then finding the maximal subgraphs in common. One approach to this problem [Levi72, Barr76] was described in Section 3.2.2.3 and is to produce the correspondence graph of the two molecules and then to find all the cliques in this graph. An alternative approach has been described by McGregor [McGr82] involving a depth first search tree in which each tree node represents the pairing of a node from the first graph with one from the second. The advantage of this approach is that it allows a wider definition of subgraph to be employed than that of Levi because a subgraph can now be defined as a subset of the nodes of a graph along with a subset of the edges which join these nodes. (As opposed to a subset of the nodes of the graph and all the edges of the graph between these points.) Figure 4.1 illustrates a graph and subgraph which

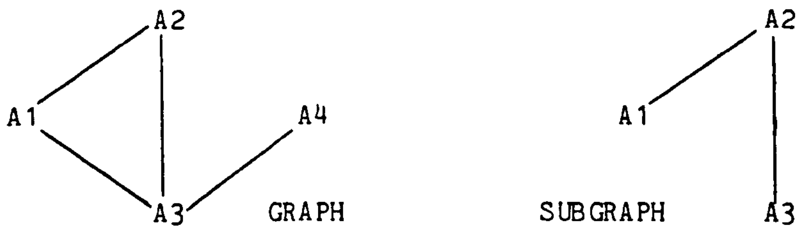
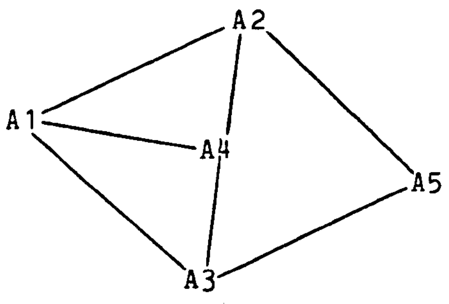


Figure 4.1 An Illustration Of The Different Definitions Of A Subgraph Employed By McGregor And Levi



	A1	A2	A3	A4	A5
A1	1	1	1	1	0
A2	1	1	0	1	1
A3	1	0	1	1	1
A4	1	1	1	1	0
A5	0	1	1	0	1

Figure 4.2 The Graph Used To Illustrate The Clique Finding Algorithms

McGregor's definition allows but Levi's disallows (because A1 is not connected to A3 in the subgraph). This approach leads to the definition of a maximal common subgraph as being the subgraph contained in both graphs which has the largest number of edges. Hence, the emphasis is very much on the edges of the graphs and the technique uses an m*n matrix, MARCS, (where m is the number of edges in the first graph and n is the number in the second) containing ones and zeros. A one in the $(r,s)^{th}$ entry means that the r^{th} edge of the first graph is a potential match for the s^{th} edge of the second and the matrix is altered in a way somewhat analogous to that of the matrix M0 in Ullman's subgraph isomorphism algorithm of Section 3.2.2.4.

In more detail, if a node x from graph one is associated with a node y from graph two at a node of the search tree, then any arc, r, connected with x can only correspond with arcs connected to y (other entries in the r^{th} row being set to zero -and likewise for the relevant columns). As is usual with tree searches, the efficiency of the algorithm is closely linked with how soon "bad" branches which cannot lead to a solution can be pruned. In this case, this means trying to ensure that a common subgraph with a large number of arcs is found early on in the search and then backtracking whenever the number of rows of MARCS which contain at least one one falls below this number of arcs.

McGregor's approach has been applied in the chemical information field to help identify the bond changes that have occurred in chemical reactions [McGr81]. However, in the present context it is not very relevant as

1) it only finds a largest common subgraph whereas Levi's method finds all the subgraphs which are not contained in a larger subgraph.

2) all the nodes in a graph (or subgraph) are connected to each other as the "weight" of the edge represents the distance between the two atoms.

3) following on from (2), not only is McGregor's wider definition of subgraph of no extra use, but the matrix MARCS is now very large.

An alternative algorithm for the maximal common subgraph problem which uses McGregor's definition of a subgraph is described in [Wong83]. This method produces a third graph from the two originals and then employs a depth first tree search to find areas of maximum correspondence in this new graph. Pruning is carried out by keeping a matrix at each level of the search which gives the maximum number of edges which can be obtained by a potential pairing of nodes from the starting graphs. The algorithm was designed for directed graphs and on conversion to dealing with undirected graphs and the other definition of subgraph, becomes very similar to the correspondence graph method using Bron and Kerbosch's clique finding algorithm described below.

Levi's method stems from the idea that, with his definition of maximal common subgraph, it is likely that an efficient algorithm will repeatedly have to test whether the relationship between A_i-A_j and B_i-B_j (where A_k is an element of the first graph and B_k is an element of the second) is the same whenever (A_i,B_i) and (A_j,B_j) are potential correspondences. Hence, it is more economical to store this information in a graph (the "correspondence graph") and the problem is thus transformed into the well studied clique detection problem.

In the last chapter, only the standard clique finding algorithm of Bron and Kerbosch was used; however, as finding common substructures is a more complex task than determining whether a subgraph isomorphism exists, several different clique finding algorithms were coded and compared with each other (as opposed to the last chapter where a single clique finding algorithm was considered). The algorithms chosen were:-

1) Bron and Kerbosch's algorithm [Bron73] which is generally regarded as being one of the most efficient of the clique finding algorithms.

2) Golender and Rozenblit's algorithm [Gole83] which they used with their common substructure detection system.

3) Version 1 of algorithm 1 described by Gerhards and Lindenberg [Gerh79] as it was reported as performing better than Bron and Kerbosch's algorithm on sparse graphs.

4) Loukakis and Tsouros' algorithm [Louk81] as it has been reported as being quicker than that of Bron and Kerbosch.

5) Loukakis' algorithm [Louk83] which has been reported as being quicker than algorithm (4).

Before giving a detailed description of these, it is probably just as well to point out that, with the exception of (3), they all employ a fairly similar depth first, tree search strategy. This uses a set of nodes, A, which is the current attempt at a clique, a set of nodes from which elements are taken to enlarge A and some indication of nodes which have previously been rejected from A. Hence, the efficiency of the algorithms derives from the data structures they use and the conditions employed to "prune" branches of the search tree as soon as possible. Therefore the descriptions of the algorithms are rather mathematical with that of (2) being the simplest and those of (3), (4) and (5) being the hardest. A reader who is not interested in the exact details of the algorithms can continue at Section 4.3.7 without losing any sense of continuity.

4.3.1 Bron And Kerbosch's Algorithm

At each level, d, of the tree search, there are two sets Nd and Cd of nodes of the graph which are connected to every node in the set Md which consists of the d nodes under consideration for inclusion in the next clique. Nd contains the nodes which have already been tried in the attempt to enlarge Md, and Cd those "candidate" nodes which have yet to be tried. The algorithm moves to the next level of the tree search by moving a candidate node from Cd to the trial set Md (which then becomes

M(d+1)). The sets N(d+1) and C(d+1) are then calculated by removing from Cd and Nd those nodes not connected to the candidate node.

When backtracking occurs, the node most recently added to M(d+1) is added to Nd and removed from Cd, and the level of the search becomes d (from its previous value of (d+1)). A clique is found when both Cd and Nd are empty (if only Cd is empty, then Md is a subset of a clique which has already been output).

The selection of a candidate node from Cd is done so as to increase the likelihood of a point in Nd being connected to all points in Cd. (When this happens, further extensions to Md from Cd cannot remove this point from Nd. Therefore Nd can never become empty by extending Md, and so, backtracking needs to occur.) This can be done by selecting the point, n, in Nd which is connected to the most elements of Cd and then every time a candidate is selected, choosing a point in Cd which is not connected with n (because if backtracking occurs it is removed from Cd).

4.3.2 Golender And Rozenblit's Algorithm

This method uses an array EXPAND(L) at each level, L, of the search tree to hold the candidates for addition to the array CLIQ which contains the current attempt at finding a clique. When an element, J, for addition to CLIQ is chosen, EXPAND(L+1) is produced from

EXPAND(L) by intersecting it with NEIGHBR(J), the set of neighbours of J which are greater than J. Hence the elements of EXPAND(I) are always connected to every element of CLIQ and they are greater than every element of CLIQ. A variable, K, is used to ensure that the cliques are not generated twice. This is done by increasing K whenever a new node is added to CLIQ, only decreasing K when backtracking occurs and selecting an element from EXPAND(I) to be the smallest element which is greater than K.

When no more elements can be added to CLIQ, the following test is applied to determine whether a clique has been produced as opposed to a subset of an earlier clique:-

<u>TEST 1</u> For the last node, J, added to CLIQ, find a neighbour, M, which is smaller than J and which is connected to all elements of CLIQ. If no such M exists, a clique has been found.

In order to improve the efficiency of this naive search, a second test is used to try to prune the tree in cases where any potential clique that the algorithm can produce, will fail on test one. This test is applied before a new node, J, is added to CLIQ and can be stated as:-

<u>TEST 2</u> Find M which is a neighbour of J but is not contained in CLIQ, such that M<J and M is connected with every element of EXPAND(L) and <u>every element of CLIQ.</u>

The point of the test is that enlarging CLIQ by using elements from EXPAND(L) will always mean that a clique cannot be produced because every element of CLIQ will still be connected to M. A full statement of the algorithm is:-

1) Set R, the root of the search, equal to one.

2) Set L, the level of the search equal to one, CLIQ(1) equal to R and EXPAND(1) equal to NEIGHBR(R).

3) Set K equal to R.

4) Select, J, the smallest element of EXPAND(L) which is greater than K. If no such J exists, go to step 7.

5) Perform test 2. If a suitable M is found, go to step 10 so as to prune the tree.

6) Add a new vertex to CLIQ by increasing L by one, producing EXPAND(L) and setting CLIQ(L) equal to J.

7) If no new vertex has been added to the search tree, backtrack by going to step 10.

8) Perform test 1. If a suitable M is found, then CLIQ is not a clique and backtracking occurs by going to step 10.

9) Output CLIQ (as it is a clique).

10) Backtrack by setting K equal to CLIQ(L), decrementing L by one and if L<>0 going to step 4.

11) Choose a new root by incrementing R by one. If R is less than or equal to the size of the graph, go to step 2.

4.3.3 Gerhards And Lindenberg's Algorithm

[Gerh79] describes two clique finding algorithms along with the results of various comparisons with the algorithm of Bron and Kerbosch, and the first version of the first algorithm was found to perform well with respect to this algorithm on sparse graphs. The first algorithm is based on the following theorem (the proof of which is given in [Gerh79]):-

Every subset Q=(TGLE(i) union with K) of NGLE(i) is the generating vertex set of a clique of B(i) if, and only if, K, which is a subset of NGLE(i)\TGLE(i), is the vertex set of a clique of the subgraph $S^*(i)$ of S(i) generated by ((NGLE(i)\TGLE(i)) union with R(i)) where R(i) is the subset of NG(i)\NGLE(i) whose elements are connected in G with all elements of TGLE(i).

where
G is the graph under consideration,
NG(i) is the set of elements of G which are connected to i (including itself),
NGLE(i) is a subset of NG(i) containing those elements which are less than or equal to i,
TGLE(i) is a subset of NGLE(i) containing those elements which are connected to every element of NGLE(i),
B(i) is the subset of all cliques of G which contain i and whose other vertices are less than i

and

S(i) is the graph whose node set is NG(i).

The basic algorithm consists of

1) Incrementing i, calculating NG(i) and NGLE(i) and then finding B(i) by steps 2 to 8.

2) Applying a simple test (given below) to try and determine whether B(i) is the empty set. If it is, return to step 1.

3) Determining whether i is connected to any other vertices. If it is not, then output B(i)={i} and return to step 1.

4) Deriving TGLE(i) from the connection table of the graph G.

5) If TGLE(i) is actually NGLE(i), then output B(i)=TGLE(i) and go to step 1.

6) Determine R(i) which is the subset of NG(i)\NGLE(i) whose elements are connected with all the elements of TGLE(i).

7) Determine X, the node set of $S^*(i)$, from X=((NGLE(i)\TGLEI(i)) union with R(i)).

8) Determine all the K's in the above theorem by calling a subroutine K-CAL, and all the sets in B(i) from B(i)={K union with TGLE(i)}. Then go to 1.

The test used in step 2 to try to determine whether B(i)={} can be stated as

<u>TEST</u> If NG(i)\NGLE(i) contains an element M such that M is connected to every element of NGLE(i), then B(i)={}.

This follows from the fact that any clique whose largest element was i would have every element connected to M (>i), and so, would not be a clique (contradicting the first statement).

In step 8, the set K is obtained by calling the subroutine K-CAL which uses a tree search to generate the subsets of NGLE(i)\TGLE(i), backtracking occurring when either the present subset is the vertex set of a clique or there are two nodes in the subset which are not connected to each other. The test for a clique is carried out using:-

<u>TEST</u> Y is the vertex set of a clique of Z if, and only if, Y is equal to the intersection of NZ(j) for every j in Y.

This comes from considering the intersection because if it contains an element x which is not in Y, then Y cannot be a maximal, complete subgraph as Y union with {x} is a complete subgraph which contains it. On the other hand, if there is an element, w, of Y which is not in the intersection, then Y cannot be totally connected (complete), and so, again it is not a clique.

4.3.4 Loukakis And Tsouros' Algorithm

An independent set of a graph, G, is a set of nodes none of which are connected to each other. A maximal independent set is an independent set which is not contained in a larger independent set and it is closely related to a clique in that a clique of a graph is a maximal independent set of the complementary graph and vice versa. (The complementary graph is formed from a graph by removing all the edges and then connecting those nodes which were originally unconnected.)

The search of this section generates the maximal independent sets lexicographically in that it produces all those containing node 1 before those which do not and within these two groups it produces those containing node 2 first, and so on. It achieves this by using three disjoint sets (SPLUS, SMINUS and STWIDDLES) of nodes of the graph. SPLUS contains an independent set which is the basis of the next maximal independent set. SMINUS contains the nodes which have been removed from SPLUS when backtracking has occurred. STWIDDLES consists of the nodes of G which are not in SPLUS, SMINUS or connected to any element of SPLUS. The algorithm operates by trying to add elements of STWIDDLES to SPLUS so as to create a larger independent set, the lexicographic ordering being achieved by always choosing the smallest possible element of STWIDDLES to add on branching and removing the most recently added element from SPLUS on backtracking.

To improve efficiency, the algorithm also makes use of the following two propositions:-

P1 Let u, an element of STWIDDLES, be the branching vertex and N(u) be a subset of (N(SPLUS) union with SMINUS), then the vertex u is contained in any maximal independent set which contains SPLUS and no element of SMINUS.
where N(u) is all the nodes which are connected to u
and N(SPLUS) is all the nodes which are connected to at least one element of SPLUS

Proof

The condition means that any extension to SPLUS (which does not contain u) will not contain any neighbours of u, and so, u can always be added to this extension and the extension will still be independent.

P2 Let K be the set of elements of SMINUS which are not connected to any element of SPLUS. If u is an element of K and u is not connected to any element of STWIDDLES, then u is contained in any maximal independent set which contains SPLUS.

Proof

u is not connected to any element of SPLUS or STWIDDLES, and so, extending SPLUS by taking elements from STWIDDLES will still mean that u can be added to SPLUS without SPLUS ceasing to be an independent set.

Proposition P1 means that u can be considered along with its predecessor for backtracking purposes, and P2 means that no maximal independent set can be produced along this branch of the search and backtracking should occur.

Loukakis and Tsouros' algorithm can now be given in a step by step form:-

1) (Initialize) SPLUS={}, STWIDDLES=V (the nodes of the graph) and SMINUS={}

2) Check whether STWIDDLES is empty in which case SPLUS is a maximal independent set and the algorithm goes to step 5, otherwise SPLUS is augmented by the first element of STWIDDLES (which is recalculated).

3) If P1 is satisfied, mark the most recently added element of SPLUS and go to step 2, else go to step 4.

4) If P2 is satisfied, go to step 5 else go to step 2.

5) (Backtrack) Find the most recently added unmarked element in SPLUS, restore SMINUS to its state when this element was added to SPLUS and add this element to SMINUS. Remove this element and all the more recently added elements from SPLUS and recalculate STWIDDLES. If SPLUS is the empty set, go to step 6, otherwise go to step 4.

6) (Termination test) Apply P2, if it is satisfied then terminate as no more backtracking is possible else go to step 2.

4.3.5 Loukakis' Algorithm

This algorithm is very similar to that of the previous section in that it finds the maximal independent sets by using the sets SPLUS, SMINUS and STWIDDLES. The theorem which the algorithm is based around can be stated as:-

<u>THEOREM</u> SPLUS is a maximal independent set of a graph if, and only if, SMINUS is a subset of ADJ(SPLUS) and STWIDDLES is the empty set.
(where ADJ(A) is the set of nodes which are connected to at least one element of A)

<u>Proof</u>

The proof can be split up into three cases as follows:-

1) SMINUS is not a subset of ADJ(SPLUS)

Therefore if u is an element of SMINUS but not of ADJ(SPLUS), then SPLUS intersection with ADJ(u) is the empty set. (If this was not the case, then if w was an element in this intersection, u would be an element of ADJ(w) which is a subset of ADJ(SPLUS).) Hence, SPLUS union with {u} is an independent set, and so, SPLUS cannot be a maximal independent set.

2) STWIDDLES is not empty

STWIDDLES is defined to be
V\(the union of SPLUS, ADJ(SPLUS) and SMINUS) where V is the set of nodes of the graph.
Therefore adding any element of STWIDDLES to SPLUS creates a larger independent set.

3) SMINUS is a subset of ADJ(SPLUS) and STWIDDLES is empty
From the definition of STWIDDLES it follows that V is the union of SPLUS and ADJ(SPLUS). Therefore any set larger than SPLUS which contains SPLUS, must contain an element of ADJ(SPLUS), and so, cannot be an independent set.

A condition closely related to this theorem is:-
C1 If SMINUS is a subset of ADJ(SPLUS) and there exists an element, u, of STWIDDLES such that ADJ(u) intersection with STWIDDLES is empty, then u is contained in every maximal independent set formed by adding elements of STWIDDLES to SPLUS.
Proof
From the definition of STWIDDLES, the intersection of ADJ(u) with the union of SPLUS and STWIDDLES is empty. Therefore any expansion of SPLUS by elements of STWIDDLES (which does not contain u) to form a new independent set, can have u added to it and still remain independent.

The point of condition C1 is that u can be added to SPLUS and when backtracking on u should occur, the backtracking can be done on the predecessor of u in SPLUS (because a maximal independent set formed from the elements of SPLUS preceding u is forced to contain u).

As was mentioned above, the algorithm generates the maximal independent sets by growing SPLUS by adding elements of STWIDDLES. If SMINUS is a subset of ADJ(SPLUS), then the theorem says that SPLUS should be extended by an element of STWIDDLES and whenever such a "branch" occurs, condition C1 is tested. When backtracking occurs, the last element, v, added to SPLUS which is suitable for backtracking (that is condition C1 was not met when it was added to SPLUS) is found and SMINUS is restored to its state when v was added to SPLUS. v is then removed from SPLUS and added to SMINUS. Whenever SMINUS contains an element v which is not in ADJ(SPLUS), the next element, w, of STWIDDLES is chosen so that v is in ADJ(w). (Of course if no such element exists, then backtracking must occur.)

The steps of the algorithm are:-

1) (Initialize) Set SPLUS={}, SMINUS={}, STWIDDLES=V (the nodes of the graph) and u to be the element of the graph with fewest neighbours. Go to step 3.

2) Test whether STWIDDLES is empty, if it is then the theorem applies (because step 4 has already tested that SMINUS is a subset of ADJ(SPLUS)) and SPLUS is a maximal independent set and backtracking occurs by going to step 5. Otherwise u=the first element of STWIDDLES and proceed to step 3.

3) If C1 is satisfied, then mark u as it can be associated with the most recently added element of SPLUS when backtracking.

4) Add u to SPLUS and recalculate STWIDDLES. If SMINUS is not a subset of ADJ(SPLUS), then go to step 6. (So as to try to choose an element of STWIDDLES to make this so.) Otherwise carry on adding to SPLUS by going to step 2.

5) Backtrack by finding the most recently added, unmarked

element of SPLUS, restore SMINUS to its state when this element was added to SPLUS, add this element to SMINUS and remove it and all the more recently added elements from SPLUS. Recalculate STWIDDLES and test for termination by seeing whether the element, u1, considered first in step 1 and all its neighbours are in SMINUS, if so, then the conditions of the theorem can never be met. Otherwise proceed to step 6.

6) Try to ensure that SMINUS becomes a subset of ADJ(SPLUS) by choosing an element, u, of STWIDDLES such that w is an element of ADJ(u), is in SMINUS but not ADJ(SPLUS). If this is possible then go to step 3 (so as to add u to SPLUS), otherwise go to step 5 (so as to backtrack).

In the actual implementation of the algorithm, STWIDDLES is not recalculated from scratch all the time but rather on branching the elements subtracted from STWIDDLES are stored and when backtracking occurs, they are added to STWIDDLES.

4.3.6 A Worked Example

To illustrate the above algorithms, the way they deal with the graph of figure 4.2 (figure 4.6 in the case of the methods which find the maximal independent sets) will be considered.

Figure 4.3 illustrates the operation of the Bron and Kerbosch algorithm, Md is the set forming the basis of

LEVEL OF TREE SEARCH	Md	Nd	Cd	ACTION
1	1	{}	2 3 4	
2	1 2	{}	4	
3	1 2 4	{}	{}	Output Clique
2	1 2	4	{}	
1	1	2	3 4	
2	1 3	{}	4	
3	1 3 4	{}	{}	Output Clique
2	1 3	4	{}	
1	1	2 3	4	
1	2	1	4 5	
2	2 5	{}	{}	Output Clique
1	2	1 5	4	
1	3	1	4 5	
2	3 5	{}	{}	Output Clique
1	3	1 5	4	
1	4	1 2 3	{}	
1	5	2 3	{}	

Figure 4.3 Bron And Kerbosch's Algorithm Applied To The Graph Of Figure 4.2

LEVEL	J	R	K	CLIQ	EXPAND	ACTION
1	2	1	1	1	2 3 4	
2	4	1	2	1 2	4	
3	X	1	4	1 2 4	{}	Output Clique
2	X	1	4	1 2	4	
1	3	1	2	1	2 3 4	
2	4	1	3	1 3	4	
3	X	1	4	1 3 4	{}	Output Clique
2	X	1	4	1 3	4	
1	4	1	3	1	2 3 4	
2	X	1	4	1 4	{}	Backtrack
1	X	1	4	1	2 3 4	
1	4	2	2	2	4 5	
2	X	2	4	2 4	{}	Backtrack
1	5	2	4	2	4 5	
2	X	2	5	2 5	{}	Output Clique
1	X	2	5	2	4 5	
1	4	3	3	3	4 5	
2	X	3	4	3 4	{}	Backtrack
1	5	3	4	3	4 5	
2	X	3	5	3 5	{}	Output Clique
1	X	3	5	3	4 5	
1	X	4	4	4	{}	Backtrack
1	X	5	5	5	{}	Finish

Figure 4.4 Golender And Rozenblit's Algorithm Applied To The Graph Of Figure 4.2

The values are those after step 4 of the algorithm.
The value X indicates that a suitable J has not been found.

the next clique, Cd the candidates for addition to Md and Nd the previously rejected elements of Md which are connected to every element of Md. When a choice is possible over which element from Cd should be used to extend Md, the element which is connected to the least number of elements of Nd is chosen.

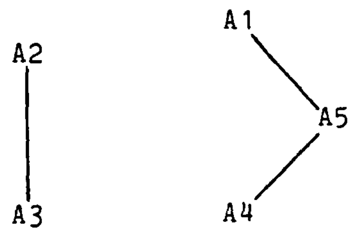
Figure 4.4 shows Golender and Rozenblit's algorithm in operation. The main idea is to increase CLIQ so that it becomes a clique by adding elements from EXPAND which are the smallest elements which are greater than K. (Also R, the root of the search tree and J, the next element to be added to CLIQ, are also given.)

Like Golender and Rozenblit's algorithm, Gerhards and Lindenberg's method produces the cliques in lexicographical order and figure 4.5 gives an outline of the steps involved with the example of this section. For each i, the set of cliques, B(i), with i as their greatest element is calculated using NG(i) (the set of elements connected to i -including itself), NGLE(i) (those elements of NG(i) which are not greater than i) and TGLE(i) (those elements of NGLE(i) which are connected to every other element of NGLE(i)).

The complementary graph of the graph of figure 4.2 is shown in figure 4.6 and, as figures 4.7 and 4.8 show, the operations of Loukakis and Tsouros' and Loukakis' algorithms on it are very similar. They both try to increase SPLUS by adding elements of STWIDDLES and mark (by *) elements of SPLUS which should be associated with the

```
NG(1)={1, 2, 3, 4}  NG(2)={1, 2, 4, 5}  NG(3)={1, 3, 4, 5}
NG(4)={1, 2, 3, 4}  NG(5)={2, 3, 5}
NGLE(1)={1}  NGLE(2)={1, 2}  NGLE(3)={1, 3}
NGLE(4)={1, 2, 3, 4}  NGLE(5)={2, 3, 5}


i=1
  B(i)={} as 2 is connected to every element of NGLE(i)

i=2
  B(i)={} as 4 is connected to every element of NGLE(i)

i=3
  B(i)={} as 4 is connected to every element of NGLE(i)

i=4
  Step 2  The test is not able to determine whether B(i)={}
  Step 3  B(i) is not isolated
  Step 4  TGLE(i)={1, 4}
  Step 5  B(i) is not equal to TGLE(i)
  Step 6  R(i)={}
  Step 7  X={2, 3}
  Step 8  K={ {2}, {3} }
          B(i)={ {1, 2, 4}, {1, 3, 4} }

i=5
  Step 2  The test is not able to determine whether B(i)={}
  Step 3  B(i) is not isolated
  Step 4  TGLE(i)={5}
  Step 5  B(i) is not equal to TGLE(i)
  Step 6  R(i)={}
  Step 7  X={2, 3}
  Step 8  K={ {2}, {3} }
          B(i)={ {2, 5}, {3, 5} }
```

Figure 4.5 Gerhards And Lindenberg's Algorithm Applied
To The Graph Of Figure 4.2



$$
\begin{array}{c|ccccc}
 & A1 & A2 & A3 & A4 & A5 \\
A1 & 1 & 0 & 0 & 0 & 1 \\
A2 & 0 & 1 & 1 & 0 & 0 \\
A3 & 0 & 1 & 1 & 0 & 0 \\
A4 & 0 & 0 & 0 & 1 & 1 \\
A5 & 1 & 0 & 0 & 1 & 1
\end{array}
$$

The Adjacency Matrix

Figure 4.6 The Complementary Graph Of The Graph Shown In Figure 4.2

AFTER STEPS	SPLUS	STWIDDLES	SMINUS	ACTION
1,2	1	2 3 4 5	{}	
3,4,2	1 2	4	{}	
3,4,2	1 2 4	{}	{}	
3	1 2 4*	{}	{}	4 is
	associated	with 2 for	backtracking	purposes
2	1 2 4*	{}	{}	Output MIS
5	1	3 4	2	
4,2	1 3	4	2	
3	1 3*	Associate 3 with 1		
2	1 3* 4	{}	2	
3	1 3* 4*	Associate 4 with 1		
2	1 3* 4*	{}	2	Output MIS
5	{}	2 3 4 5	1	
6,2	2	4 5	1	
3,4,2	2 4	{}	1	
3,4				
5	2	5	1 4	
4,2	2 5	{}	1 4	Output MIS
3,2	2 5*	{}	1 4	
5	{}	3 4 5	1 2	
6,2	3	4 5	1 2	
3,2	3 4	{}	1 2	
3,4				
5	3	5	1 2 4	
4,2	3 5	{}	1 2 4	
3,2	3 5*	{}	1 2 4	Output MIS
5,6		Terminate		

Figure 4.7 Loukakis And Tsouros' Algorithm Applied To The Graph Of Figure 4.6

AFTER STEPS	SPLUS	STWIDDLES	SMINUS	ACTION
1,3,4	1	2 3 4 5	{}	
2,3,4	1 2	4	{}	
2,3,4	1 2 4*	{}	{}	4 has been associated with 2 for backtracking purposes
2	1 2 4*	{}	{}	Output MIS
5	1	3 4	2	
6,3,4	1 3*	4	2	
2,3,4	1 3* 4*	{}	2	
2	1 3* 4*	{}	2	Output MIS
5	{}	2 3 4 5	1	
6,3,4	5	2 3	1	
2,3,4	5 2	{}	1	
2	5 2·	{}	1	Output MIS
5	5	3	1 2	
6,3,4	5 3*	{}	1 2	
2	5 3*	{}	1 2	Output MIS
5	TERMINATE			

Figure 4.8 Loukakis' Algorithm Applied To The Graph Of Figure 4.6

preceding element element of SPLUS when backtracking occurs. However, Loukakis and Tsouros' algorithm generates the maximal independent sets lexicographically while Loukakis' algorithm chooses elements of STWIDDLES to try to make SMINUS a subset of ADJ(SPLUS). Hence, when 1 is removed from SPLUS and added to SMINUS, Loukakis and Tsouros' algorithm chooses the element 2 from STWIDDLES while Loukakis' algorithm chooses 5.

4.3.7 Comparing The Algorithms

The five algorithms described above were coded in FORTRAN 77, with Bron and Kerbosch's being converted from the ALGOL 60 given in [Bron73] while the others were coded from their step by step descriptions. (The recursion present in the ALGOL 60 code was dealt with by making repeated copies of the subroutine which called itself and then modifying the copies so that they called each other in a chain.) A sort routine was also used with the algorithms of Gerhards and Lindenberg, Loukakis and Tsouros, and Loukakis so as to order the nodes of the graphs on how many neighbours they possessed. The time taken for this sorting stage is included in the times given below for these algorithms.

The code produced for the algorithms was optimised [Metc85] so as to try to obtain highly efficient implementations. However, it is unlikely that the resulting versions of the algorithms were as efficient as the authors' own and, in particular, the algorithm of Gerhards and Lindenberg caused problems mainly because there is very little description of the data structures to be used (and those which are present are intended for when logical "AND" and "OR" functions are available). On the other hand the implementation of Loukakis and Tsouros produced code comparable with that in [Louk81] and the simplicity of Golender and Rozenblit's approach leaves very little room for inefficient coding. The code for [Louk83] is also likely to be comparable with the author's own as the data structures are explicitly stated.

In addition to the above five algorithms, a modified version of Bron and Kerbosch's algorithm was also tested. The modification was that when a node was added to the list of nodes potentially making up the next clique, it was checked to see whether it was connected to all the candidates for addition to this list. If it was then, when backtracking to this node occurred, the algorithm could immediately backtrack to the node's predecessor in the list (as the node is contained in any clique containing the earlier nodes in the list.)

The algorithms were compared by taking a molecule from the Cambridge Crystallographic Database and producing "another" molecule from it by reordering the atoms. Both molecules then had some of their atoms slightly distorted so as to obtain two different but similar "molecules". The first stage of Crandell and Smith's algorithm was then applied to the molecules (this was so as to ensure that the clique finding approach and Crandell and Smith's algorithm produced the same common substructures) and the correspondence graph was produced from the resulting "distance" tables. The clique finding algorithms were then run (on a Prime 9950) on this correspondence graph.

The molecules chosen were of sizes 16 (14 carbons, 1 oxygen, 1 bromine), 23 (20 carbons, 3 oxygens), 19 (11 carbons, 8 oxygens, 2 nitrogens) and 25 (15 carbons, 9 oxygens, 1 nitrogen) and the results are given in tables 4.1 to 4.4. Gerhards and Lindenberg was aborted after it had taken 40 minutes of cpu time when finding a substructure of size 19 in table 4.3. Therefore no times are given for this algorithm in tables 4.3 and 4.4 (when a similar problem occurred). Additionally, a very large number of cliques of size 5 were produced when the molecules in Table 4.3 which have a common substructure of size 19 were distorted and this led to Bron and Kerbosch's algorithm having to be terminated after it had used 40 minutes of cpu time. Hence there are no times in this table for molecules with a small common substructure.

Tables 4.1 to 4.4 appear to indicate (even after taking into consideration the coding problems mentioned above) that the extra heuristics introduced by the algorithms of Gerhards and Lindenberg, Loukakis and Tsouros, and Loukakis increase the computation rather than reduce it (the extra computation being consumed by the calculation of the heuristics). This is probably due to the correspondence graphs being "simpler" than the random graphs that these algorithms were designed for in that, for example, if n nodes are connected to each other and another node is connected to the first (n-1) of these nodes, then it is very likely to be connected to the n^{th}. Hence, the time consuming tests do not lead to a significant improvement in the search tree. (This "less difficult" property of the correspondence graphs stems from the fact that not all the N*(N-1)/2 inter-atomic distances in the molécule are independent.)

One of the reasons for Gerhards and Lindenberg's poor performance can be seen in the figures given in [Gerh79] (for random graphs of size 36) where the time for a graph of edge density 10% is 0.5 seconds while that for one where the density is 90% is 1131 seconds. Hence, in practical applications (where the nodes are likely to be in "clusters") the algorithm is likely to perform badly even on sparse graphs.

Overall the algorithm of Bron and Kerbosch is significantly quicker than the other algorithms. However, the modified version of the algorithm is only very marginally quicker than the original and it was the original version which was used for the comparison with Crandell and Smith's method.

Unfortunately, all the algorithms ran into problems when dealing with the molecule of size 23 but this was an extreme case as over 32000 cliques of size 5 were present. However, as the size of the correspondence graph increased, even Bron and Kerbosch's performance began to deteriorate fairly rapidly and this will be considered in more detail in Sections 4.5 and 4.6 where the clique finding approach for finding the largest common substructures will be compared with that of Crandell and Smith. (The clustering of the inter-atomic distances using the first step of Crandell and Smith's algorithm will also be investigated in Section 4.5. The results of running the two versions of Bron and Kerbosch's algorithm on the correspondence graph produced by using a molecule composed of 24 carbons and the actual inter-atomic distances rather than the clustered ones is given in table 4.5. Of the other algorithms, Gerhards and Lindenberg took 136 cpu seconds when dealing with the common substructure of size 24 whilst the rest ran into storage problems -having been coded to optimise their speed of execution rather than to save space.) However a more in depth look at the choice between clustering or not clustering distances is given in Section 4.5.1.2 (tables 4.13 to 4.17).

4.4 MODIFYING THE TWO APPROACHES FOR FINDING COMMON SUBSTRUCTURES

The two methods which have been used for finding common substructures each have a major drawback. The basic version of Crandell and Smith's algorithm described above can compare several molecules with each other but it is expensive in cpu time taken and storage space required if there is a large common substructure. On the other hand, the clique finding method can deal very well with comparing two molecules which might have a large common substructure, but extending the comparison to 3 molecules by using triples (where one element comes from each molecule) to be the nodes of the correspondence graph, instead of the previous pairs, greatly increases the size of this graph (in the worst case by a factor of the size of the third molecule). Therefore the clique finding method runs into difficulties when comparing more than two molecules.

4.4.1 Sorting The Growths In Crandell And Smith's Algorithm

Table 4.7(a) shows the cpu times (for a Prime 9950) taken by each step of the described version of Crandell and Smith's method when comparing two slightly distorted versions of a molecule of size 16, while table 4.6 shows the number of grown substructures after step 3 of each generation. It can be seen that the comparison stage consumes most of the time and, when there are two very similar molecules, it is likely to be of the order of n*n*(the time taken to check whether two growths are equal), where n is the number of growths after step 3 of the algorithm. (This is because, if we assume that

1) All the growths associated with a molecule have different names

2) No growths are eliminated by the comparison stage

then, when comparing molecule A's growths with those of B's, each of B's growths matches with one of A's. The

number of comparisons a growth from A undergoes if it matches with growth j from B, is j. Therefore the total number of comparisons undergone by A's growths is
the sum 1+..+n which is n*(n-1)/2
which is of the order of n*n.)

However, if each molecule's name list is sorted into ascending order, then, using pointers which are increased if the relevant name is less than that pointed at by the other pointers, the lists can be compared in order n comparisons (with the above assumptions). A standard sorting routine can carry out the extra sorting in order n*log(n) time, which is significantly smaller than n*n as n becomes very large.

This modification was coded (using the sorting algorithm of [Sing69]) and its performance on the example described at the start of this section is shown in table 4.7. It can be seen that the time taken by the amending step is now of significance, and as this step is carried out only to try to improve the efficiency of the algorithm, it does not have to occur in the algorithm. Hence four versions of the algorithm were tested depending on whether the extra sorting and/or the amending stage was/were present. (The performance of the versions which did not have an amending stage on the above example are given in tables 4.8 and 4.9.). Nevertheless, it should be pointed out though that with very similar molecules the amending step does not make very many modifications to the distance tables, and so, the example given is almost a "worst case" for the algorithms incorporating an amending stage. (However, the situation is not quite this straightforward as the time taken by the amending stage is likely to be larger when the molecules are not nearly identical and this will be considered in Section 4.5.1.2).

4.4.2 Extending The Clique Finding Algorithm To More Than Two Molecules

As was mentioned in Chapter 3, the clique finding approach has been used in the chemical context by Golender and Rozenblit [Gole83] and by Kuhl et al. [Kuhl84], but neither of these applications was interested in finding common substructures for more than two molecules. The former used common substructural features in structure-activity relationships, producing the features by finding common substructures between pairs of molecules and determining whether they were present in the other molecules by using a subgraph isomorphism algorithm. On the other hand, Kuhl et al. were interested in finding receptor-ligand binding positions in the context of Crippen's "distance geometry" [Crip81] (which is a way of producing 3D co-ordinates for a molecule from simple limits on inter-atomic distances). Additionally, [Cone77] has described a similar use of correspondence graphs in 2D comparisons of molecules but again only comparisons of pairs of molecules were used.

The approach adopted in this section to extending the method to n molecules (n>2) is to select one of the molecules and find its common substructures with each of the other (n-1) molecules in turn. If the selected molecule is of size m, then each common substructure can be represented as a set whose elements are taken from 1,..,m. Hence, the problem of finding substructures common to the n molecules becomes one of intersecting (n-1) sets (one coming from each of the (n-1) "groups" of common substructures). Unfortunately, a naive approach of generating all the possible intersections is likely to lead to problems because, if there are K common substructures in each group, then there are $K^{(n-1)}$ ways of intersecting the sets. One way to tackle this problem is to "grow" the subsets in common in an analogous way to Crandell and Smith's and Varkony et al.'s [Vark79] methods grow the common substructures. More specifically, for each group, i, the union of all the sets is taken so as to form a set UNION(i). A set INTERSECT is formed by intersecting all the UNION(i)'s and the common substructures of size one are all the elements of this set. The algorithm then proceeds by

1) Growing each substructure by adding an element of INTERSECT which is greater than every element of the substructure. If there are several possible elements from INTERSECT which can be added, then an enlarged substructure is produced for each of these.

2) Every enlarged substructure is tested to see whether it is contained in at least one set from each group. If it is not, it is eliminated.

3) Every element of INTERSECT is checked to see if it is still contained in a substructure. If it is not, then it is eliminated from INTERSECT and any of the sets it was contained in. Any sets whose size is less than or equal to the current substructure size are also eliminated and the algorithm goes back to step (1).

4.5 COMPARING THE TWO APPROACHES

4.5.1 Comparing Two Molecules

4.5.1.1 Methodology

Molecules were chosen from the Cambridge Crystallographic Database (CCDB), the atoms reordered and both the original molecule and its copy were distorted slightly (in a way analogous to that of Section 4.3.7). The four versions of Crandell and Smith's method and the clique finding approach using the unmodified clique detection algorithm of Bron and Kerbosch, were run on the molecules which were of sizes 14 (12 carbons, 1 oxygen, 1 nitrogen), 15 (10 carbons, 1 oxygen, 3 nitrogens, 1 chlorine) and 20 (13 carbons, 3 oxygens, 3 nitrogens, 1 sulphur), and the results are given in tables 4.10 to 4.12. It is understood from a referee's comments that the version of Crandell and Smith's algorithm used in [Cran83a, Cran83b] did incorporate the extra sorting stage and thus corresponds to version 3 of Crandell and Smith's algorithm in the tables. However, the two versions of the algorithm without the extra stage were considered so as to provide indications of the likely times for circumstances where unclustered distances were being used. These were suggested in [Cran83a] as a means of overcoming the difficulties associated with the clustered distances (see Section 4.5.1.2) but they make it much more difficult to sort the grown substructures.

Ideally, it would have been liked to use a molecule of size around 35 with a common substructure of size 14 or 15 but the clique finding approach took over forty minutes of cpu time whilst Crandell and Smith required too much storage for the Prime. This latter point also meant that it was not possible to use this algorithm when the common substructure was greater than size 15.

Instead, molecules were compared with the first 250 molecules in the CCDB and collections of molecules were formed which had substructures of size 7 or greater in common with the query molecule. (For reasons of convenience, the comparisons of inter-atomic distances when setting up the correspondence graph involved the actual distances rather than the "clustered distances" and an error tolerance of 0.15 A was used for two distances to be considered the same.) The algorithms then compared the query molecule with the other molecules in each collection and the times for these are given in tables 4.13 to 4.16 (for molecules of sizes 9 (8 carbons, 1 oxygen), 24 (24 carbons), 28 (27 carbons, 1 oxygen) and 15 (13 carbons, 2 oxygens)). The clustering method was also investigated in this comparison by using two versions of the clique finding algorithm, one using the actual distances and an error tolerance of 0.15 A and the other the distances clustered with an error limit of 0.09 A (as in [Cran83a]). The effect of varying the error limit when not using clustering is shown in table 4.17 when the molecule of size 23 from the molecule of size 9's collection is used.

4.5.1.2 Discussion Of The Results

The comparison of the distorted molecules of sizes 14 and 15 produced results in line with the discussion of Section 4.4.1 in that, for large common substructures, adding a stage to sort the substructures into order, and, to a much smaller (and less decisive) extent, discarding the amending of the distance tables stage led to a significant increase in speed. As the size of the largest common substructure decreased, the performances of the different versions of Crandell and Smith's algorithm became much closer together as well as getting closer to that of the clique finding approach.

The amending stage did sometimes lead to an improvement of the algorithm when there is no sorting stage. However, when this stage was present the amending stage led to an increase in the time taken (because with the sorting stage the comparison stage is nowhere near as time consuming, and so a reduction in the number of structures to be compared has relatively little effect -but the fewer substructures to be compared does lead to some saving in storage space). Reducing the common substructure size often led to an increase in the time taken by the amending stage because if a particular inter-atomic distance was present in a small number of substructures, more substructures had to be examined before it was found.

The tables for the comparisons of the collections of molecules also indicate that the clique finding approach (using clustering) is faster than the various versions of Crandell and Smith. However, the speeds of all of the programs are reasonably close together as the largest common substructure sizes are small. However, tables 4.13 to 4.16 do show a serious weakness in the clustering technique in that clusters can be produced which contain inter-atomic distances whose difference can be relatively large. Hence, the large number of common substructures which are found using the clustering method and this leads to:-

1) Difficulties for the algorithms, with Crandell and Smith running out of storage space and the clique finding method taking much longer to finish.

2) Even if the algorithms terminate, in a practical situation the common substructures which they output have to be evaluated for their significance.

Unfortunately, despite the shortcomings of the clustering approach, some way has to be used with Crandell and Smith's algorithm for converting the inter-atomic distances into integers if the "grown" substructures are to be compared with each other efficiently.

On the other hand, Crandell and Smith's algorithm should be better at dealing with molecules of size 40 say, which have a small common substructure because the correspondence graph for the clique finding approach then becomes very large. However, in the examples which were tried, the clustering caused there to be a large number of common substructures and this led in turn to a storage space problem.

4.5.2 Comparing More Than Two Molecules

The method described in Section 4.4.2 was used to extend the clique finding approach to be able to deal with more than two molecules and it was compared with the two versions of Crandell and Smith's algorithm which use the extra sorting. The results are shown in tables 4.18 to 4.26 with the times taken by the INTERSECT/UNION stage of the clique finding approach being given in brackets, and with some of the runs being repeated using different distortions of the original molecules. The order of the molecules affects the clique finding approach because it compares the first molecule with each of the others in turn, and so tables 4.18 to 4.23 give the maximum and minimum times for the clique finding approach (as well as the maximum and minimum times for the INTERSECT/UNION stage). However, because the maximum and minimum times in these tables are fairly close to each other, when more than four molecules were being compared only one order was tested.

The results show that generally the (n-1) comparisons of pairs of molecules take more time than the INTERSECT/UNION stage of the clique finding method (except when the common substructure is very large). This was even true where the comparison of pairs of molecules produced very large numbers of cliques of size 5 or greater; for instance the clustering method applied to the comparison of six molecules produced more than 200 such cliques in the comparison of some pairs of molecules in table 4.24. Therefore the clique finding approach is likely to be roughly linearly related to the number of molecules being considered.

Overall, the clique finding approach was again superior to Crandell and Smith's algorithm which ran into storage problems.

4.6 EXTENDING THE MAXIMUM COMMON SUBGRAPH ALGORITHM TO LARGER MOLECULES

Unfortunately, the performance of the clique finding approach runs into difficulties very rapidly as the molecules under consideration become larger than about 35 atoms. If two molecules of greater size are compared then there are problems over storing the resulting correspondence graph and the clique finding algorithm becomes much slower reflecting the exponential nature of finding all cliques in a graph. In an attempt to get around this problem, [Bol179, Bol182] has suggested three different techniques (although the problem under consideration was to match a machine part against a template) :-

1) Reducing the number of features in the template by removing those which contribute little to the determination of the orientation of the part.

2) Reducing the number of features to those within a specified distance of an "important" feature.

3) Screening out irrelevant features by applying the clique finding approach to groups of features, eliminating the groups which do not occur in large enough cliques and then applying the clique finding method to the features in the surviving groups.

Suggestions 1 and 2 are not suitable for comparing two molecules where there is no a priori information about the common substructure being looked for, and so only method 3 was investigated. The implemented algorithm groups together atoms of the same atomic types from the second molecule, with the size of these groups being variable. All pairs whose first element is an atom from the first molecule and whose second element is a group from the other molecule such that both elements are of the same atomic type, are formed. Two pairs are connected in the correspondence graph if one of the inter-atomic distances between atoms from the two groups corresponds to the distance between the atoms from the first molecule. Pairs which feature in cliques larger than a given size, are marked and at the end of the clique finding stage, all combinations of the atom from molecule one and one atom from the group are formed for each marked pair. These then go to make up the nodes of the second correspondence graph.

The results of running this algorithm on two structures (one of size 63 atoms (45 carbons, 18 oxygens), the other of size 67 (40 carbons, 27 oxygens)), are given in tables 4.27 and 4.28 where the size of the groups and the size of the relevant cliques were varied. Because of storage considerations, a cut off of 1000 was placed on the number of nodes allowed in a graph.

Using a tolerance of 0.09 A for considering two inter-atomic distances to be equivalent, three common substructures of size 8 were found. So as to make the molecules more similar, 7 atoms (4 carbons, 3 oxygens) from the molecule of size 67 were added to the other molecule and the results of running the algorithm on the new structures are shown in tables 4.29 and 4.30.

The examples of the use of the algorithm only provide a brief look at its use with it clearly being possible to have a multiple stage "screening" system, however the times taken are much greater than those reported earlier in this chapter when small molecules were compared. This is mainly caused by the clique finding algorithm taking longer owing to the increased density of the graphs. Whilst the problem is a difficult one because of the inherent combinatorial explosion, cpu times of the order of five minutes are unlikely to be acceptable and additionally large common substructures cannot be dealt with because the method will not be able to eliminate enough pairings. Therefore it seems likely that the best method for dealing with very small common substructures is that of Crandell and Smith with some form of modified way of comparing distances, while comparing molecules with large common substructures could possibly be done by using some form of set reduction algorithm and eliminating all atoms which do not have at least the cut off number of neighbours.

4.7 OVERVIEW

Two methods for finding common 3D substructures between two molecules have been compared (when the substructures were of size 6 or larger) and the clique finding approach has been found to be considerably quicker than Crandell and Smith's algorithm. Various different clique detecting algorithms were compared in this context, and the widely used algorithm of Bron and Kerbosch was found to be considerably superior to the others. However, finding cliques in graphs of sizes greater than 1000 becomes very demanding in terms of storage and time. Therefore the algorithm of Crandell and Smith could be better if the molecules are of size, say, 40 with a small common substructure. Unfortunately, the clustering method used in conjunction with the algorithm leads to distances being considered equivalent even though their difference is quite large, and this in turn led to problems in trying to find such an example. The problem with the clustering method was another advantage of the clique finding approach in that this initial stage was no longer necessary.

The clique finding approach was extended to deal with more than two molecules and the results of the tests which were carried out indicate that the time taken is generally proportional to the number of molecules being considered. The clique finding algorithm was also extended to try to deal more effectively with large molecules but with very little success.

However, several riders must be applied to the above work. Firstly, as it stands, the algorithm is only likely to find the rings in a structure, and so some way of representing a ring by using two points in space could well be needed -as it is assumed that this bias towards ring detection is undesirable. This leads on to another drawback in that reported pharmacophores [Watt84] have tended to be small, and so listing large common regions does not guarantee that a small common pharmacophore will be found (though the pattern might be contained in a larger common region). Another very serious qualification to the above work is that it has only dealt with molecules' rigid conformations whereas a molecule is quite likely to have a fair degree of flexibility (see Section 2.1). Finally, in a similar way to the last chapter, the structures chosen for the test runs were very artificial.

In spite of its poor performance in the tests in this chapter, the version of Crandell and Smith's algorithm with no sorting stage is interesting in that it consists of a large number of independent computations. Therefore it is a prime candidate for implementation on parallel hardware and this will be considered in the next two chapters; while the clique finding approach will be met again in Chapter 8 which describes a system for finding molecules in the Cambridge Crystallographic Database similar 3D structurally to a query molecule.

KEY FOR TABLES 4.1 TO 4.5 :-

* The times are in cpu seconds	
B and K 1	is the original version of Bron and Kerbosch
B and K 2	is the modified version of Bron and Kerbosch
G and R	is Golender and Rozenblit
G and L	is Gerhards and Lindenberg
L and T	is Loukakis and Tsouros
L	is Loukakis

	Largest Clique Size And Number Of Cliques > Size 4					
	8	10	11	14	15	16
Algorithm	190	241	257	121	138	158
B and K 1	2.5	2.6	2.8	1.6	1.7	1.7
B and K 2	2.6	2.7	2.9	1.7	1.8	1.8
G and R	7.7	8.4	9.4	3.0	3.6	3.8
G and L	11.4	13.7	16.5	17.1	28.8	31.7
L and T	22.4	23.0	24.5	12.7	13.5	14.5
Loukakis	23.0	24.0	25.4	12.7	13.5	13.8

Table 4.1 Times(*) For Finding All Cliques In The Correspondence Graph Of The Molecules Of Size 16

	Largest Clique Size And Number Of Cliques > Size 4						
	6	8	10	13	15	17	19
Algorithm	4	6	12	11	12	15	12
B and K 1	0.7	0.8	0.8	0.8	0.8	0.8	0.8
B and K 2	0.7	0.7	0.7	0.7	0.7	0.7	0.7
G and R	0.6	0.6	0.7	0.7	0.8	0.9	1.0
G and L	1.3	1.3	1.5	1.6	1.6	1.5	1.5
L and T	3.6	3.7	3.9	4.0	4.0	3.9	4.1
Loukakis	3.3	3.6	3.8	3.8	3.7	3.6	3.6

Table 4.2 Times(*) For Finding All Cliques In The Correspondence Graph Of The Molecules Of Size 19

	Largest Clique Size And Number Of Cliques > Size 4		
	19	21	23
Algorithm	3654	632	593
B and K 1	20.1	9.8	10.6
B and K 2	19.5	9.5	10.2
G and R	134.3	30.8	30.7
L and T	281.9	114.5	116.8
Loukakis	273.2	116.1	120.1

Table 4.3 Times(*) For Finding All Cliques In The Correspondence Graph Of The Molecules Of Size 23

	Largest Clique Size And Number Of Cliques > Size 4						
	8	9	10	13	17	21	25
Algorithm	934	1293	1106	471	340	152	106
B and K 1	15.3	17.7	14.8	7.7	6.5	5.0	3.8
B and K 2	14.6	17.0	14.2	7.3	6.1	4.7	3.7
G and R	80.3	102.4	78.4	25.5	19.3	11.5	5.5
L and T	154.1	187.8	153.1	79.8	66.1	46.9	32.3
Loukakis	156.5	187.5	152.1	77.2	62.4	46.2	30.5

Table 4.4 Times(*) For Finding All Cliques In The Correspondence Graph Of The Molecules Of Size 25

	Largest Clique Size And Number Of Cliques > Size 4			
	12	14	19	24
Algorithm	16	18	49	142
B and K 1	8.5	8.5	9.0	9.9
B and K 2	8.3	8.3	8.7	9.6

Table 4.5 Times(*) For Finding All The Cliques In The Correspondence Graph Of The Molecules Of Size 24 Produced Without Clustering The Inter-Atomic Distances

Iteration Number	Number Of Growths Mol 1	Mol 2
1	16	16
2	111	113
3	450	472
4	1180	1195
5	2098	2212
6	2711	3023
7	2730	3081
8	2103	2246
9	1213	1266
10	506	517
11	144	145
12	25	25
13	2	2

Table 4.6 The Number Of Grown Substructures After Step 3 Of Crandell And Smith's Algorithm When Applied To The Example Of Section 4.5.1

Step	Time Taken
Initialise	0.3
Grow	5
Name	86
Compare	916
Amend	22
Total	1029

(a)

Step	Time Taken
Initialise	0.3
Grow	5
Name	90
Compare	20
Amend	31
Extra Sort	68
Total	479

(b)

Table 4.7 The Times Taken By The Various Steps Of Crandell And Smith's Algorithm When Applied To The Example Of Section 4.5.1

(a) Is the simple Crandell And Smith Algorithm
(b) Is the version where the grown substructures are sorted

The cpu times are in seconds. The discrepancies between the two "grow" and "name" steps can be attributed to the inaccuracy of the system clock and rounding errors.

Iteration Number	Number Of Growths Mol 1	Mol 2
1	16	16
2	111	113
3	450	472
4	1180	1195
5	2098	2212
6	2733	3032
7	2737	3082
8	2104	2330
9	1213	1302
10	506	526
11	144	146
12	25	25
13	2	2

Table 4.8 The Number Of Grown Substructures After Step 3 Of Crandell And Smith's Algorithm With No Amending Step When Applied To The Example Of Section 4.5.1

Step	Time Taken
Initialise	0.3
Grow	5
Name	89
Compare	889
Total	983

(a)

Step	Time Taken
Initialise	0.3
Grow	5
Name	91
Compare	20
Extra Sort	68
Total	184

(b)

Table 4.9 The Times Taken By The Various Steps Of Crandell And Smith's Algorithm With No Amending Step When Applied To The Example Of Section 4.5.1

(a) Is the simple Crandell And Smith Algorithm
(b) Is the version where the grown substructures are sorted

The cpu times are in seconds.

KEY FOR TABLES 4.10 TO 4.26 :-

* The times are in cpu seconds	
C and S 1	is the simple version of Crandell and Smith
C and S 2	is the simple version with no amending of the distance tables
C and S 3	is the version with sorting, and amending of the distance tables
C and S 4	is the version with sorting but no amending
Clique	is the clique finding approach using the original version of Bron and Kerbosch's algorithm (with distance clustering)
No Cluster	is the clique finding approach with no distance clustering
Cluster	is the clique finding approach with distance clustering
tm sr	too much storage required
10Min+	indicates that over 10 minutes of cpu time was used
41Min+	indicates that over 41 minutes of cpu time was used

	Size Of The Largest Common Substructure				
Algorithm	7	8	10	12	14
C and S 1	8.4	9.4	25.5	135.7	1388.7
C and S 2	8.4	9.4	24.4	137.5	1390.4
C and S 3	5.0	6.1	14.0	55.0	290.0
C and S 4	4.5	5.3	12.0	47.9	266.3
Clique	2.1	2.3	2.4	2.3	2.4

Table 4.10 The Times (*) Taken By The Maximal Common Substructure Algorithms Examining The Molecule Of Size 14

	Size Of The Largest Common Substructure					
Algorithm	6	7	9	11	13	15
C and S 1	4.5	5.9	9.0	42.0	318.3	41Min+
C and S 2	4.3	5.5	7.8	45.0	332.9	41Min+
C and S 3	3.1	3.9	6.5	21.7	80.7	409.5
C and S 4	2.6	3.2	4.7	19.1	70.6	382.0
Clique	1.5	1.5	1.5	1.5	1.4	1.4

Table 4.11 The Times (*) Taken By The Maximal Common Substructure Algorithms Examining The Molecule Of Size 15

	Size Of The Largest Common Substructure				
Algorithm	6	7	9	11	14
C and S 1	52.9	93.9	103.3	435.8	tm sr
C and S 2	51.0	94.1	121.7	497.6	tm sr
C and S 3	12.5	20.6	23.9	70.9	tm sr
C and S 4	8.6	14.6	19.0	59.8	tm sr
Clique	4.1	4.0	3.6	3.8	3.8

Table 4.12 The Times (*) Taken By The Maximal Common Substructure Algorithms Examining The Molecule Of Size 20

Algorithm	Clique Size	Molecule Size 17	22	23	29	36
No Clustering	5	10	46	23	7	18
	6	10	22	20	22	20
	7	1	2	3	1	2
	8	1	3	2	2	0
	9	0	0	0	1	2
With Clustering	5	4	21	32	37	23
	6	10	8	44	32	27
	7	1	1	2	1	2
	8	1	3	6	4	4

Table 4.13(a) The Number Of Cliques With And Without Distance Clustering For The Molecules In The Collection Of The Molecule Of Size 9

	Molecule Size 17	22	23	29	36
No Cluster	1.2	2.2	2.1	2.4	2.6
Cluster	1.5	3.5	3.4	3.8	3.9
C and S 1	3.3	11.2	13.5	7.4	8.0
C and S 2	3.4	9.4	11.8	5.3	5.7
C and S 3	3.3	12.4	12.9	7.3	7.9
C and S 4	3.2	9.8	10.1	5.1	4.9

Table 4.13(b) Times (*) For Finding Common Substructures Between The Molecule Of Size 9 And Its Collection Of Molecules

Algorithm	Clique Size	Molecule Size 13	14	24	34
	5	27	70	109	181
No	6	28	64	132	118
Clustering	7	16	56	86	31
	8	2	10	28	7
	9	0	0	0	1
	10	0	0	0	6
	11	0	0	0	2
	12	0	4	8	0
	5	2952	3368	15298	--
	6	380	516	2670	--
	7	168	176	442	--
With	8	56	48	136	--
Clustering	9	16	24	32	--
	10	4	4	8	--
	11	0	0	0	--
	12	4	4	8	--

Table 4.14(a) The Number Of Cliques With And Without Distance Clustering For The Molecules In The Collection Of The Molecule Of Size 24

	Molecule Size 13	14	24	34
No Cluster	6.7	7.1	22.4	36.5
Cluster	28.4	29.4	109.9	40Min+
C and S 1	Too much storage required			
C and S 2	Too much storage required			
C and S 3	Too much storage required			
C and S 4	Too much storage required			

Table 4.14(b) Times (*) For Finding Common Substructures Between The Molecule Of Size 24 And Its Collection Of Molecules

Algorithm	Clique Size	Molecule Size 22	25	26	29	33
No Clustering	5	294	425	35	672	369
	6	60	93	6	214	72
	7	22	24	7	75	17
	8	6	13	0	34	2
	9	0	0	0	8	0
	10	0	0	0	7	0
	11	0	2	0	1	0
	12	0	0	0	5	0
	13	0	0	0	1	0
	14	0	0	0	0	0
	15	0	0	0	2	0
With Clustering	5	--	13616	14730	31855	--
	6	--	4815	4537	18432	--
	7	--	1027	661	8535	--
	8	--	161	65	4014	--
	9	--	27	18	1185	--
	10	--	16	2	290	--
	11	--	5	0	72	--
	12	--	4	0	25	--
	13	--	0	0	13	--
	14	--	0	0	22	--
	15	--	0	0	19	--
	16	--	0	0	11	--
	17	--	0	0	5	--

Table 4.15(a) The Number Of Cliques With And Without Distance Clustering For The Molecules In The Collection Of The Molecule Of Size 28

	Molecule Size 22	25	26	29	33
No Cluster	19.5	22.5	9.5	34.9	28.0
Cluster	10Min+	97.2	82.3	218.9	10Min+
C and S 1	Too much storage required				
C and S 2	Too much storage required				
C and S 3	Too much storage required				
C and S 4	Too much storage required				

Table 4.15(b) Times (*) For Finding Common Substructures Between The Molecule Of Size 28 And Its Collection Of Molecules

Algorithm	Clique Size	Molecule Size 20	21	22	23	36
No Clustering	5	34	1	3	30	111
	6	40	26	48	34	37
	7	21	4	11	22	2
	8	6	0	0	5	0
	9	3	0	0	2	8
	10	0	0	0	0	4
With Clustering	5	174	40	64	117	1533
	6	33	22	16	33	472
	7	46	8	32	36	164
	8	12	0	0	8	86
	9	0	0	0	0	32
	10	4	0	0	4	4
	11	0	0	0	0	2
	12	0	0	0	0	2

Table 4.16(a) The Number Of Cliques With And Without Distance Clustering For The Molecules In The Collection Of The Molecule Of Size 15

	Molecule Size 20	21	22	23	36
No Cluster	3.2	2.2	3.2	3.8	13.0
Cluster	5.7	3.9	5.6	6.4	36.5
C and S 1	39.2	11.5	10.6	49.6	tmsr
C and S 2	37.3	11.3	9.9	45.9	tmsr
C and S 3	27.9	9.0	9.3	28.4	tmsr
C and S 4	22.2	7.5	8.0	20.2	tmsr

Table 4.16(b) Times (*) For Finding Common Substructures Between The Molecule Of Size 15 And Its Collection Of Molecules

Number Of Cliques Of Size	Distance Error Tolerance (In Angstroms) 0.05	0.10	0.15	0.20	0.25	0.35	0.50
5	16	0	23	31	26	37	91
6	10	20	20	28	34	38	50
7	1	5	3	5	1	2	4
8	0	1	2	3	2	2	2
9	0	0	0	0	2	2	2
Time Taken (cpu secs.)	1.7	1.8	2.1	2.3	2.4	2.7	3.3

Table 4.17 The Effect Of Varying The Error Limit When Using The Clique Finding Method With No Distance Clustering When Comparing Two Molecules Of Sizes 9 And 23

Algorithm		Largest Common Substructure Size 8	9	12	14
Cluster:	Max	6.9 (2.5)	7.5 (2.9)	7.0 (3.3)	9.6 (5.5)
	Min	6.1 (1.8)	6.8 (2.2)	6.4 (2.7)	9.6 (5.5)
C And S 3		13.3	21.7	103.4	522.4
C And S 4		11.6	18.2	90.2	478.7

Table 4.18 Times(*) For Comparing Three Molecules Of Size 14

Algorithm		Largest Common Substructure Size 9	10	11	14
Cluster:	Max	5.2 (1.5)	5.4 (1.7)	6.1 (2.5)	9.6 (5.5)
	Min	4.9 (1.2)	4.3 (1.0)	5.3 (1.7)	9.6 (5.5)
C And S 4		19.1	26.6	48.4	478.7

Table 4.19 Times(*) For Comparing Three Molecules Of Size 14

Algorithm		Largest Common Substructure Size 8	10	12	14
Cluster:	Max	16.4 (9.0)	11.9 (5.9)	13.8 (8.1)	16.8 (11.5)
	Min	15.4 (8.0)	11.2 (5.4)	11.6 (6.0)	8.8 (9.2)
C And S 3		46.1	69.7	191.7	532.0
C And S 4		32.2	54.2	162.7	469.8

Table 4.20 Times(*) For Comparing Three Molecules Of Size 20

Algorithm		Largest Common Substructure Size 6	9	11	12
Cluster:	Max	6.1 (1.1)	7.3 (2.0)	9.2 (4.1)	9.0 (3.9)
	Min	5.9 (1.1)	7.0 (1.9)	8.8 (3.5)	8.8 (3.7)
C And S 4		16.5	36.1	72.8	103.4

Table 4.21 Times(*) For Comparing Three Molecules Of Size 20

Algorithm		Largest Common Substructure Size 7	8	9	12
Cluster:	Max	15.8 (5.1)	23.5 (10.8)	17.8 (7.3)	17.9 (8.3)
	Min	14.9 (4.5)	21.6 (8.7)	17.0 (6.7)	15.7 (5.9)
C And S 3		33.6	60.6	77.4	191.8
C And S 4		25.5	47.9	62.0	155.3

Table 4.22 Times(*) For Comparing Four Molecules Of Size 20

Algorithm		Largest Common Substructure Size 6	8	9	12
Cluster:	Max	23.0 (12.4)	17.6 (8.7)	16.0 (7.4)	17.7 (9.0)
	Min	22.3 (11.6)	16.9 (7.6)	15.3 (6.6)	16.6 (8.2)
C And S 4		25.9	50.6	60.6	198.5

Table 4.23 Times(*) For Comparing Four Molecules Of Size 20

Algorithm	Largest Common Substructure Size 8	11	14
Cluster	25.8 (8.0)	17.7 (4.3)	26.3 (13.9)
Crandell And Smith 3	116.4	159.9	TMSR
Crandell And Smith 4	79.1	112.5	TMSR

Table 4.24 Times(*) For Comparing Six Molecules Of Size 20

Algorithm	Largest Common Substructure Size 8	12	14
Cluster	34.0 (15.6)	22.5 (7.3)	28.2 (16.4)
Crandell And Smith 4	98.7	245.1	TMSR

Table 4.25 Times(*) For Comparing Six Molecules Of Size 20

Algorithm	Largest Common Substructure Size 7	9	11
Cluster	43.6 (14.0)	45.4 (15.0)	34.3 (11.0)
Crandell And Smith 4	209.0	206.5	TMSR

Table 4.26 Times(*) For Comparing Nine Molecules Of Size 20

Group Size	Minimum Clique Size	Number Of Nodes Graph1	Graph2	Time For Set Up	Clique1	Clique2	Total
2	ANY	1000+	***	***	*****	*****	*****
3	6	792	603	85	172	22	279
3	7	792	177	84	171	2	258
4	6	576	1000+	78	357	*****	*****
4	7	576	1000+	78	355	*****	*****
4	8	576	436	78	355	12	445
5	8	468	1000+	66	889	*****	*****

Table 4.27 Times(*) For Comparing The Two Molecules Of Section 4.6 Taking The Molecule Of Size 63 First

Group Size	Minimum Clique Size	Number Of Nodes Graph1	Graph2	Time For Set Up	Clique1	Clique2	Total
2	ANY	1000+	***	***	*****	*****	*****
3	6	762	645	76	161	26	264
3	7	762	207	75	161	3	240
4	6	615	1000+	73	302	*****	*****
4	7	615	1000+	71	303	*****	*****
4	8	615	368	71	303	9	383
5	8	468	1000+	66	675	*****	*****

Table 4.28 Times(*) For Comparing The Two Molecules Of Section 4.6 Taking The Molecule Of Size 67 First

Group Size	Minimum Clique Size	Number Of Nodes Graph1	Graph2	Time For Set Up	Clique1	Clique2	Total
2	ANY	1000+	***	***	*****	*****	*****
3	5	875	1000+	104	206	*****	*****
3	6	875	633	103	207	25	335
4	6	637	1000+	93	434	*****	*****
4	7	637	1000+	88	430	*****	*****

Table 4.29 Times(*) For Comparing The Altered Molecules Of Section 4.6 Taking The Molecule Of Size 70 First

Group Size	Minimum Clique Size	Number Of Nodes Graph1	Graph2	Time For Set Up	Clique1	Clique2	Total
2	ANY	1000+	***	***	*****	*****	*****
3	5	869	1000+	95	197	*****	*****
3	6	869	651	94	197	26	317
4	6	682	1000+	88	382	*****	*****
4	7	682	810	88	383	40	510

Table 4.30 Times(*) For Comparing The Altered Molecules Of Section 4.6 Taking The Molecule Of Size 67 First

CHAPTER 5

SIMULATING A MULTIPROCESSOR SYSTEM FOR FINDING THE LARGEST COMMON SUBSTRUCTURE

5.1 INTRODUCTION

Crandell and Smith's algorithm [Cran83a] for finding the largest common substructure of two or more molecules was considered in Chapter 4 and found to be very computationally expensive if the common substructure was greater than about eleven atoms. Additionally, it involves a great number of largely independent computations, and so it would appear that this algorithm could make efficient use of a multiprocessor system. Unfortunately, the more efficient version of the algorithm with an extra sorting stage is less suitable for a parallel implementation. This is because splitting up the arrays to be sorted into blocks and then sorting these blocks on separate processors before merging the results back together, has the problem that the elements of the arrays are lists. This means that the merge stage is likely to be relatively expensive as the lists being compared will be fairly similar. Additionally, it is less easy to devise an efficient parallelization of the new comparison stage. Thus, attention was restricted to the basic version of the algorithm without the extra sorting stage. The hypothesis that the algorithm should perform well on a parallel computer was investigated in two stages; in the first stage, which is described in this chapter, a simulation was undertaken to determine whether the potential for increases in efficiency did indeed exist. The results of this work led to the implementation of an operational system, and this system is described in the next chapter.

5.2 PARALLEL COMPUTER ARCHITECTURES

It is thought unlikely that (in the foreseeable future) improvements in the physical design of chips will lead to the large increases in the speed of computers that are required by many application areas [Kuck86]. This is in contrast to the past where improvements in silicon technology and VLSI led to much faster computers. Therefore the use of concurrency in computers is being intensively investigated [Hayn82, Hwan84, Zakh84].

In a conventional computer, a program counter steps through the code with a single (central) processing unit executing one instruction at a time. Major departures from this approach are:-

1) to have several processors to carry out an instruction (for example, inverting a matrix),

2) to have several functional units and to reject the idea of a program counter; instead instructions are carried out as soon as their input data are available,

3) to break the code up into blocks which are executed on separate processors which can communicate with each other,

and these will be considered in Sections 5.2.2, 5.2.3 and

5.2.4 respectively. First, however, less radical modifications to the above "von Neuman" computer will be considered.

5.2.1 Parallelism In "Conventional" Architectures

Since the 1960's (and computers such as CDC's 6600 and 7600) pipelining has been extensively used in conventional architectures as a way of obtaining a degree of parallelism. The idea of pipelining is to split up an operation such as a memory access into several stages and to enable another instruction to start stage one as soon as the original instruction has passed on to stage two. (The instructions being taken strictly sequentially from the compiled code.) By using multiple pipelined processing units, several arithmetic operations can be carried out at the same time (in the ideal case, an instruction could be assigned to an arithmetic unit every clock cycle, in contrast to the von Neuman approach where an operation such as division could take 18 clock cycles and no other arithmetic instructions could start until it had finished). However, some form of tagging has to be employed to ensure that code such as

```
X:=A*X
B:=B+X
```

is carried out correctly.

The use of associative memory can also be regarded in a loose sense as having a degree of parallelism. Each element in an associative memory block has two fields -the address field and the data field. When an address is presented to the associative memory, it is compared simultaneously with all the stored address fields and the data field of any match is read out. In conventional architectures it is often used for high speed cache buffers where the contents of recently accessed memory locations are stored, thus saving time if these locations are accessed again a few lines later in the program.

Vector processors such as the Crays and Cyber-205, have some arithmetical units and registers designed to handle vectors of numbers instead of individual elements. Because of the saving on overheads and the parallelism which can be introduced, a very high theoretical performance is possible. However, the performance can be very disappointing in practice because it is very hard to vectorise most problems sufficiently well. In the parallel computers described below an analogous problem is encountered, that of granularity. This is the amount of computation processes which are executing in parallel, undertake before communicating with each other.

5.2.2 Arrays Of Processors

Problems such as matrix triangularization or the forming of convolutions which are time consuming for a single processor, can be solved by arrays of very simple, identical "processors". [Kung82] gives the example of calculating the convolution {Y1,Y2,...} where

$$Yi=W1*Xi+W2*X(i+1)+...+Wk*X(i+k-1)$$

A possible linear array of processors for this problem is shown in figure 5.1 where each processor just forms the product of Win and Xin and passes it on to an adder whilst also outputting Xin.

The very simple nature of the processors (with them often consisting of only a handful of gates) means that large numbers can be contained on one chip. However, the problems that can be dealt with by using elementary processors are fairly rare. On the other hand, distributed array processors connect microprocessors together in arrays with each microprocessor carrying out the same instructions. An example of this type of architecture is the ICL Distributed Array Processor (DAP) [Gost81] which uses an array of 64*64 processors with each processor being connected to its four neighbours. Each processor is also connected to an ICL 2900 which can access the processor's memory and from which it receives the current instruction. The 2900 also has access to "conventional" memory and in effect it executes a section of program by carrying it out itself if the code is marked as being sequential (when the DAP is just used as ordinary memory) or by activating the DAP and just providing IO facilities if it is marked as parallel. Hence, for tasks such as searching where the memory to be searched can be broken up into small pieces which are distributed to the processors, a large amount of

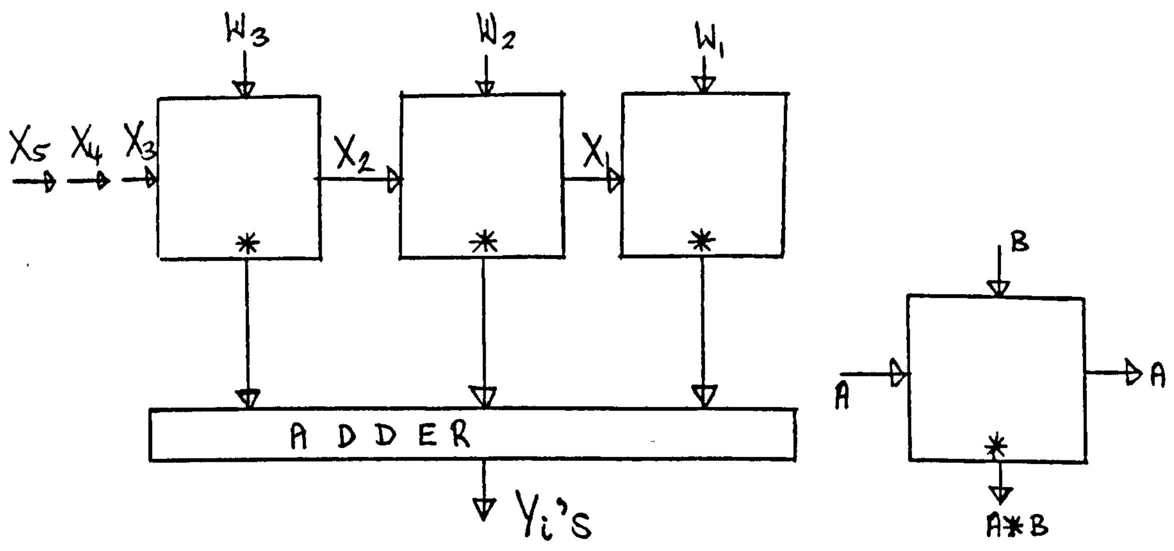


Figure 5.1 A Systolic Array For Calculating A Convolution

```
b::=x*a

x::=x+a

y::=x*a+b

z::=x+y

c::=a+b
```

Figure 5.2 The Code Used To Illustrate How Instructions Need Not Be Carried Out In A Strict Sequential Order

```
A:=B*C+D*E

I:=I+1

X:=A-E*I

I:=K+C

Y:=I/A
```

Figure 5.3 The Program Code For The Dataflow Diagram Of Figure 5.4

parallelism can be obtained.

5.2.3 Data Flow Computers

The idea behind data flow computers can be illustrated by the section of program in figure 5.2. Traditional computers progress through the code from top to bottom, and so they only carry out c:=a+b when the program counter reaches this instruction (or at least, the neighbourhood of this instruction). However, after the first instruction in the list a and b do not change, thus the line c:=a+b can be carried out as soon as b:=x*a is completed. Therefore, if there are several functional units (adders, multipliers, etc.), a degree of parallelism can be achieved by executing instructions such as c:=a+b as soon as the elements on the right hand side are available. Hence data flow computers execute instructions when all the data for the instructions is ready, rather than when the program counter of traditional computers reaches the instructions.

Several data flow computers have been built and the one which will be described here was produced at Manchester [Gurd85, Gurd86]. To understand how it works, consider the data flow diagram of figure 5.4 for the program section of figure 5.3. The criterion for any of the (circled) functions to start is only that the required input values have been calculated. This is achieved by labelling the output of a function with a destination symbol and then storing these output tokens (consisting of

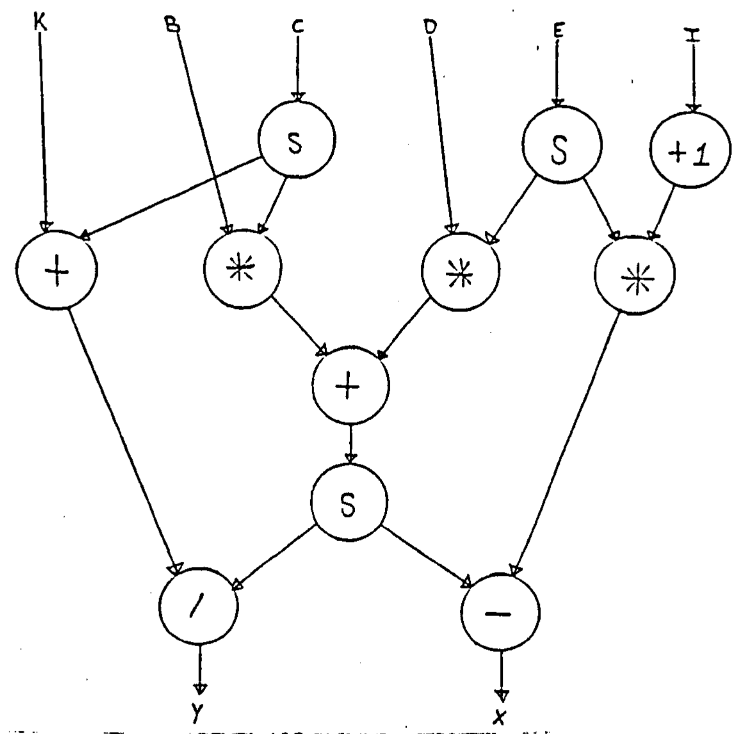


Figure 5.4 The Dataflow Diagram For The Code Of Figure 5.3

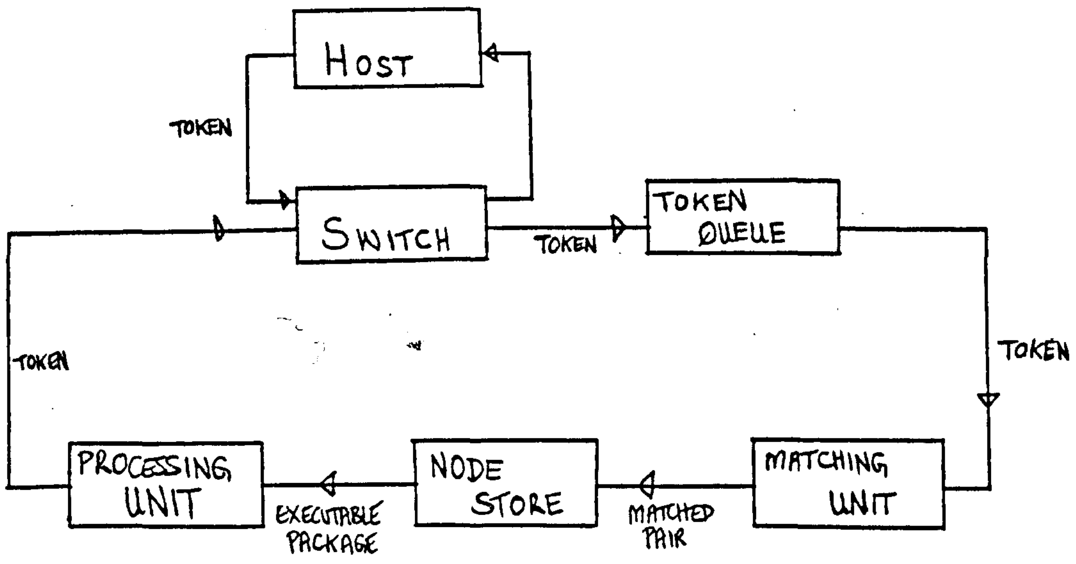


Figure 5.5 The Ring Used In The Manchester Dataflow Computer

label plus data) until the label can be matched with that from another token. The tokens are then passed on to the relevant functional unit as the input values have now been calculated. In this somewhat simplified view of things, it is necessary to have functional units (marked by S in the diagram) to make duplicate copies of data output from a functional unit which is used as an input to more than one functional unit.

Figure 5.5 is the basic layout of the ring of units in the Manchester computer and their basic functions are as follows:-

1) The Switch provides the interface for input and output operations (and allows several rings to be connected together for increased speed).

2) The Token Queue is a first in, first out queue which stores tokens when they are being produced by the processing unit faster than the matching unit can deal with them.

3) The Matching Unit tries to match the incoming token with a token which has the same label and destination field. If a match is found, the pair of tokens is sent to the node store. If no match is found, then the token is stored in the matching unit.

Ideally, the unit should be a very large associative memory, but as this would be extremely expensive, hardware hashing is used to simulate associative memory.

4) The Node Store really stores the program's "code". When

a matched pair of tokens arrives at the node store, the tokens' destination field is used to index into it and the relevant entry specifies the operation to be performed on the tokens and the destination field for the output of this operation. The executable package so produced, is sent to the processing unit.

5) The Processing Unit is made up of a group of processing elements -and it is the fact that many executable packages can be being processed at the same time which leads to the potential increase in speed over a conventional architecture.

Overall, data flow computers have the potential to very effectively exploit any concurrency in the code because of the "fine-grain" parallelism used. They also have the potential to be extremely powerful by connecting many rings together (via the switch). However, it is not clear [Gurd86, Hori86] whether the proposed increase in parallel activity outweighs the extra processing that the ring introduces or whether data flow architectures have any use beyond being single user scientific computers. Additionally, the preliminary figures of the Manchester prototype machine compared unfavourably (5 to 10 times slower) with a VAX 11/780, although a large part of this was thought to be due to the inefficiency of the generated code [Gurd85].

5.2.4 Multiple Independent Processors

Increasing interest has recently been shown in processors which carry out their own blocks of code but which can communicate with the other processors. This communication can be carried out by means of shared memory such as in the Cyba-M [Aspi84] (although memory contention can be a problem if there are a large number of processors), or by having communication channels associated with each processor such as INMOS' Transputer [Aspi84, Barr86] or Intel's iPSC series [Haye86] (where, because each processor is only connected to a small number of other processors, the data paths can become long and complex).

The individual processors can range from mini-supercomputers as in the Cedar supercomputer [Kuck86] which is trying to achieve a performance comparable with a Cray 2, to the iPSC and Transputer microprocessor chips. Where microprocessors are used, the performance is obtained by connecting large numbers of them together in a regular architecture (for instance Intel's hyper or cosmic cube [Seit85]). INMOS have developed the programming language occam [INMO84] which includes channel structures, so as to allow the maximum exploitation of the provided parallelism. However [Kuck86] feels that automatic restructuring of conventional program codes is likely to be more significant in the long term. A detailed description of the Transputer and of occam will be given in the next chapter.

5.3 MULTIPROCESSOR SIMULATIONS

The multiprocessor systems described in Section 5.2.4 have only recently become commercially available [Hock85], hence most of the work on multiprocessor algorithms has either been of a general abstract nature, for example [Wah85], or has involved the simulation of multiprocessor systems.

5.3.1 Chemical Simulations

In the chemical information area, Wipke and Rogers [Wipk84] simulated a system for subgraph matching. In the standard subgraph matching algorithm, various possible matches are produced for each atom in the pattern and the final stage in the algorithm tests the possible permutations of atoms using backtracking. Wipke and Rogers eliminated backtracking by splitting a process into several processes whenever a choice was possible for which atom of the structure under study was to be assigned to a given atom in the pattern. Consequently, a large number of non-interacting processes were created, thus enabling a simulated multiprocessor machine to be used very efficiently.

Gillet et al. [Gill86] have also used a similar simulation in the chemical field, but they were interested in generic substructure searching (although the actual simulation only used specific substructure searches) and a relaxation algorithm was used. Relaxation methods [Davi81, Pric85] independently assign possible labels to each point and possibly an associated probability, the provisional assignments for each point are then compared with assignments for nearby points and any inconsistencies are eliminated (or the probabilities adjusted). This comparison stage is then repeated as many times as is required. Hence some of the methods met earlier in this thesis, such as Ullman's subgraph isomorphism algorithm, can be regarded as belonging to the relaxation category as they only use information about neighbours. The parallelism was obtained by dividing some of the steps making up each iteration using the independence of the various assignments and comparisons. Although the amount of speed up predicted by the model varied greatly depending on the structures being examined, it averaged out at about a five-fold increase for a twenty processor machine. However the actual transputer implementation [Lync87] partitioned the database, and each structure-query matching process was carried out on a single transputer.

5.3.2 General Models

The two examples described above both assume that there is a pool of processors available any of which can be allocated to any process. Each processor has access to a region of memory shared with the other processors in addition to its own local memory. No account is taken of the architectural configuration of the processors, although the time to transfer data to and from the shared memory is estimated.

More complex models have been developed for dealing with general combinatorial problems (for example [McCo82a, McCo82b, McCa85]). These investigate such things as:-

1) Which pattern for the communication channels between processors is best,

2) Whether a busy processor should pass large or small problems to an idle neighbour

and

3) Whether an idle processor next to a busy processor should be detected by polling by the busy processor or by polling by the idle processor.

5.4 MODELLING THE CRANDELL AND SMITH ALGORITHM

5.4.1 PASSIM

The simulation was carried out using PASSIM [Shea82, Gill86, Stew87] which is a Pascal simulation system developed in the Division of Economic Studies at Sheffield University. This produces Pascal code from a description of a simple queueing system which has processes which take elements from one queue to another. The time which the processes take can be specified to be modelled by various probability distributions including the normal and negative exponential distributions. Two other simulations using PASSIM in the Department Of Information Studies have been reported [Gill86, Stew87].

A large proportion of the reported simulations, including [Wipk84, McCo82a, McCo82b, McCa85], have used the programming language SIMULA 67 [Birt73]. The reason for the development of PASSIM was to prevent the need for learning a new programming language by generating code in an already familiar language, namely Pascal. However, the facilities offered by PASSIM and SIMULA 67 are very similar, with the SIMULA system class SIMULATION corresponding to the PASSIM queueing structure. As both SIMULA 67 and Pascal were derived from Algol 60, the PASSIM-produced Pascal code allows similar facilities to the SIMULA 67 code of a model.

5.4.2 The Processor Organisation

It was decided to model a multiprocessor system similar to those in [Wipk84, Gill86] in that the processors communicated by using a region of shared memory. The large amount of extra effort needed to produce a more complex model was considered not to be worthwhile. This was because the study was intended to investigate whether it was likely that the algorithm would run efficiently on a multiprocessor system with a view to undertaking such an implementation if the answer was the affirmative.

[Aspi84] reports trials with the Cyba-M (which is a multiprocessor where the processors communicate with each other via shared memory) which indicate that even under adverse conditions the overall utilization (that is the overall percentage of time the processors are busy) for a 16 processor application was over 92% (as long as sufficient parallelism was present in the problem).

5.4.3 The Algorithm

The algorithm, which has been described in detail in Chapter 4, can be regarded as being composed of four steps. For each generation, the common substructures are found by:-

1) "Growing" the previous generation's substructures using the distance tables.

2) "Naming" the growths produced from step (1).

3) "Comparing" the named growths from each molecule with those from all the other molecules in order to eliminate the substructures which are not common to all of the molecules.

4) "Amending" the distance table of each molecule in order to eliminate the distance pairs which no longer occur in the set of compared growths.

5) Either stopping or increasing the generation number and returning to step (1).

The serial version of the algorithm for two molecules is shown in figure 5.6. As the only step which requires interaction with the other molecule's growths or distance table, is the compare stage, the algorithm can be written in the parallel form shown in figure 5.7 (where

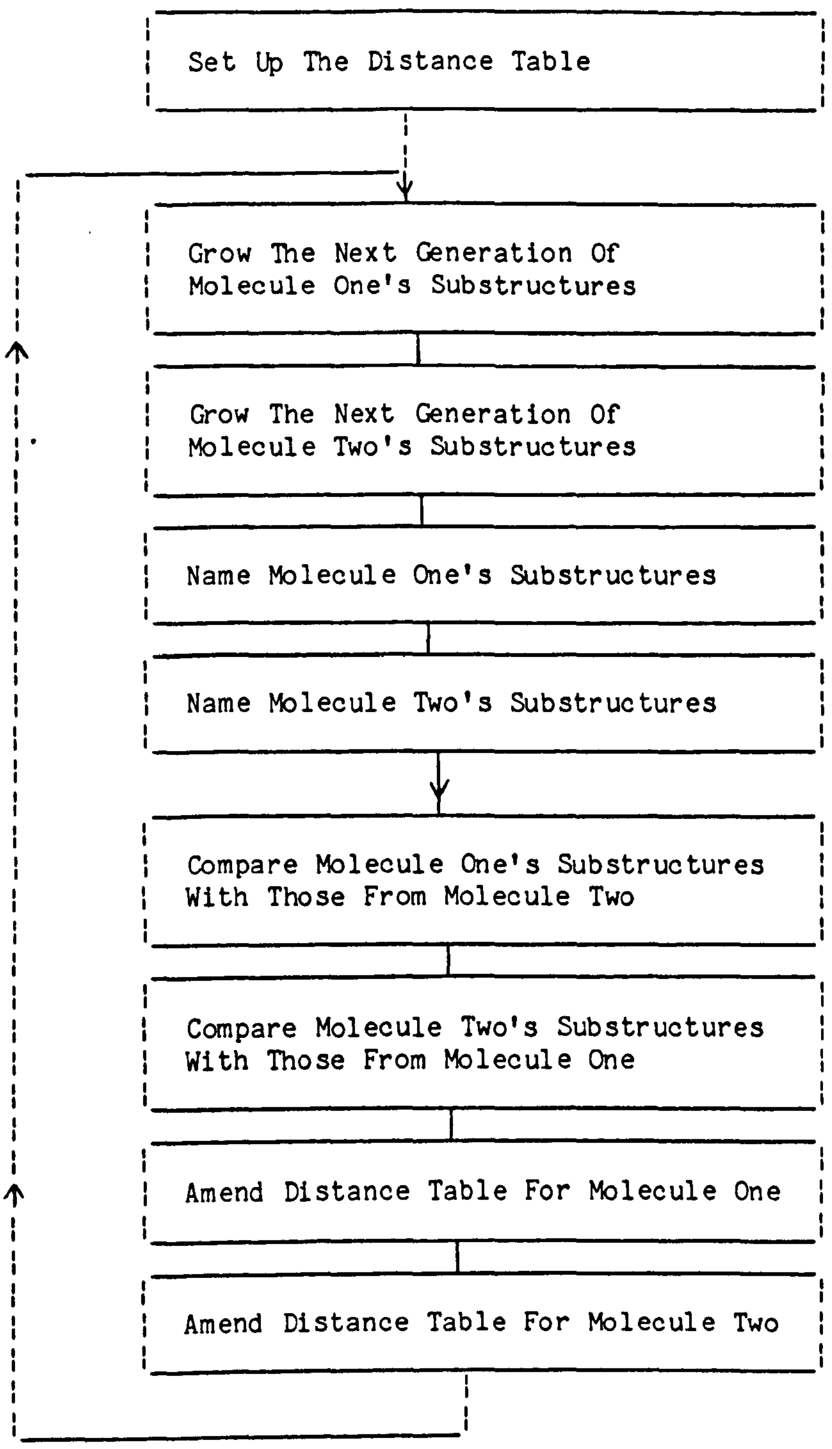

Set Up The Distance Table
Grow The Next Generation Of Molecule One's Substructures
Grow The Next Generation Of Molecule Two's Substructures
Name Molecule One's Substructures
Name Molecule Two's Substructures
Compare Molecule One's Substructures With Those From Molecule Two
Compare Molecule Two's Substructures With Those From Molecule One
Amend Distance Table For Molecule One
Amend Distance Table For Molecule Two

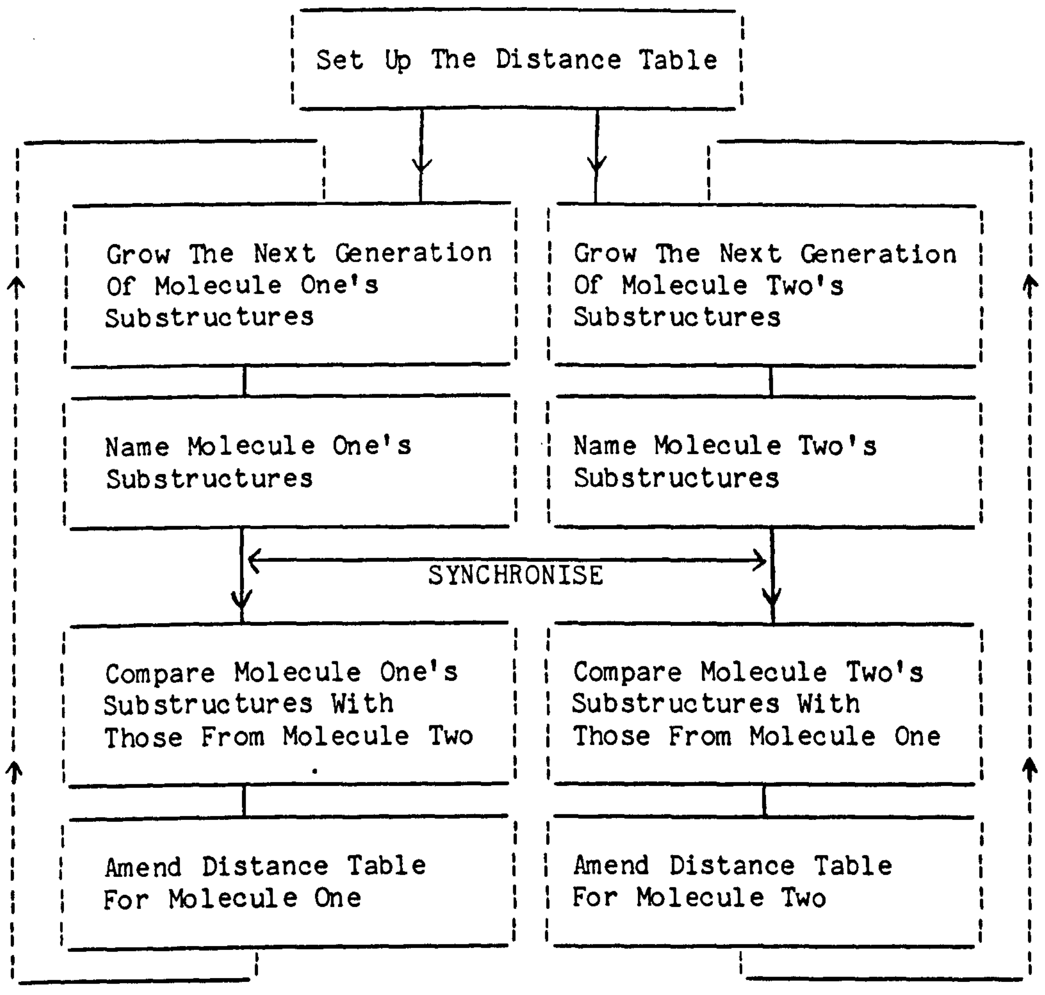
Figure 5.6 The Serial Form Of Crandell And Smith's Algorithm For Comparing Two Molecules

neither comparison stage can start until the other molecule's cycle has finished the naming stage).

5.4.4 The PASSIM Model Of The Algorithm

Each box in figure 5.7 represents a PASSIM process. In fact, with the exception of the initialization box, each of the boxes represents 50 identical PASSIM processes, thus allowing up to 50 processors to be working on the same stage at the same time. Each process takes a processor from the processor queue when it starts and returns it to this queue when it finishes. The initial size of the processor queue is one of the parameters of each simulation run (and represents the number of processors in the multiprocessor machine). This is all achieved in the PASSIM-generated Pascal code by trying to start a process. When a process is found which can start (ie. there are elements waiting in all the queues it uses), its finishing time is stored in a table. When no more processes can start, the "clock" is moved on to the earliest finishing time in the table and any processes which can finish are ended (and the elements they produce returned to the relevant queues). The procedure of trying to start a process is then repeated.

Some of the processes also take elements from various "growth" queues when they start. Hence the sizes of these queues at the start of each generation determine how many "jobs" each stage is to be split up into (that is how



Set Up The Distance Table
Grow The Next Generation Of Molecule One's Substructures
Grow The Next Generation Of Molecule Two's Substructures
Name Molecule One's Substructures
Name Molecule Two's Substructures
SYNCHRONISE
Compare Molecule One's Substructures With Those From Molecule Two
Compare Molecule Two's Substructures With Those From Molecule One
Amend Distance Table For Molecule One
Amend Distance Table For Molecule Two

Figure 5.7 The Parallel Form Of Crandell And Smith's Algorithm For Comparing Two Molecules

many separate procedures each of the original procedures is divided into), and they are specified by the user.

Each process can only start when there are elements waiting in the relevant queues. Additionally, by extra code having been added to the Pascal generated from the PASSIM model, the correct sequence of steps (in figure 5.7) is obeyed and the compare step of one molecule cannot start until the other molecule is at the same stage.

5.4.5 The Duration Of The Processes

The c.p.u. time taken for each step of each generation was found by running a FORTRAN 77 version of the serial algorithm on a Prime 9950. However the time taken for each molecule's comparison stage depends on how many growths there are belonging to the other molecule and each molecule's comparison stage reduces the number of growths associated with this molecule. Hence, the time taken for the comparison stage of molecule A when this is done before the comparison stage for molecule B will be greater than or equal to the time for this stage when it is done after B's comparison stage. Therefore the program was run with both orders of the molecules and the larger times for a molecule's comparison stage were used.

To obtain the times for the procedures each stage was split up into, it was assumed that each stage was made up of a large number of small processes whose durations formed a normal distribution. This assumption was made so that sampling theory for normal distributions could be used to obtain the times for the procedures each stage was split up into in the simulation. The values obtained from the runs were used to specify the mean duration times of the stages in the model (ie. the sum of the durations of all the small processes). The standard deviations for comparing one growth with the growths from the other molecule were estimated to be the means divided by root three. This was because:-

If we assume that

1) All growths are matched with a growth in the other molecule's growth list. (An approximation which was largely true for the pairs of molecules that the test was run on.)

2) That this matching is a one to one correspondence.

and

3) That the time taken to compare any growth with the other molecule's growth list is equal to the position in the list of the growth that it matches (or at least proportional to it).

Then, for a particular molecule's comparison stage,

the variance, $v=s^2$, is $\left(\sum_{i=1}^{N} (i-m)^2 \right)/N$

where N is the number of growths in one of the molecules' growth lists (both lists are the same size from assumption (2)) and m and s are the mean and the standard deviation for this comparison stage.

$$= (\sum_{i=1}^{N} i^2)/N - 2m^2 + m^2$$

$$= (\sum_{i=1}^{N} i^2)/N - m^2$$

But

$$\sum_{i=1}^{N} i^2 = N*(2*N+1)*(N+1)/6$$

So

$$s^2 = (2*N+1)*(N+1)/6 - m^2$$

But

$$m = (\sum_{i=1}^{N} i)/N = (N+1)*N / (2*N)$$

$$= (N+1)/2$$

Therefore

$$s^2 = m*(2*N+1)/3 - m^2$$

$$= m*(4*m-1)/3 -m^2$$

$$= m*(m-1)/3$$

So, as most of the c.p.u. time is consumed when N is very large, we have that the standard deviation, s, becomes approximately m/SQRT(3). (The observation that we are mainly interested in large N to some extent justifies the initial three assumptions.)

Unfortunately, the assumptions needed to undertake a similar analysis for the growing, naming and amending stages were felt to be more dubious. Therefore the standard deviations were estimated by splitting the relevant stages up into tenths (or, in the case of the amending stage, splitting up the calculation into pieces each of which was associated with a different distance table entry) and running the serial algorithm on a pair of molecules with a common substructure of size 14. The standard deviations were calculated (from these sets of 10 values) for the largest times for the growing, naming and amending stages. Finally, using the assumption that the standard deviations were linear functions of the means, the standard deviations for the growing, naming and amending steps were taken to be 0.19*m, 0.14*m and 1.78*m respectively. (This last assumption is likely to be approximately true for the amending stage due to its similarities with the comparison stage dealt with above but it is more suspect for the growing and naming stages. However, the times for these steps were very small when compared with the other two steps, and so this inexactitude is fairly inconsequential.) The problems associated with the probability distributions will be considered again in Section 5.6.1.

5.4.6 The Parameters Of The Model

Besides being able to alter the number of processors in the multiprocessor and the duration of each process (by changing the means and standard deviations) between each run of the simulation, it is also possible to vary

1) The data transfer rate between the shared memory and the processor's local memory.

2) The overhead incurred every time a task is allocated to a processor.

3) The number of jobs each box in figure 5.7 is split into.

In more detail, when a job is allocated to a processor, a variable representing the time for the relevant data transfers is added to the mean of the process' duration. Before certain stages of the algorithm, the PASSIM clock is moved forward to represent copying information to all of the processors at the same time, for example, the distribution of the growths before the comparison stage.

The processor overhead is a constant which is added to the mean of the process' duration for every process. This is a "rough and ready" attempt to account for the scheduling overheads in a multiprocessor system.

The partition factor (which is the number of jobs each step in the algorithm is split into) allows many processors to be working on the same step. The new means and variances of the durations are taken to be the old ones divided by the partition factor. This is an approximation as it does not take into account the fact that the sum of the times taken by all of the jobs should be equal to the old mean.

There are in fact two partition factors, one for the naming and growing stages, and the other for the amending stage and the more computationally expensive comparing stage.

5.5 THE INITIAL RESULTS OF THE SIMULATION

The simulation was run with the two molecules in the model being the same 14 (non-hydrogen) atom molecule. This data was chosen because, as the serial version of the Crandell and Smith algorithm took over 29 minutes of c.p.u. time on a Prime 9950, it was felt that this example would provide ample scope for parallel processing.

5.5.1 The Partition Factors And The Data Transfer Rate

With the data transfer rate set at 2 Mbytes/S and the processor overhead at zero, the number of processors and the partition factors were varied. The factor for the growing and naming stages was found to have very little effect, and so it was set to be equal to the number of processors, although this can lead to a poorer performance if the processor overhead is very large. The partition factor for the comparing and amending stages was found to have a quite large effect and it was decided to vary this parameter so as to achieve an optimal performance when conducting later runs.

Varying the data transfer rate between 0.2 and 10 Mbytes/S only altered the simulated time taken by less than 3%, and so, this parameter was set at the value 2 Mbytes/S which is a fairly typical rate for transferring data from a backing store (and well below the maximum data transfer rate of 25 Mbytes/S for off chip memory with the Transputer).

5.5.2 Varying The Standard Deviations

Altering the factor by which the standard deviations are obtained from the means for the growing, naming and amending stages from their usual values to those of 3, 6 and 9 had little effect on the simulated running time. However, altering the factor for the comparing stage had much more effect with the smaller the standard deviation the shorter the simulated time for completion (one of the reasons for this will be considered in Section 5.6).

5.5.3 Varying The Processor Overhead

For the first set of results the processor overhead was set to zero and the number of processors was varied between 1 and 50. Subsequent sets of results were obtained by setting the overhead equal to one, ten and one hundred times the number of processors. (The time being measured in thousandths of a c.p.u. second.) The final set was produced with the overhead having the value of 10 times the number of processors squared.

The simulated times produced from the above runs are given in table 5.1 and the corresponding graphs are figures 5.8 and 5.9.

5.5.4 Using Other Molecules

The above example is an extreme case in that the molecules were identical, and so the comparison stages were very expensive in terms of c.p.u. time used. Therefore the simulation was run on two other pairs of molecules, one where the molecules were of size 19 and the largest common substructure was of size 12, and another where the values were 14 and 9. The results of the runs (simulating a linear processor overhead) are given in tables 5.2 and 5.3 and figures 5.11 and 5.10 respectively.

Although in any multiprocessor problem the main point of interest is how long the computer took to solve it, figures 5.8 to 5.11 are of a reciprocal nature, and so are not that easy to interpret. Consequently, figure 5.12 was plotted (for the no processor overhead cases of the above examples) with the y values being the time taken when one processor was used divided by the time taken when the x co-ordinate number of processors was used (the co-ordinate values are given in table 5.4).

5.5.5 Comparing Three Molecules

The basic technique outlined in figure 5.7 can be extended to any number of molecules (the diagram for the

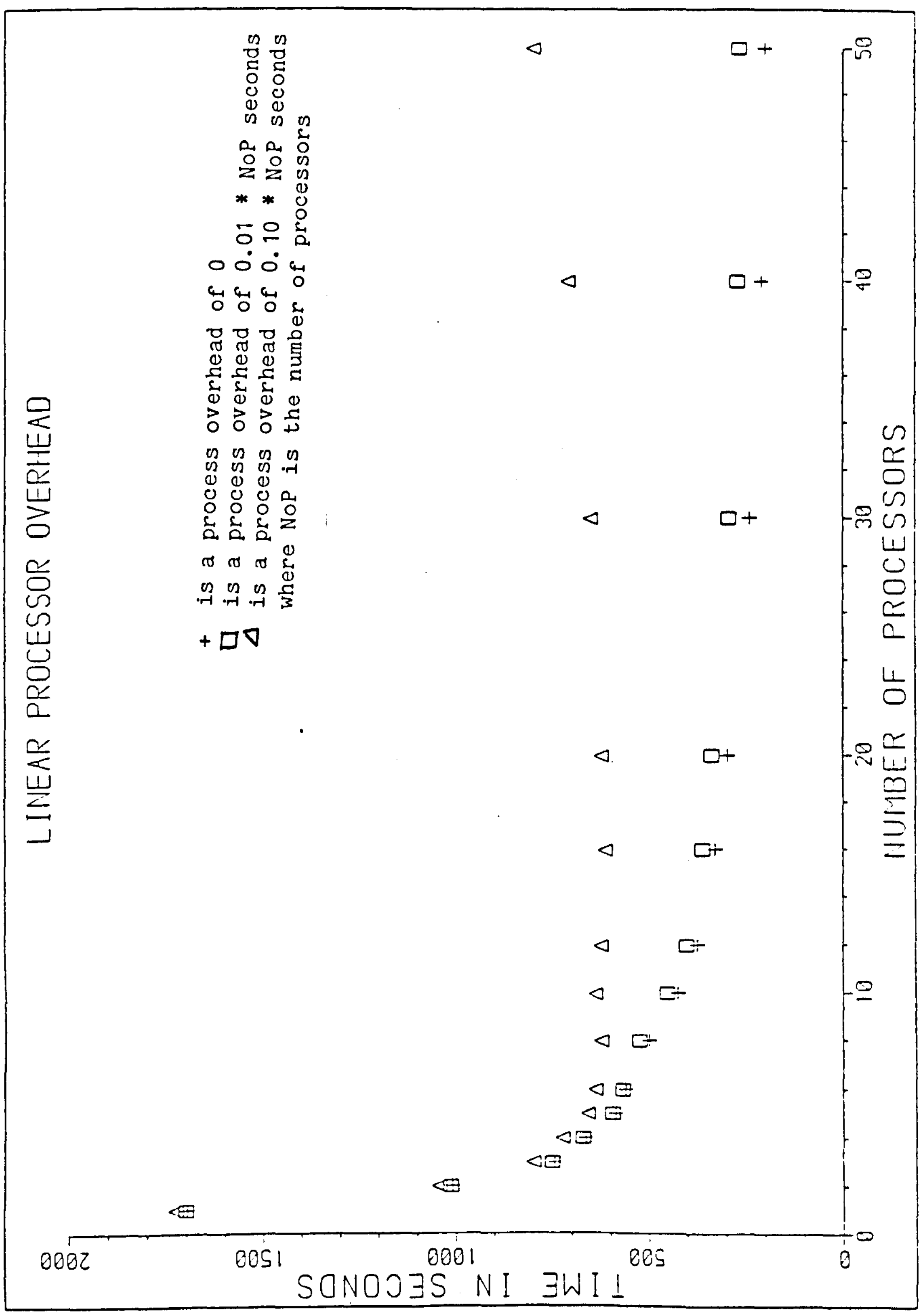


Figure 5.8 Comparing Two Molecules With A Largest Common Substructure Of Size 14 With A Linear Processor Overhead

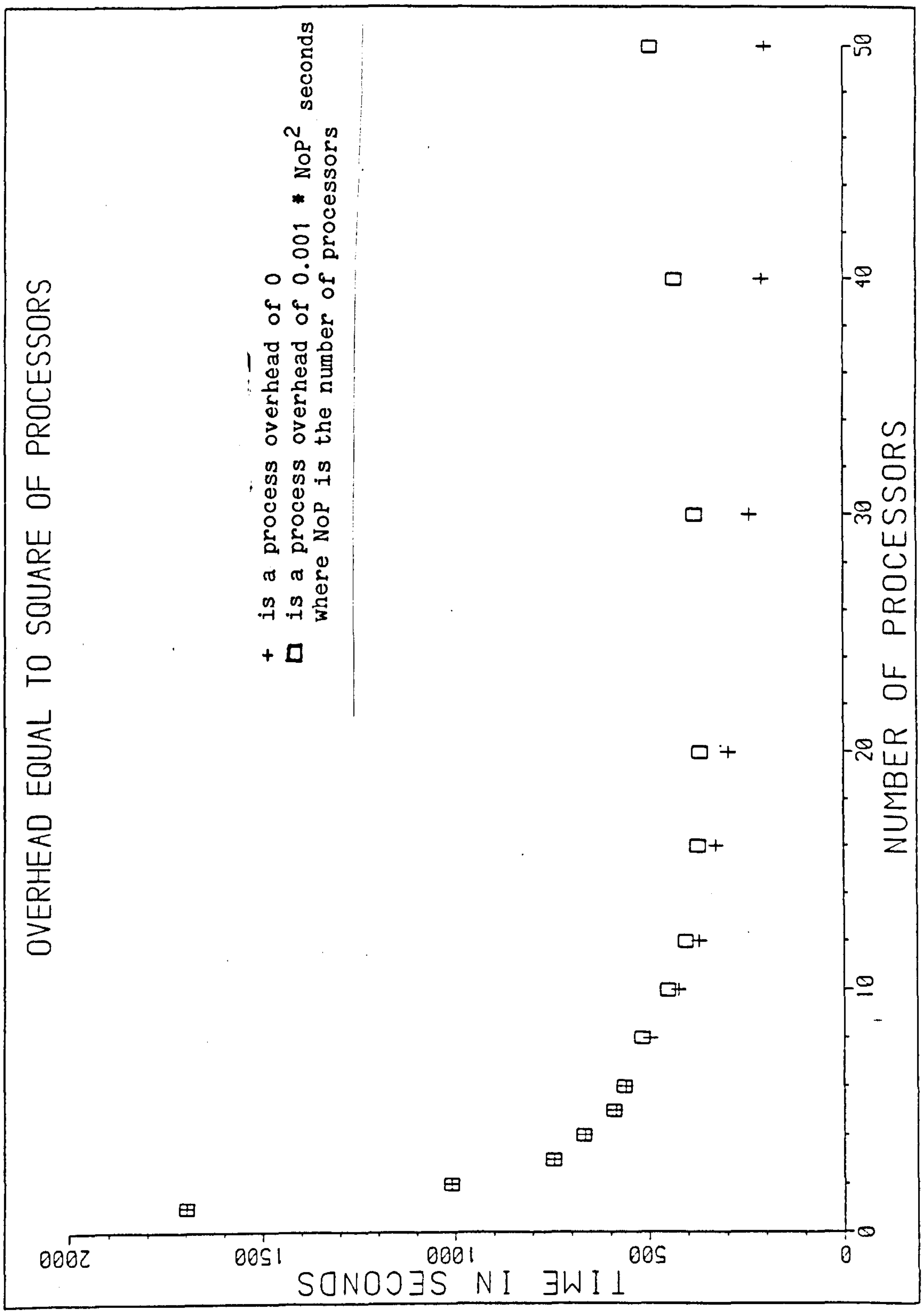


Figure 5.9 Comparing Two Molecules With A Largest Common Substructure Of Size 14 With The Processor Overhead Equal To The Square Of The Number Of Processors

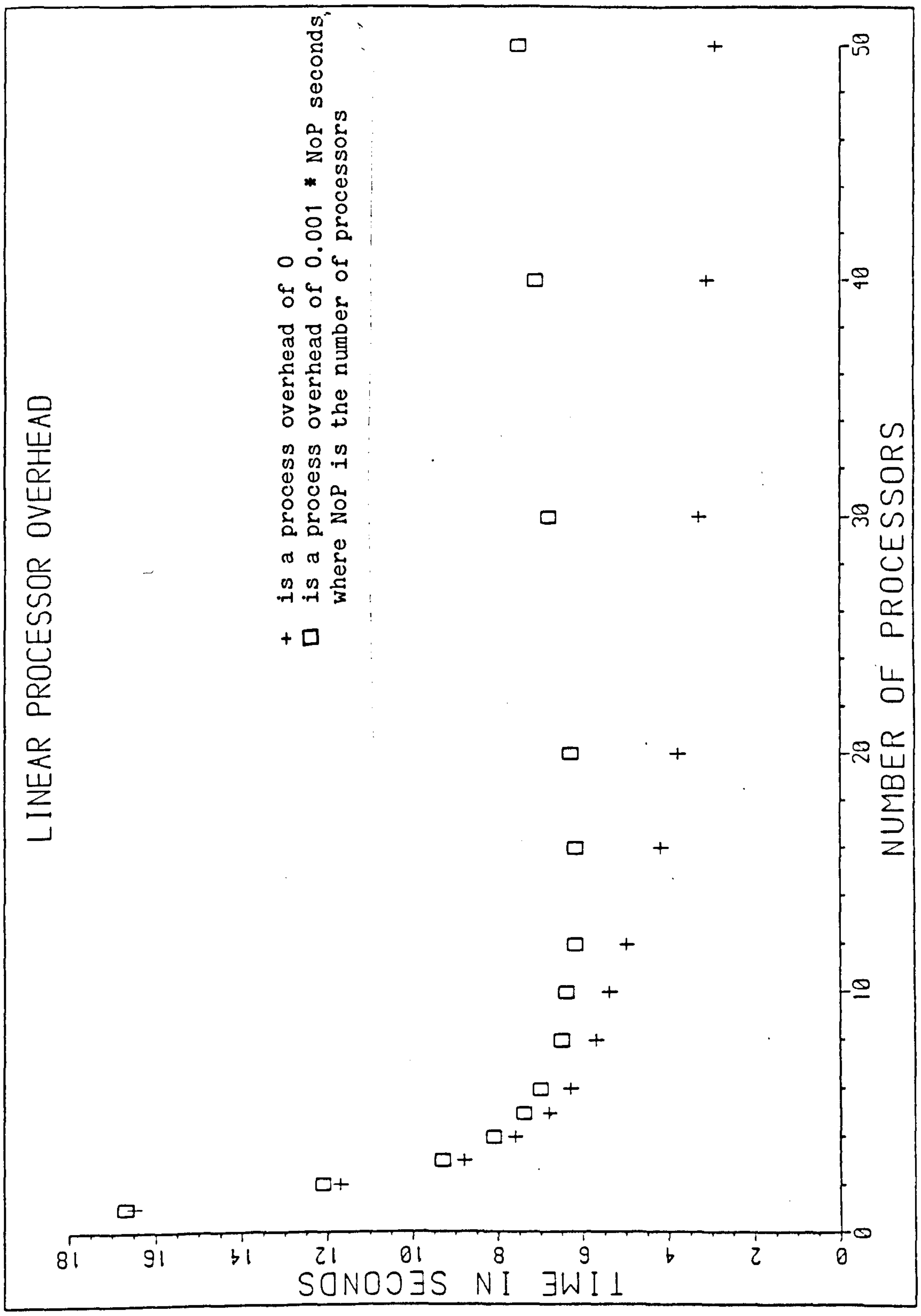


Figure 5.10 Comparing Two Molecules With A Largest Common Substructure Of Size 9

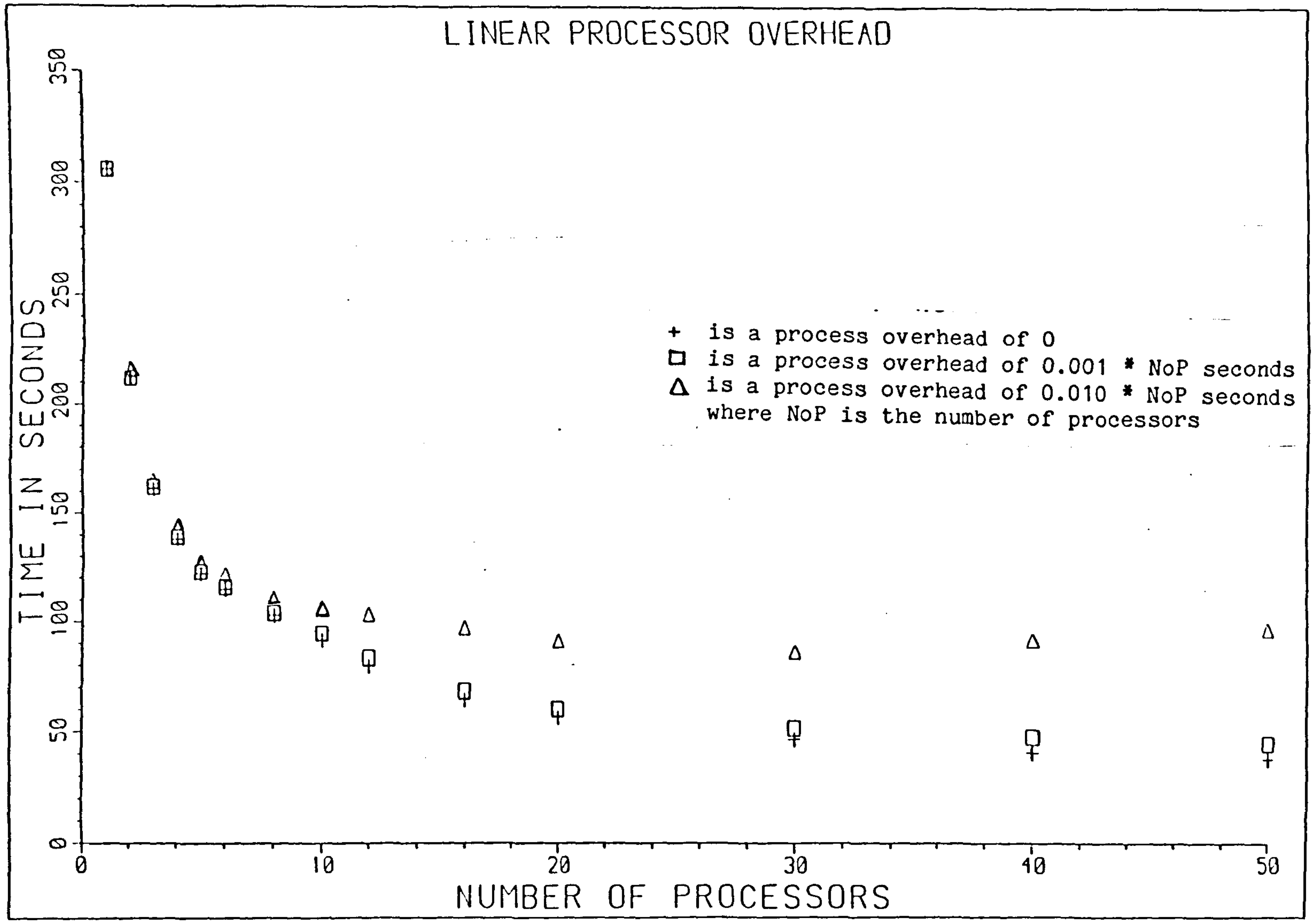


Figure 5.11 Comparing Two Molecules With A Largest Common Substructure Of Size 12

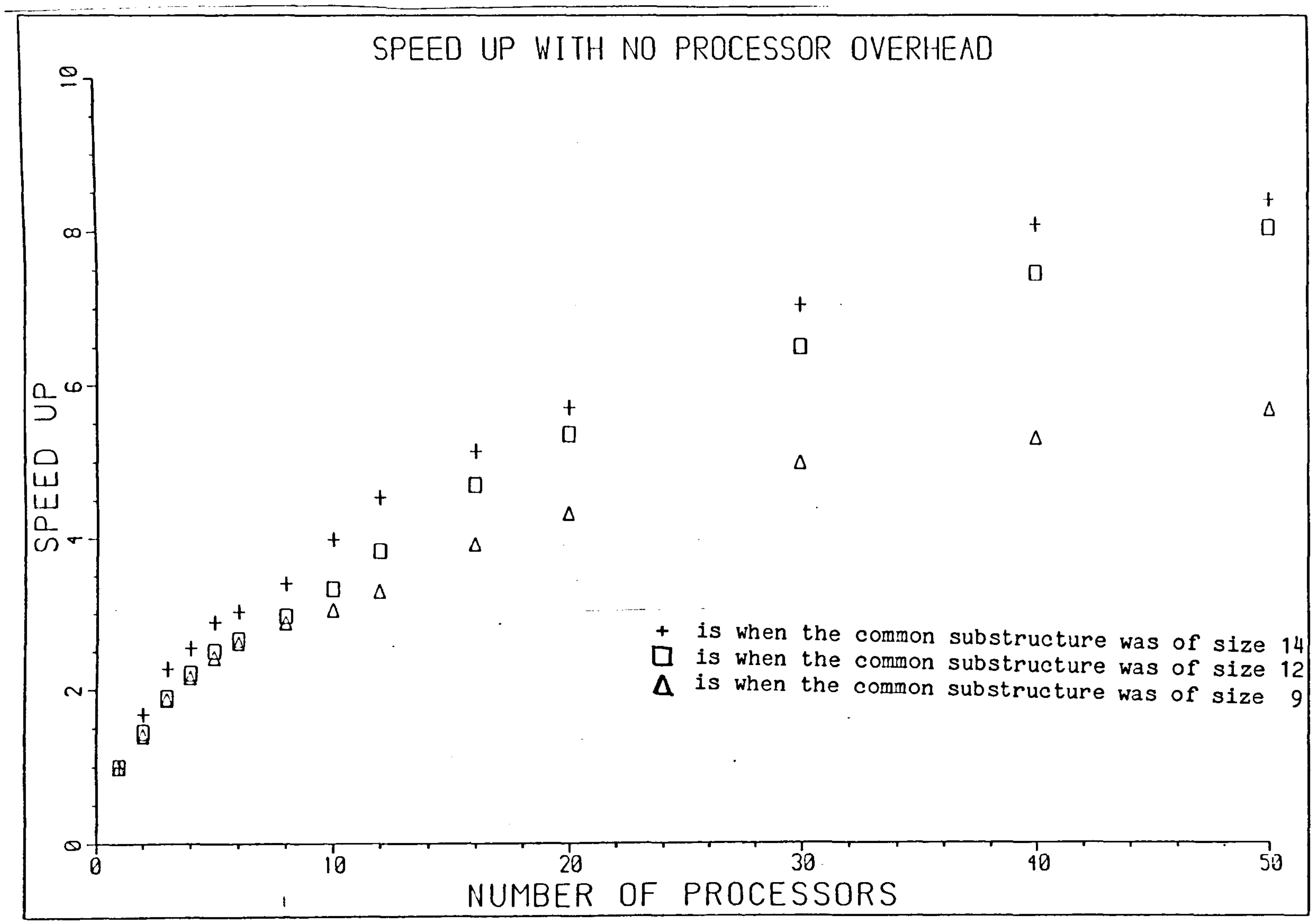


Figure 5.12 The Speed Up When Comparing Two Molecules With No Processor Overhead

three molecule case is given in figure 5.13). As an example of this, three molecules of size 14 and with a largest common substructure of size 11 were considered and the results of the simulation are given in table 5.5 and figure 5.14.

5.6 COMMENTS

5.6.1 Limitations

The principal problem area for the simulation was with the durations of the procedures formed by splitting up larger procedures. As the FORTRAN 77 program took over 29 minutes of c.p.u. time (on a Prime 9950) when there was a common substructure of size 14, it was clearly not possible to obtain these times experimentally. Therefore approximations to the standard deviations for the lengths of the larger procedures were obtained and assuming that the smaller procedures could be regarded as samples drawn from a normal distribution, the durations of the smaller procedures were estimated. The difficulties with this approach are:-

1) The assumption that the samples are drawn from a normal population (but the theory would be much more difficult if some other distribution was used).

2) The approximation to the standard deviations, with one of the problems being the assumption that the ratios between the standard deviations and the means of the four stages were the same for all molecules (although as Section

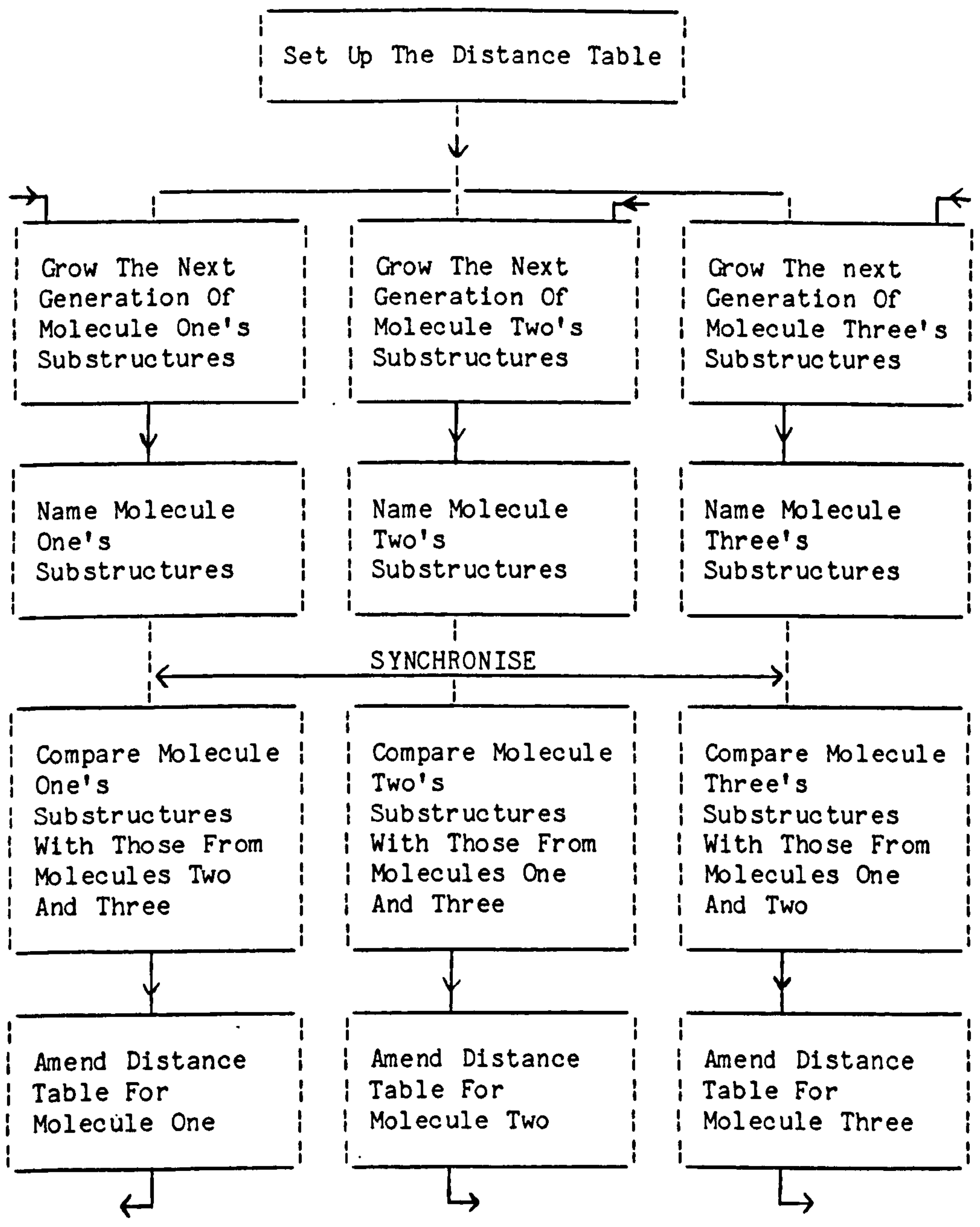

Set Up The Distance Table
Grow The Next Generation Of Molecule One's Substructures
Grow The Next Generation Of Molecule Two's Substructures
Grow The next Generation Of Molecule Three's Substructures
Name Molecule One's Substructures
Name Molecule Two's Substructures
Name Molecule Three's Substructures
SYNCHRONISE
Compare Molecule One's Substructures With Those From Molecules Two And Three
Compare Molecule Two's Substructures With Those From Molecules One And Three
Compare Molecule Three's Substructures With Those From Molecules One And Two
Amend Distance Table For Molecule One
Amend Distance Table For Molecule Two
Amend Distance Table For Molecule Three

Figure 5.13 The Parallel Form Of Crandell And Smith's Algorithm For Comparing Three Molecules

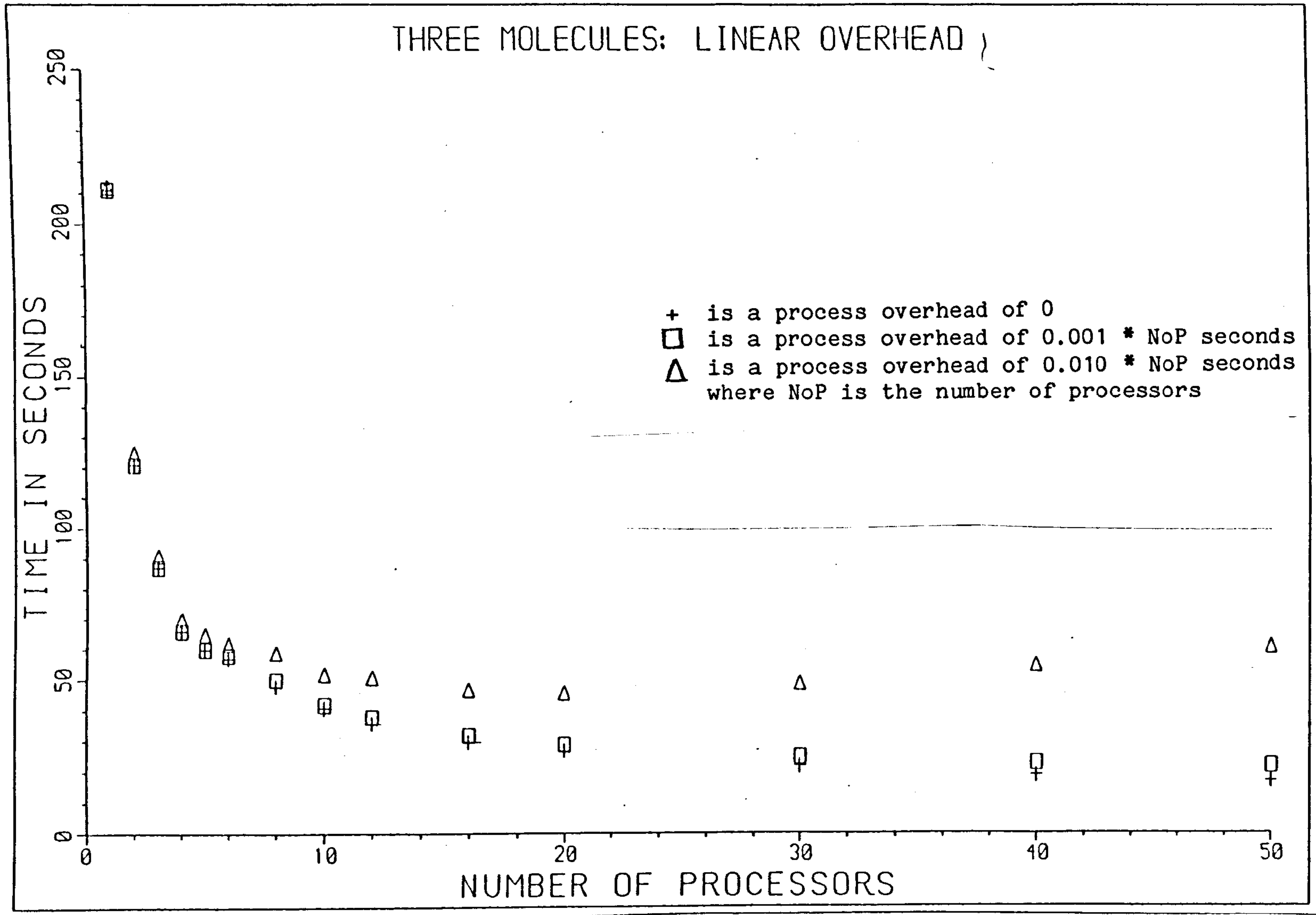


Figure 5.14 Comparing Three Molecules With A Largest Common Substructure Of Size 11

5.5.2 indicated the only significant standard deviation was that of the comparison stage).

3) The fact that the sum of the deviations of the smaller procedures is unlikely to be the duration of the larger process.

Another problem was that

4) The c.p.u. times used were taken from a large minicomputer.

5.6.2 Conclusions

The aim of the simulation was to provide a rough guide as to whether the version of Crandell and Smith's algorithm with no sorting stage is suitable for a multiprocessor system and to provide some indication of the speed up likely to be produced if such a system were to be implemented. Consequently, a compromise had to be made between ease of implementation and the various drawbacks mentioned in Section 5.6.1. (* See "Alterations" for missing sentences*)

After saying which, table 5.4 (along with figure 5.12) seems to indicate that using 50 processors can produce a speed up of about 8 and a speed up of around 5 for 16 processors if the problem is sufficiently computationally expensive (and assuming no processor overhead). However, table **5.6** (with figure 5.14) shows a speed up of over 12 when three molecules were being compared (which is in accordance with the extra parallelism that the extra molecule introduces).

This speed up was well below the expected value bearing in mind the largely independent nature of the computations making up each stage and the time taken by some of the stages. Part of the reason why the speed up is not equal to the number of processors is the high standard deviation (when compared with the mean) for the comparison stage because this leads to some of the procedures which the stage is broken up into taking much longer than the others and the next stage cannot begin until all of the procedures have finished. The high standard deviation also leads to the fact that the durations of the smaller procedures can add up to a value lower than the original duration. Hence the runs with a few processors (where this effect is more significant) take less time than they should (by up to about 15%), leading to a lowering in the speed up.

All in all, the simulation does indicate a significant decrease in the time taken when a simulated multiprocessor system is used. So it was decided to go ahead and implement the algorithm on a set of transputers and this is described in the next chapter. A point which has not been brought out in the above description is the way varying the number of procedures each stage was split up into had an unpredictable effect on the overall time taken and this led to the transputer implementation of the algorithm allowing this number to be input by the user (and will be met again in Section 6.4).

Number Of	Overhead Added To A Process' Duration Time				
Processors	0	.001*I	.01*I	.1*I	.001*I*I
1	1698	1698	1700	1726	1698
2	1010	1010	1013	1043	1010
3	745	746	750	799	746
4	666	667	672	725	669
5	588	589	595	659	592
6	561	561	568	639	565
8	500	503	525	625	520
10	426	429	453	641	453
12	374	377	402	626	408
16	330	333	362	616	376
20	297	301	338	626	370
30	241	243	294	654	383
40	210	215	272	710	436
50	202	208	269	799	497

Table 5.1 The Times(*) Taken By The Simulation For Comparing Identical Molecules Of Size 14

Number Of	Overhead Added To A Process' Duration Time		
Processors	0	.001*I	.01*I
1	306	306	307
2	211	212	216
3	161	162	165
4	138	139	144
5	122	123	128
6	115	116	122
8	103	104	111
10	92	95	107
12	80	84	104
16	65	69	98
20	57	61	92
30	47	52	87
40	41	48	92
50	38	45	97

Table 5.2 The Times(*) Taken By The Simulation For Comparing Molecules With A Common Substructure Of Size 12

(*) The simulated c.p.u. times are in seconds

Number Of Processors	Overhead Added To A Process' Duration Time	
	0	.001*I
1	16.5	16.7
2	11.7	12.1
3	8.8	9.3
4	7.6	8.1
5	6.8	7.4
6	6.3	7.0
8	5.7	6.5
10	5.4	6.4
12	5.0	6.2
16	4.2	6.2
20	3.8	6.3
30	3.3	6.8
40	3.1	7.1
50	2.9	7.5

Table 5.3 The Times(*) Taken By The Simulation For Comparing Molecules With A Common Substructure Of Size 9

Number Of Processors	Largest Common Substructure Size		
	14	12	9
1	1.00	1.00	1.00
2	1.68	1.45	1.41
3	2.28	1.90	1.88
4	2.55	2.22	2.17
5	2.89	2.51	2.43
6	3.03	2.66	2.62
8	3.40	2.97	2.89
10	3.99	3.33	3.06
12	4.54	3.83	3.30
16	5.15	4.71	3.93
20	5.72	5.37	4.34
30	7.05	6.51	5.00
40	8.09	7.46	5.32
50	8.41	8.05	5.69

Table 5.4 The Speed Up In The Case Of No Processor Overhead For The Cases Of Tables 5.1 To 5.3

(*) The simulated c.p.u. times are in seconds

Number Of Processors	Overhead Added To A Process' Duration Time		
	0	.001*I	.01*I
1	211	211	212
2	121	121	125
3	87	87	91
4	66	66	70
5	60	60	65
6	57	58	62
8	48	50	59
10	41	42	52
12	36	38	51
16	30	32	47
20	27	29	46
30	22	25	49
40	19	23	55
50	17	22	61

Table 5.5 The Times(*) Taken By The Simulation Of Comparing Three Molecules With A Common Substructure Of Size 11

* The simulated c.p.u. times are in seconds

CHAPTER 6

A TRANSPUTER IMPLEMENTATION OF CRANDELL AND SMITH'S ALGORITHM WITH NO SORTING STAGE

The results of a simulation of a multiprocessor system for Crandell and Smith's algorithm were described in the last chapter. Although the results did not indicate as large a speed up as had been hoped for, nevertheless they were favourable enough for a multiprocessor implementation to be undertaken. Because of their ready availability, transputers were chosen to be the processors making up the multiprocessor and before describing the implementation and its results, a short review of transputers will be given.

6.1 TRANSPUTERS AND OCCAM

6.1.1 A Brief Introduction To Transputers

Various parallel processing systems were considered in Section 5.2 and transputers were briefly mentioned there. Basically, they are microprocessors which can communicate with their neighbours by using "channels", thus allowing a powerful multiprocessor to be built. In more detail, each transputer consists of (see figure 6.1)

1) A conventional microprocessor which has a relatively low number of commands in its instruction set and all of them have the same format. Hence the transputer can be regarded as a reduced instruction set computer (RISC) [Taba87].

2) Four links which each have an input and an output

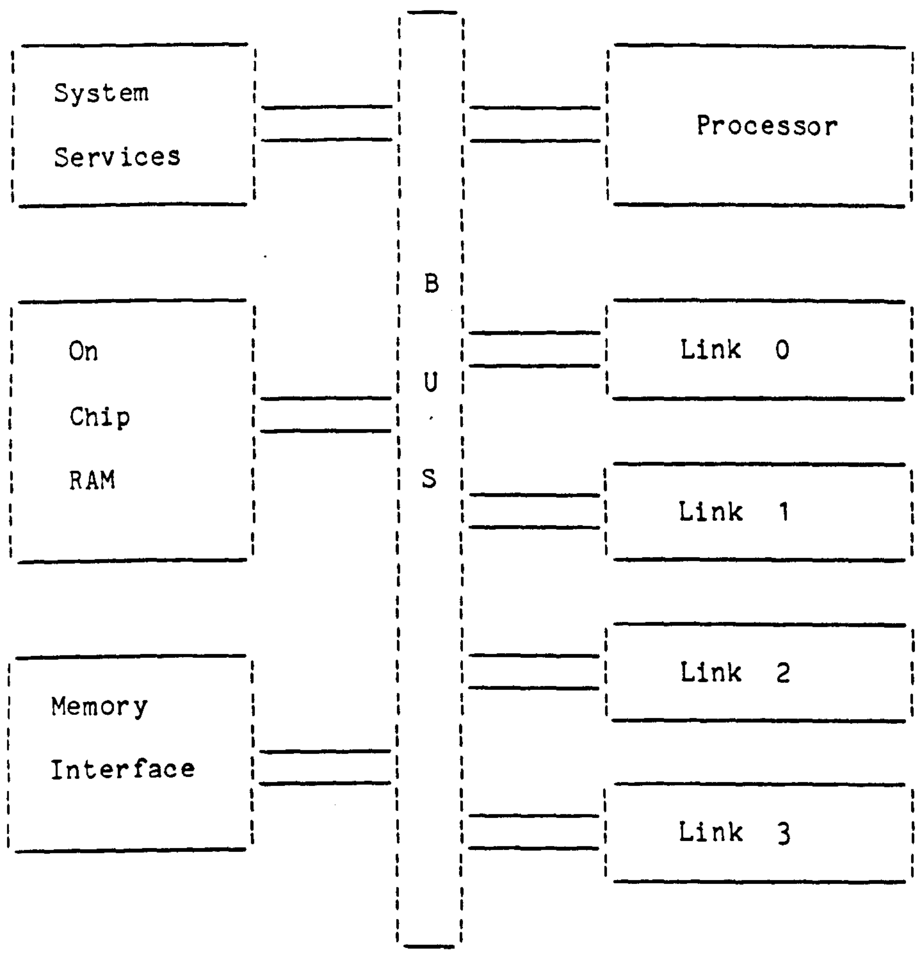


Figure 6.1 A Block Diagram Of A Transputer

channel for connection to another transputer.

3) 2K of on chip RAM with a maximum data transfer rate of 80 MBytes/second.

4) A memory interface for allowing off chip RAM with a maximum data transfer rate of 25 MBytes/second.

5) Various extra pins for use in booting the system, error tracing and the like (these are labelled as system services).

6) A bus to connect them all together.

The figures given above are taken from the description of the T414A Transputer in [INMO85] and give some indication of the performance values for transputers even though these figures will be slightly different for other members of the family.

Transputers have been designed so that programs written for a particular configuration of transputers can be executed on any other network of transputers. This is achieved by allowing a transputer to carry out two or more occam processes in parallel by time slicing (with the time slice for the T414A being approximately 800 microseconds).

6.1.2 A Short Outline Of Occam

Occam [May84, INMO84] is a high level language which is quite similar to Pascal but which does not have such elaborate data structures (eg. records and pointers). Although it has been developed principally for the transputer (and is pitched at a level just above the transputer's assembly language for ease of compilation), it is also intended to be a language which can be used on other concurrent systems [Fish86]. Blocks of code (the equivalent of code between the delimiters BEGIN and END in Pascal) are labelled as processes in occam and a named process is analogous to a procedure in Pascal. The major features of occam are:-

1) Concurrent execution in contrast to Pascal where if ProcA, ProcB are two procedures/processes, then BEGIN ProcA; ProcB END means execute ProcA and when it has finished execute ProcB (sequential execution), occam allows the two processes to be executed simultaneously. This is done by using the PAR (or parallel) construct:-

```
PAR
  ProcA
  ProcB
```

If it is required to execute ProcA and ProcB sequentially then PAR is replaced by SEQ.

2) To avoid competition for variables and hence non-deterministic execution, a process is not allowed to change variables which are being used by processes operating in parallel with it. Communication between parallel processes has to be carried out using channels and in a process they appear as a line giving the channel name, whether the data is being sent or received and the name of the data element. On reaching such a line, the execution of a process is suspended until the process that is being communicated with

reaches its corresponding communication line. If a ring of processes waiting to communicate with each other but no two of which are at corresponding communication lines develops (a deadly embrace or <u>deadlock</u>), then the program will never terminate.

3) The ALT construct allows input to be selected from one of several channels. For example, the code

```
    ALT
      ChannelA ? DataA
        SEQ
          BodyForA
      ChannelB ? DataB
        SEQ
          BodyForB
```

will receive code from ChannelA and then execute the associated code if the data is waiting on ChannelA when the ALT is first met. If there is no data waiting on ChannelA, then if there is any data waiting on ChannelB, it is received and the relevant body of code is executed, otherwise the process waits at the ALT until there is data available on one of the channels.

There are several differences between the original version of occam [INMO84] and occam 2 [Poun86] , the version currently being supplied for use on transputers. The major ones are:-

1) Abbreviations -to increase efficiency, sections of arrays can be abbreviated eg.

```
matrix.reduced IS [matrix FROM 1 FOR 3]:
```

where matrix is a one dimensional array, assigns to matrix.reduced three elements of matrix.

2) Two and three dimensional arrays are catered for whereas the first version of occam only allowed one dimensional arrays.

3) Originally, occam only had the single type INT (integer), now LOGICAL and (on some installations) REAL are provided.

4) .The types of data it is allowed to send on a channel have to be stated in occam 2 so that compile time checks for correct usage can be implemented. However, the version of occam 2 used for the implementation of Crandell and Smith's algorithm did not have this extra typing and no further mention will be made of it.

To give more of a feel for occam, a short example program is given in figure 6.2. The illustration process receives a stream of integers on channel "in" and sends the answers back out on channel "out". Internally there are four blocks of code being carried out in parallel one of which decides which of the two processing blocks to send the data to, and another block to receive the returned data. The process terminates when it receives the value zero on channel "in".

6.1.3 Structured Programming And Occam

One of the design aims lying behind occam is to try to make the programmer using occam write well

```
PROC take.square.roots(CHAN in, out)
CHAN from.a, to.a, from.b, to.b, to.end:
PAR
  SEQ
    LOGICAL flag:
    INT number, returned.value:
    SEQ
      flag:=TRUE
      WHILE flag
        SEQ
          ALT
            in ? number
              SEQ
                IF
                  number > 0
                    to.a ! number
                  number < 0
                    to.b ! number
                  number = 0
                    flag:=FALSE
                    to.a ! 0
                    to.b ! 0
                    to.end ! 0
  SEQ
    LOGICAL flag.receive:
    INT returned.value:
    SEQ
      flag.receive:=TRUE
      WHILE flag.receive
        ALT
          to.end ? returned.value
            flag.receive:=FALSE
          from.a ? returned.value
            out ! returned.value
          from.b ? returned.value
            out ! returned.value
```

Figure 6.2 An Example Of An Occam Process For Reading In A Stream Of Integers And Finding Their Square Roots

(Continued On The Next Page)

```
  SEQ
    LOGICAL flag.a:
    INT number.a, returned.value.a:
    SEQ
      flag.a::=TRUE
      WHILE flag.a
        SEQ
          to.a ? number.a
          IF
            number.a <> 0
              SEQ
                square.root(number.a, returned.value.a)
                from.a ! returned.value.a
            number.a = 0
              flag.a::=FALSE
  SEQ
    LOGICAL flag.b:
    INT number.b, returned.value.b:
    SEQ
    flag.b::=TRUE
    WHILE flag.b
      SEQ
        to.b ? number.b
        IF
          number.b <> 0
            SEQ
              square.root(-number.b, returned.value.b)
              from.b ! returned.value.b
          number.b=0
            flag.b::=FALSE
:
```

Figure 6.2 (Continued) An Example Of An Occam Process For Reading In A Stream Of Integers And Finding Their Square Roots structured programs [Dahl72]. Some examples of this philosophy are:-

1) The lack of a GO TO construct (the main constructs being WHILE, FOR and IF). However, although the problems resulting from the misuse of GO TO's have been recognised for many years [Dijk68], the issue as to whether they should be totally avoided has always been controversial [Knut74].

2) The level of nesting of a line of a program is determined by how far the statement is indented rather than, say Pascal's BEGIN/END structure. Hence the occam programmer is forced to "lay out" his code properly. The disadvantage is that making alterations to the code is more difficult (and the code is less easy to read) and again there is a degree of controversy over the worth of indenting programs [Shei81].

3) The folding editor (which is part of INMOS's transputer development system) works in an analogous way to hierarchies of menus in that lines of code can be "entered" to reveal the details of the underlying "procedure" which may in turn contain lines which can be entered (figure 6.3 gives an illustration of this). The overall effect is to encourage a program to be broken up into a hierarchy of small, largely self-contained blocks -which is a significant part of the definition of structured

```
PROC take.square.roots(CHAN in, out)

CHAN from.a, to.a, from.b, to.b, to.end:

PAR
  ...  receive.data
  ...  process.a
  ...  process.b
  ...  output.data
:
```

Figure 6.3 (a) The Top Level View Of The Program

```
{process.b
SEQ
  LOGICAL flag.b:
  INT number.b, returned.value.b:
  SEQ
  flag.b:=TRUE
  WHILE flag.b
    SEQ
      to.b ? number.b
      IF
        number.b <> 0
          SEQ
            square.root(-number.b, returned.value.b)
            from.b ! returned.value.b
        number.b=0
          flag.b:=FALSE
}
```

Figure 6.3 (b) Fold Process.b

Figure 6.3 An Example Of The Folding Editor Applied To The Program Of Figure 6.2

programming.

4) In the original version of occam, if none of the alternatives following an IF are true, the program continues by executing the next instruction. However, with occam 2, the program halts at the IF statement, thus forcing the programmer to consider all eventualities. Unfortunately, in practice it is quite likely that a TRUE SKIP (the equivalent of ELSE BEGIN END; in Pascal) will be included at the end of every IF statement.

6.1.4 Stages In The Development Of Software Environments For Parallel Processing

[Denn86] outlines four stages in the production of a software environment for use with parallel processors. Stage 1 is when parallelism is used with a single processor (eg. pipelining) and this is nearly invisible to the programmer, only requiring a slight restructuring of the code for vector or array processors. The next stage is the use of languages such as occam where the parallel execution of pieces of code on different machines is up to the programmer to control. However, this adds a far greater complexity to software development and [Denn86] suggests that functional languages will become widely used on parallel machines (stage 3). This is because the lack of variables means that if we are to try to evaluate f(g(2),h(3)) where f, g and h are functions, then g(2) and h(3) can be computed at the same time. [Gaud86] has proposed a design for a machine of this type based on transputers and using the data flow programming language SISAL. However, [Denn86] argues that functional languages cannot be easily applied to a wide range of problems and this will lead to a much higher level interface (stage 4) where a knowledge-based system will lead the user through the design process.

6.2 THE IMPLEMENTATION

Although, as was mentioned above, an occam program written for a particular configuration of transputers can run on any other configuration (running on a single transputer if necessary), there is quite likely to be a significant loss of efficiency. Therefore in a situation where an algorithm is being run on varying numbers of transputers, the implementation of the algorithm could be very different depending on the number of transputers being used. So as to avoid this problem (and the complexity and the extra work involved), Crandell and Smith's algorithm was broken up into stages which were in turn broken up into "jobs" in an analogous way to Chapter 5. These jobs were then distributed to the individual transputers which were connected to form a tree structure. A tree structure was chosen as opposed to other "regular" extensible patterns because the presence of closed loops seems to complicate the problem as

1) If they are used to allow alternative routes for jobs to reach the same destination, then intermediate transputers have the problem of not being aware of the whole situation.
2) If they are used as rings, then the path length the jobs have to travel is increased.

While neither of these reasons is strong enough to rule out the use of a structure containing rings (for example, (1) could be circumvented by using the "top" transputer to control the destination of jobs rather than the local control method described in the next section), they do provide some rationale for the assumed greater simplicity of a tree structure. Once a tree structure has been chosen, it seems natural to use as "thick" a tree as possible, that is one where transputers near the root of the tree use all four of their channels (thus minimising the path length from a transputer to the root transputer). The highly regular hypercube architecture which has been used with Intel's iPSC chip [Haye86] is not suitable for the current range of transputers as in an n-cube each node is connected to n other nodes. Chains of transputers have been compared with ternary trees in [Lync87] with the double linked chain performing surprisingly well, probably on account of the increased communication band width for data transfers between transputers.

6.2.1 The Tree Structure

Figure 6.4 shows 5 transputers making up a branch of the tree. Assuming that each "job" consists of reading a number from an array on Tran1, sending it to a transputer, undertaking a calculation using this number and returning the result to Tran1, then the occam executed by Tran11 can be split up into the processes shown in figure 6.5. These processes all execute in parallel and their basic structures are:-

1) CHECKING

This process is used to access a three element array giving information on whether Tran111, Tran112 and Tran113 are waiting for another data element to be sent, are executing or have closed down as all the data has been processed. The array is stored in a process so that it is protected from simultaneous accesses from processes executing in parallel (these processes have to communicate with CHECKING via channels).

2) RECEIVE_FROM_TRAN1

It receives data from Tran1, asks the CHECKING process for the name of a free transputer and then sends the data to this transputer.

3) RECEIVE_FROM_TRAN11*

Receives data from Tran111, Tran112 or Tran113, informs CHECKING that this transputer is waiting for further data and then passes the data on to TRAN1.

Tran111, Tran112 and Tran113 just receive data

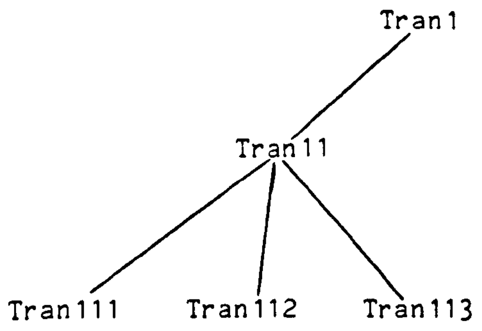


Figure 6.4 A Five Transputer Branch Of The Tree Structure

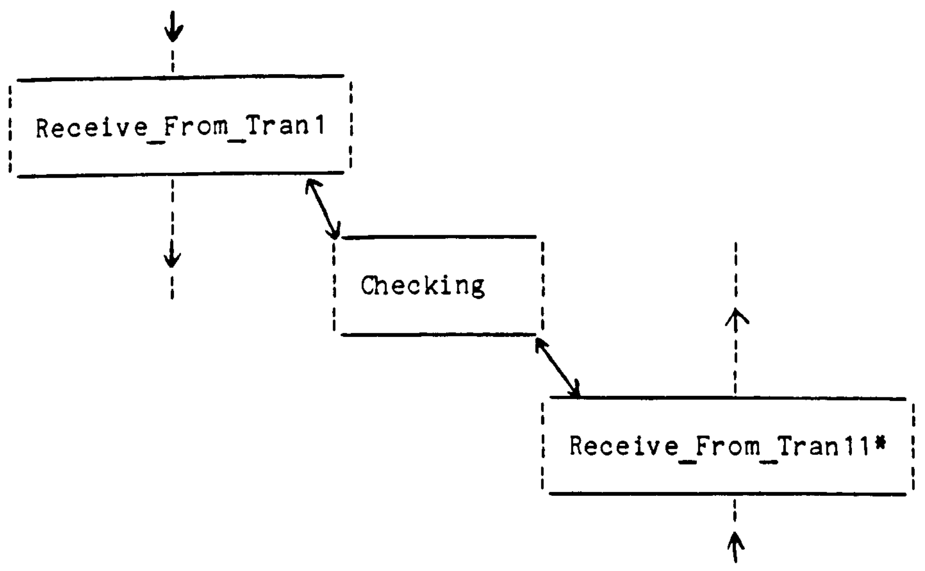


Figure 6.5 The Parallel Processes Executed By Tran11

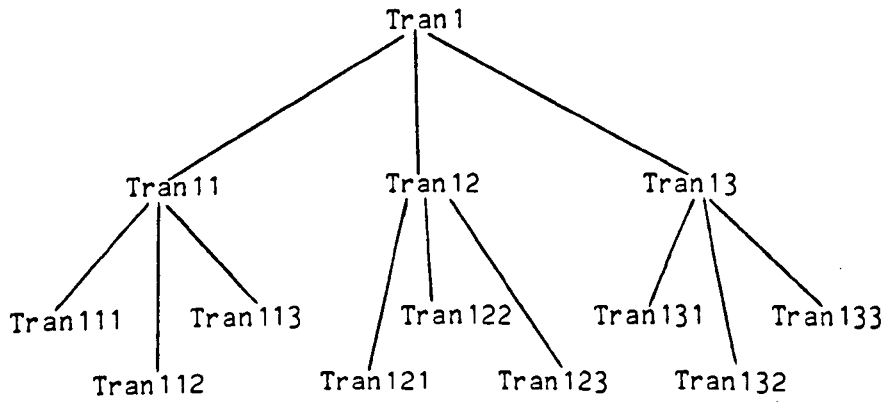


Figure 6.6 A Thirteen Transputer Branch Of The Tree Structure

elements, perform the calculation and then transmit the result, whilst Tran1 receives data, stores it and sends out the new data. The code avoids deadlocks because data is only sent to Tran11 if one of Tran111, Tran112 or Tran113 can receive it. Hence, RECEIVE_FROM_TRAN1 does not ever have to wait in mid-stream to input or output (apart from temporarily waiting its turn for CHECKING). Consequently no circle of processes halted in mid execution can occur, and so there cannot be a deadlock.

RECEIVE_FROM_TRAN11* informs CHECKING that a transputer has finished before sending the data to TRAN1 because otherwise TRAN1 could send the next piece of data to this transputer before its flag has been set and it could then be reset by RECEIVE_FROM_TRAN1. The tree is closed down by sending out a flag instead of a fresh data element every time Tran1 receives a result but has sent out all its data.

This basic structure can be extended by having Tran1 and Tran11 perform the same job processing function as Tran111, Tran112 and Tran113 in parallel with their "administrative activities". Also, additional branches can be added to Tran1 to give the 13 transputer tree shown in figure 6.6 (although only 11 transputers were available for use when the work described below was carried out). More transputers can be used by changing the arrays in CHECKING so that they indicate the number of free transputers down each of the branches (and making corresponding changes in the RECEIVE's so that these elements are incremented/decremented instead of being set or reset).

6.2.2 Distributing The Algorithm Over The Tree

The three stages in the loop of Crandell and Smith's algorithm, namely the grow_and_name, compare and amend steps, were split up into 2, x and y (where x and y could be varied) separate jobs which were then distributed over the tree. The growing and naming stage was only divided into two jobs because

1) growing a common substructure can lead to many candidate substructures being produced. This causes a problem as extra storage has to be allocated to take account of the worst possible case and memory requirements are as equally important as cpu time constraints on restricting the common substructure size. This situation is exacerbated by having several processes executing in parallel on one transputer as this causes several copies of the large data arrays used by the algorithm to be stored.

2) the added complexity of splitting up this stage further was not felt to be worthwhile as the time taken by this stage of the algorithm in the cases where the algorithm takes a long time to run, is relatively low (see the results given in Chapter 4). Additionally, the purpose lying behind the implementation was to analyse the performance of transputers on a chemical information

problem with a large apparent degree of parallelism rather than producing the fastest possible version of the algorithm at any cost.

In addition to distributing these jobs (and receiving their output), the root transputer also sends out global data in between the stages so as to keep the other transputers informed of the progress of the algorithm. The first of these stages is to send the distance table to the other transputer which is involved in the growing and naming stage. The other two distributions are the sending out of the named growths (ie. the n*(n-1)/2 inter-atomic "distances" for generation n) and the growth sets (ie. the n atoms making up each growth), and the sending out of the compared growth sets. These both involve the communication between transputers of very large three dimensional arrays, only a part of which is being used. As occam 2 only allows abbreviations to be used to take a slice of an array in one dimension only, it was decided to send out the arrays one row at a time and, on receiving one of these rows, a transputer makes three copies of it so that it can pass it on to the three transputers connected to it which are lower down in the tree, in parallel. (An alternative way of distributing the arrays would have been to send out the whole array in one go including the bits which are not relevant.) The implementation of the algorithm for the root transputer is shown in figure 6.7 where each box has to finish before the next one can begin.

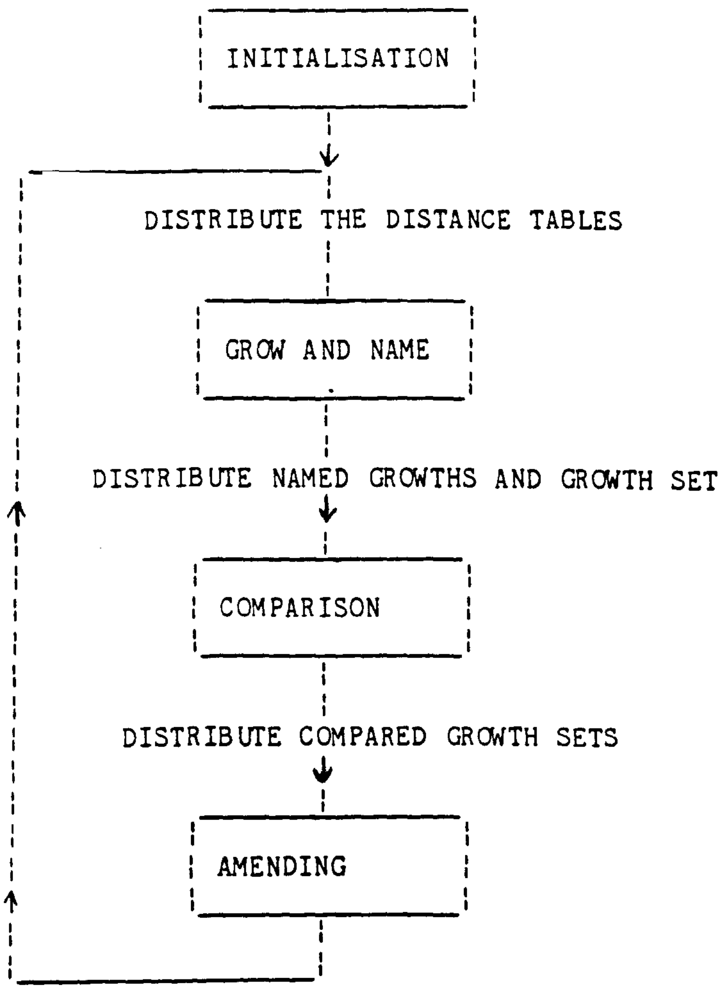

INITIALISATION
DISTRIBUTE THE DISTANCE TABLES
GROW AND NAME
DISTRIBUTE NAMED GROWTHS AND GROWTH SET
COMPARISON
DISTRIBUTE COMPARED GROWTH SETS
AMENDING

Figure 6.7 The Implementation Of The Crandell And Smith Algorithm

6.3 EXPERIMENTAL RESULTS

A prototype version of the algorithm was run on a Prime using Fisher's occam compiler [Fish85] before moving on to a Nimbus-hosted T414A transputer network where the algorithm was developed on one transputer before running it on more than one.

The algorithm was run using a distorted molecule of size 14 (9 carbons, 4 nitrogens, 1 oxygen) with a common substructure of size 12, an undistorted molecule of size 11 (5 carbons, 5 oxygens, 1 nitrogen), an undistorted molecule of size 8 (6 carbons, 2 bromines) and the same molecule distorted to have a common substructure of size 7. The number of transputers was varied for each structure and the number of jobs the comparing and amending stages were split into was also varied so as to try to produce the lowest possible times. The results are given in tables 6.1 to 6.4 where the entries are the lowest obtained for the given structures using the stated number of transputers ie. the compare and amend entries might have been obtained using different values for the number of jobs the algorithm was split into. The relevant values for the speed up, ie. the increase in speed over one transputer, are given in tables 6.5 to 6.7 and the corresponding graphs are figures 6.8 to 6.10.

(* See "Alterations" for an additional paragraph. *)

6.4 CONCLUSIONS

While it should be emphasised that the results are only relative in the sense that transputers with higher clock speeds (20 MHz as opposed to 12.5) and faster links are now available, tables 6.5 to 6.7 show that a high speed up can be obtained if there is a sufficiently large amount of computation to be undertaken. This concept of how much computation can be undertaken before communication has to take place (granularity) is at the root of how useful a parallel processing system is and the results indicate a small amount of speed up even when the granularity is quite low. Interestingly the speed up for the comparison stage when there was a common substructure of size 12 was higher than the number of transputers being used, stemming from the fact that the ordering of the substructures had been changed from splitting the compare stage up into many jobs. (The ordering for the next generation being the order in which the "packets" of compared growths are received back by the root transputer, and so it is likely that the "more oftenly occurring" growths move to the start of the list.) In retrospect, the growing and naming stage should have been split up into more than two jobs, but the problems over the extra storage needed must not be overlooked as it is storage considerations rather than the magnitude of the run times which prevents larger common substructures from being used.

It is very hard to compare the results obtained

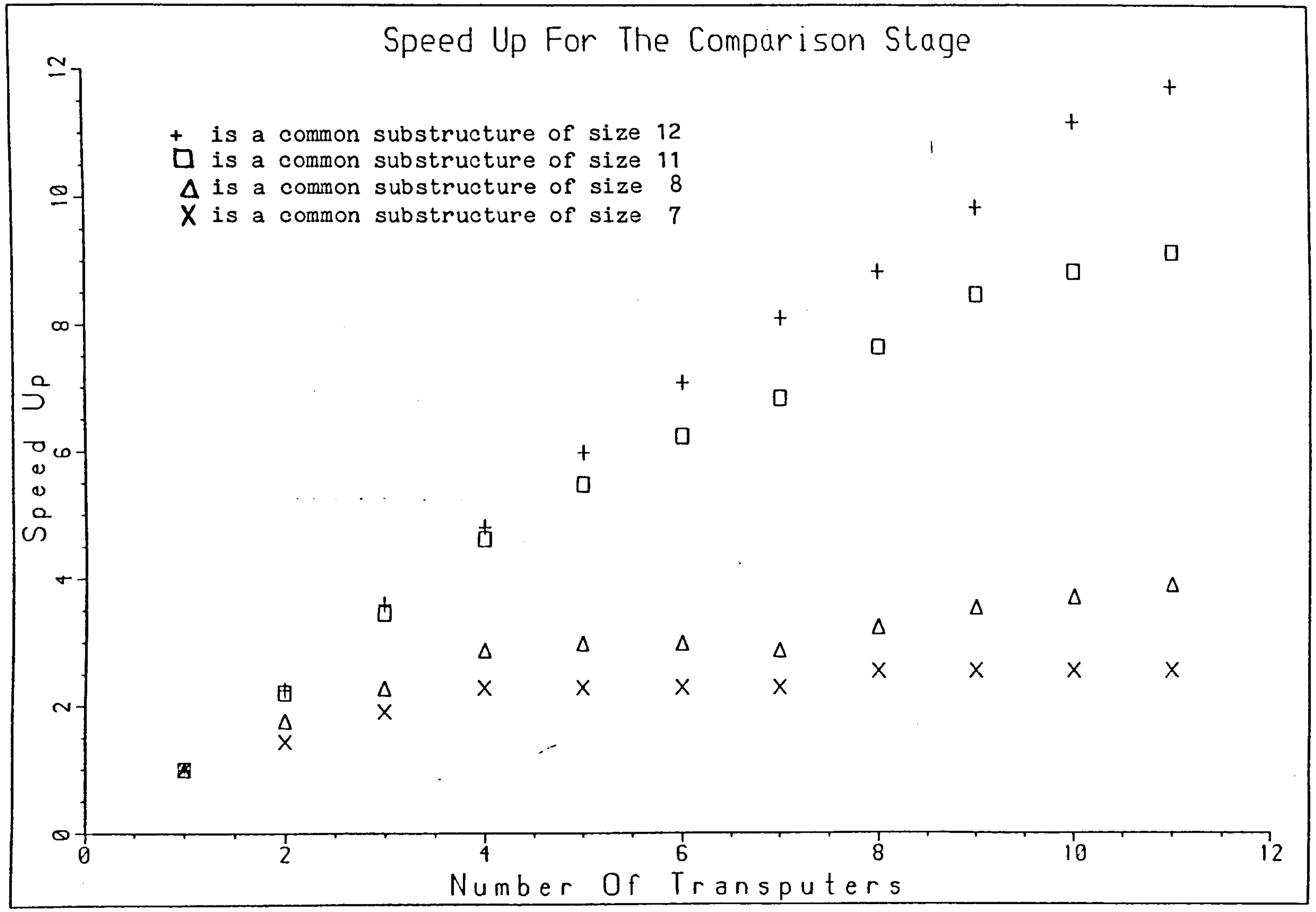


Figure 6.8 The Speed Ups For The Comparison Stages

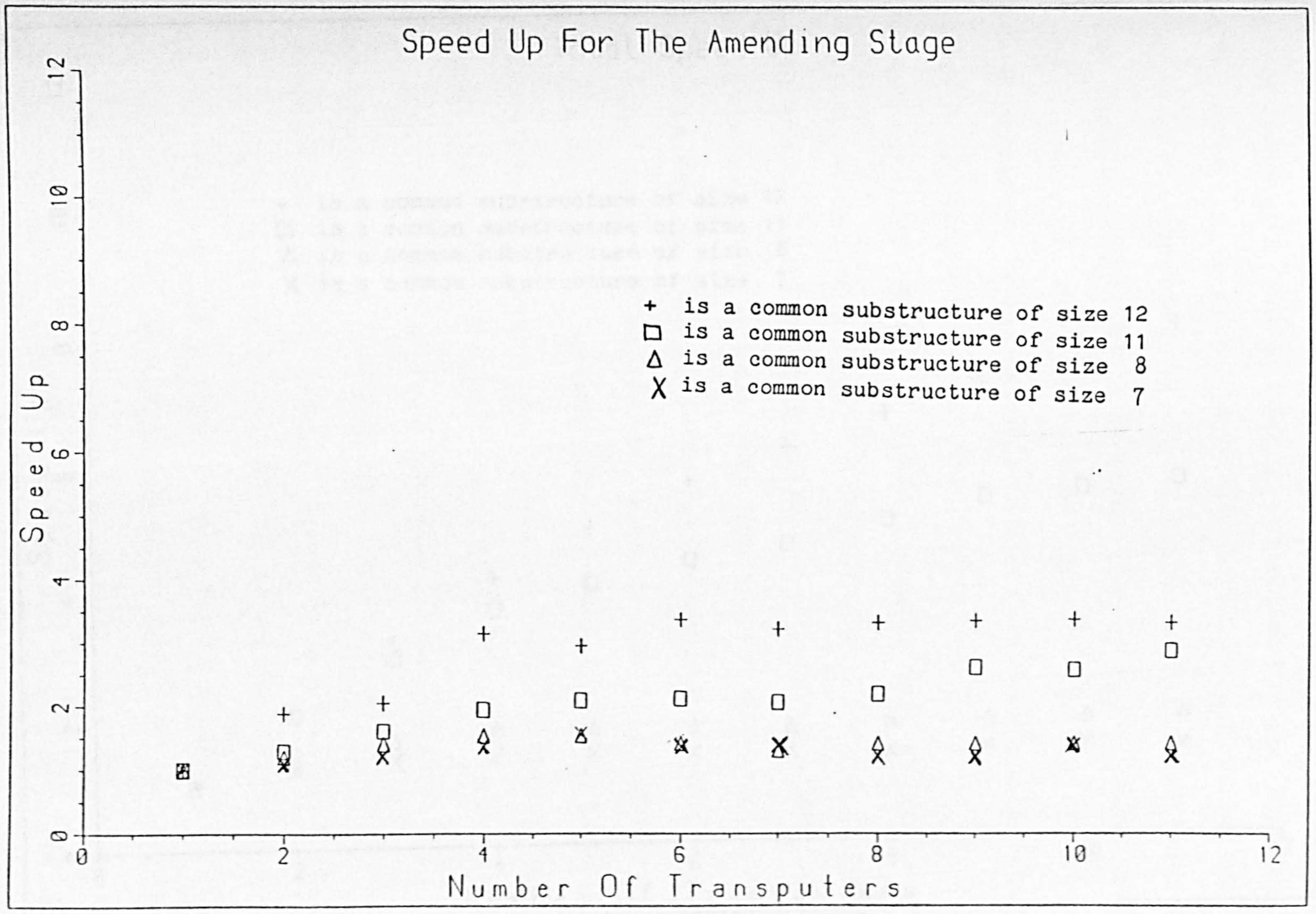


Figure 6.9 The Speed Ups For The Amending Stages

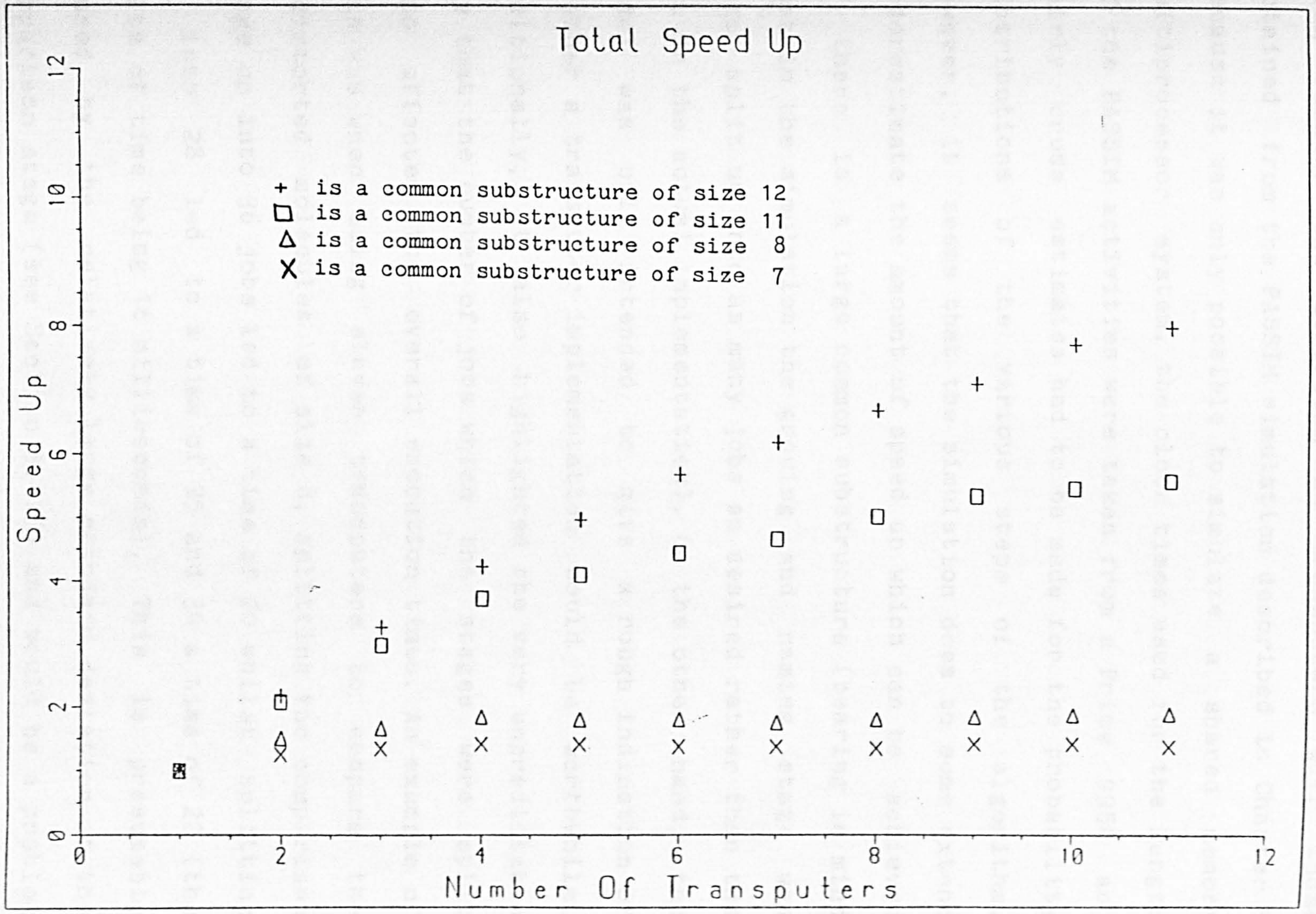


Figure 6.10 The Total Speed Ups

from running the algorithm on transputers with those obtained from the PASSIM simulation described in Chapter 5 because it was only possible to simulate a shared memory multiprocessor system, the clock times used for the length of the PASSIM activities were taken from a Prime 9950 and fairly crude estimates had to be made for the probability distributions of the various steps of the algorithm. However, it seems that the simulation does to some extent underestimate the amount of speed up which can be achieved if there is a large common substructure (bearing in mind that in the simulation the growing and naming stage was also split up into as many jobs as desired rather than the two in the actual implementation). On the other hand, the model was only intended to give a rough indication of whether a transputer implementation would be worthwhile. Additionally, it also highlighted the very unpredictable way that the number of jobs which the stages were split into affected the overall execution times. An example of this was when using eleven transputers to compare the undistorted molecules of size 8, splitting the comparison stage up into 36 jobs led to a time of 20 whilst splitting it into 28 led to a time of 25 and 34 a time of 22 (the units of time being 16 milliseconds). This is presumably caused by the relatively large standard deviation of the comparison stage (see Section 5.4.5) and would be a problem for any practical system.

As it stands, a transputer implementation of the above version of Crandell and Smith's algorithm is not very useful because Chapter 4 showed that the clique finding approach and the version of Crandell and Smith with an extra sorting stage were superior. However, as has been mentioned previously, a possible way around the problem of trying to extend the common substructure finding capability to molecules with larger common substructures might be to use the Crandell and Smith algorithm but with no distance clustering. The growths would then contain the end points for the range of values each inter-atomic distance could take and the comparison stage would then involve determining whether the ranges overlapped rather than a simple comparison of integers [Cran83a]. Therefore the net effect is likely to make the compare stage take appreciably longer with the growing and naming stage likely to be of similar duration to what it was originally (because there is essentially no structural change in this stage). Consequently, the algorithm without distance clustering is likely to take much longer to run than the version with clustering with most of the extra time being spent in the comparison stage, and so a transputer implementation might be an attractive way to speed up the algorithm.

KEY FOR TABLES 6.1 TO 6.4 :-

Send 1 is the distribution of the distance tables
Send 2 is the distribution of the named growths with their growth sets
Send 3 is the distribution of the compared growth sets
The times are in units of 16 milliseconds

	Number Of Transputers										
	1	2	3	4	5	6	7	8	9	10	11
Set Up	11	11	11	11	11	11	11	11	11	11	11
Grow	1379	724	724	724	724	724	724	724	724	724	724
Amend	722	380	349	228	242	213	223	216	214	213	216
Compare	25477	11279	7067	5286	4254	3594	3142	2882	2588	2280	2175
Send 1		3	3	3	3	3	3	3	3	3	3
Send 2		119	144	157	235	235	259	259	260	260	260
Send 3		32	39	43	63	63	63	66	66	66	69
Total	27590	12656	8417	6511	5538	4844	4449	4125	3881	3575	3457

Table 6.1 Common Substructure Of Size 12

	Number Of Transputers										
	1	2	3	4	5	6	7	8	9	10	11
Set Up	7	7	7	7	7	7	7	7	7	7	7
Grow	590	312	313	313	313	312	312	312	312	312	312
Amend	148	114	91	75	70	69	71	67	56	57	51
Compare	5490	2472	1580	1184	998	878	803	718	648	622	601
Send 1		2	2	2	2	2	2	2	2	2	2
Send 2		50	62	68	99	99	108	105	110	110	110
Send 3		14	18	19	28	28	28	30	30	30	31
Total	6235	2985	2080	1666	1512	1397	1333	1240	1165	1140	1116

Table 6.2 Common Substructure Of Size 11

	Number Of Transputers										
	1	2	3	4	5	6	7	8	9	10	11
Set Up	4	4	4	4	4	4	4	4	4	4	4
Grow	40	24	24	24	24	24	24	24	24	24	24
Amend	17	14	12	11	11	12	13	12	12	12	12
Compare	78	44	34	27	26	26	27	24	22	21	20
Send 1		1	1	1	1	1	1	1	1	1	1
Send 2		4	6	6	8	8	8	9	9	9	9
Send 3		2	2	2	3	3	3	3	3	3	3
Total	139	92	83	75	77	77	79	77	75	74	73

Table 6.3 Common Substructure Of Size 8

	Number Of Transputers										
	1	2	3	4	5	6	7	8	9	10	11
Set Up	4	4	4	4	4	4	4	4	4	4	4
Grow	16	11	11	11	11	11	11	11	11	11	11
Amend	11	10	9	8	7	8	8	9	9	8	9
Compare	23	16	12	10	10	10	10	9	9	9	9
Send 1		1	1	1	1	1	1	1	1	1	1
Send 2		2	3	3	4	4	4	4	4	4	4
Send 3		1	1	1	1	1	1	2	2	2	2
Total	54	43	40	38	38	39	39	40	38	38	39

Table 6.4 Common Substructure Of Size 7

Number Of	Common Substructure Size			
Transputers	12	11	8	7
1	1.00	1.00	1.00	1.00
2	2.26	2.22	1.77	1.44
3	3.61	3.47	2.29	1.92
4	4.82	4.64	2.89	2.30
5	5.99	5.50	3.00	2.30
6	7.09	6.25	3.00	2.30
7	8.11	6.84	2.89	2.30
8	8.84	7.65	3.25	2.56
9	9.84	8.47	3.55	2.56
10	11.17	8.83	3.71	2.56
11	11.71	9.13	3.90	2.56

Table 6.5 The Speed Up For The Comparison Stage

Number Of	Common Substructure Size			
Transputers	12	11	8	7
1	1.00	1.00	1.00	1.00
2	1.90	1.30	1.21	1.10
3	2.07	1.63	1.42	1.22
4	3.17	1.97	1.55	1.38
5	2.98	2.11	1.55	1.57
6	3.39	2.14	1.42	1.38
7	3.24	2.08	1.31	1.38
8	3.34	2.21	1.42	1.22
9	3.37	2.64	1.42	1.22
10	3.39	2.60	1.42	1.38
11	3.34	2.90	1.42	1.22

Table 6.6 The Speed Up For The Amend Stage

Number Of Transputers	Common Substructure Size			
	12	11	8	7
1	1.00	1.00	1.00	1.00
2	2.18	2.09	1.51	1.26
3	3.28	3.00	1.67	1.35
4	4.24	3.74	1.85	1.42
5	4.98	4.12	1.81	1.42
6	5.70	4.46	1.81	1.38
7	6.20	4.68	1.76	1.38
8	6.69	5.03	1.81	1.35
9	7.11	5.35	1.85	1.42
10	7.72	5.47	1.88	1.42
11	7.98	5.59	1.90	1.38

Table 6.7 The Speed Up For The Whole Algorithm

CHAPTER 7

DISTANCE GEOMETRY CALCULATIONS USING MULTIPROCESSORS

The previous chapter described a multiprocessor implementation of an algorithm for finding the largest substructure in common between two molecules. However, although the work was of theoretical interest in that it illustrated the capabilities of a multiprocessor system, it had no practical value because the chosen algorithm performed very badly when compared with other common substructure finding algorithms in Chapter 4, and also the value of being able to find common substructures is unclear. Therefore the current chapter examines the possibilities of using multiprocessors to perform distance geometry calculations (see Section 2.3.2); the reason for choosing the distance geometry field stems from the facts that a Monte Carlo method is used to generate different conformations and this can be carried out in parallel, and that a transputer system for use in the closely related field of molecular graphics is being developed commercially by Chemical Design Limited, Oxford. Additionally using distance geometry to try to find common pharmacophores is very computationally expensive, as can be seen from the 3.5 cpu hours used on a VAX 11/785 with floating point accelerator by [Sher86].

7.1 AN OUTLINE OF DISTANCE GEOMETRY

Chapter 2 mentioned that the use of distance geometry in chemistry has been developed by Crippen et al. [Crip81, Have82, Have83] as a means of finding possible conformations of a molecule given upper and lower bounds on the inter-atomic distances. The method comprises two stages, the first of which tries to remove any slack from the original upper and lower bounds by using geometrical considerations such as the triangle inequality. The second stage chooses a set of points which satisfy some of the inter-atomic distance bounds and attempts to refine these points until all the bounds are satisfied.

7.1.1 Tightening The Inter-Atomic Bounds

7.1.1.1 The Triangle Inequality

The triangle inequality says that if $D(i,j)$ is the distance between the points i and j, then

$$D(i,j) <= D(i,k)+D(k,j)$$

for all points k. So if $U(i,j)$ is the upper bound for the distance from i to j (and $L(i,j)$ is the lower bound), then

$$D(i,j) <= D(i,k)+D(k,j) <= U(i,k)+U(k,j)$$

for any point k. Therefore $U(i,j)$ can be set to the minimum of $U(i,j)$ and $U(i,k)+U(k,j)$. Consequently, a tightening of the upper bound matrix can be obtained by iteratively refining all the $U(i,j)$ in this way until no more slack can be removed [Crip81]. However, [Have83] points out that this is just equivalent to finding the shortest paths between points using the upper bound matrix -a problem for which a simple, highly efficient algorithm [Drey69, Floy62] exists. Alternatively, the shortest paths can be found by repeatedly applying Dijkstra's algorithm for finding the shortest paths from one point to all others in a graph [Aho74].

7.1.1.2 The Inverse Triangle Inequality

The triangle inequality says that

$$D(i,j) >= D(i,k)-D(j,k)$$

for all points k. But

$$D(i,k)-D(j,k) >= L(i,k)-U(j,k)$$

and so L(i,j) can be set equal to the greater of L(i,j) and the right hand side of this equation. The presence of U(j,k) in the equation means that the upper bounds need to be lowered before attempting to raise the lower bounds.

Similarly, L(i,j) is also bounded below by L(j,k)-U(i,k), and it can be shown [Have83] that repeatedly raising the lower bounds using these two constraints until no more lower bounds can be raised, is equivalent to setting L(i,j) equal to

$$\mathrm{MAX}_m\{ \mathrm{MAX}_k\{L(k,m)-U(i,k)\} -U(j,m)\}$$

Hence, by maximising L(k,m)-U(i,k) with respect to k first and then maximising the result minus U(j,m) with respect to m, any "inverse triangle inequality" slack can be removed. The maximising of L(k,m)-U(i,k) can be carried out using a variant of Dijkstra's algorithm for finding the shortest paths from one point to all others in a graph [Aho74]. The idea is to select any point, raise the lower bounds using any paths passing through it and then to remove this point from any further consideration; more details can be found in [Have83]. However, [Have84] performs the tightening of bounds by transforming the problem into that of finding the shortest paths between all points i and j where

$1 <= i <= n$ and $n+1 <= j <= 2n$

(n being the number of points) in the graph G where

```
G(k,m)=U(k,m)      if   k <= n     and     m <= n
G(k,m)=U(k,m)      if   n < k      and     n < m
G(k,m)=-L(k,m)     if   k <= n     and     n < m
G(k,m)=0           if   n < k      and     m <=n
```

The shortest path between i and j is then the negative of the inverse triangle bound between these points.

7.1.1.3 The Tetrangle Inequality

For four points, the cosine of the dihedral angle phi about the axis between points 1 and 2 (see figure 7.1) can be given in terms of the 6 inter-point distances as [Have83] (although the equation is actually stated incorrectly in this reference):-

```
        COS(phi) = (g + h) / (SQRT(e * f))
where e=4*D(1,4)*D(1,4)*D(1,2)*D(1,2)-
            (D(1,4)*D(1,4)+D(1,2)*D(1,2)-D(2,4)*D(2,4))**2
      f=4*D(1,2)*D(1,2)*D(2,3)*D(2,3)-
```

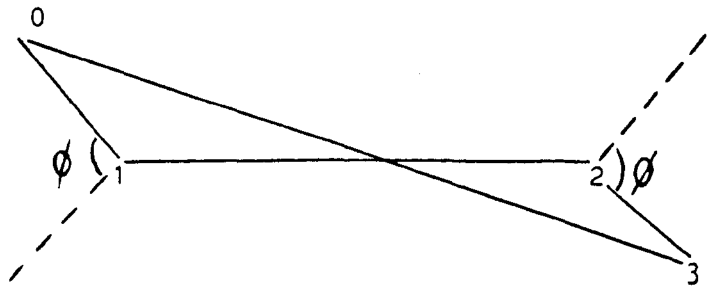


Figure 7.1 The Dihedral Angle About The Axis 1-2 (Taken From [Have83])

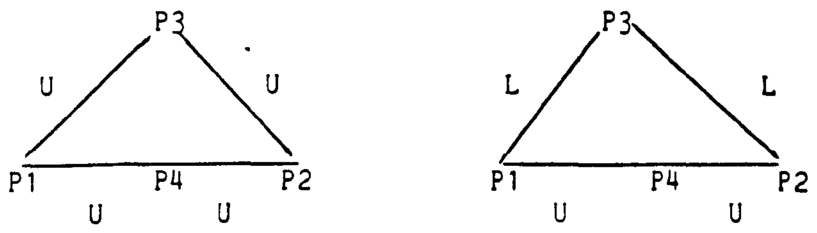


Figure 7.2 Checking Whether P4 Can Lie On P1-P2 When They Are As Far Apart As Possible. If It Can Then The Tetrangle Inequality Does Not Provide A Constraint On The Maximum Distance Between P1 And P2 (Taken From [Have83])

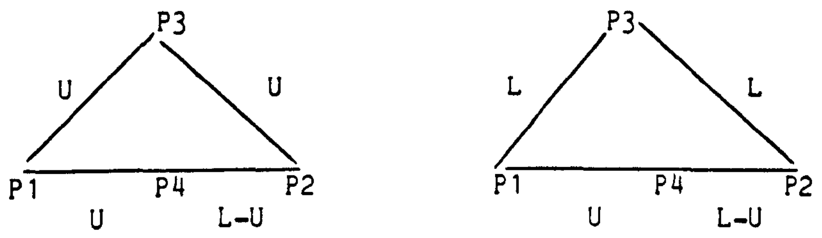


Figure 7.3 Checking Whether P4 Can Lie On P1-P2 When They Are As Close Together As Possible. If It Can Then The Tetrangle Inequality Does Not Provide A Constraint On The Minimum Distance Between P1 And P2 (Taken From [Have83])

$(D(1,2)^2+D(2,3)^2-D(1,3)^2)**2$

$g=(D(1,4)^2+D(1,2)^2-D(2,4)^2)*$

$(D(1,2)^2+D(2,3)^2-D(1,3)^2)$

$h=2*D(1,2)^2*$

$(D(2,4)^2+D(1,3)^2-D(1,2)^2-D(3,4)^2)$

Given the 5 distances other than D(3,4), then varying phi shows that the maximum and minimum values of D(3,4) occur when all 4 points lie in a plane. Further, it can be proved that [Have83]

Theorem Consider 5 of the 6 inter-point distances. If the upper and lower bounds on these distances prevent any three points becoming collinear, then the value for the sixth distance given by the above equation for COS(phi), attains its maximum and minimum when the 5 distances are at some combination of their upper and lower bounds.

Consequently, a tetrangle inequality can be obtained between 4 points for limiting an inter-point distance by taking the 64 combinations of the 5 other distances at their upper and lower bounds and COS(phi) equal to 1 or -1. Hence the upper and lower bound matrices can be tightened by iteratively taking sets of four points at a time until no more slack can be removed. However three checks need to be carried out in conjunction with the tetrangle inequality:-

1) The non-colinearity of the points under analysis needs

to be tested. Consider first the tetrangle inequality applied to the upper bound between points P1 and P2 and assume that

$$U(P1,P3)+U(P2,P3) > U(P1,P4)+U(P2,P4),$$

then it is required to determine whether P4 can be on the line from P1 to P2 when P1 and P2 are at the maximum distance apart that the triangle inequality allows. This is done by considering the two triangles in figure 7.2 and the two limits on the distance for P3 to P4 which they produce (which can be found by using simple trigonometry). If this distance range overlaps with the allowed upper-lower bound range for P3-P4, then the tetrangle inequality cannot be used to lower the upper bound for this distance because the triangle inequality upper bound can be obtained when three points are collinear. Therefore the theorem cannot be invoked to try to lower this limit.

In a similar way the tetrangle inequality can only be applied to the lower bound between P2 and P4 if at least one of the inverse triangle inequality limits is positive (otherwise the points are not necessarily distinct) and for each positive limit (say L(P1,P2)-U(P4,P1)) the allowed range of values for D(P3,P4) in figure 7.3 does not overlap with the interval L(P3,P4) to U(P3,P4).

2) The distances being passed to the tetrangle inequality need to be checked to ensure that the triangle inequality holds between them (as they will be a mixture of distances

from the upper and lower bound matrices). This check is needed because the above equation for COS(phi) is obtained by using the law of cosines. It is also necessary to check that when considering the distance P3-P4 neither P1, P2 and P3 nor P1, P2 and P4 are collinear (as then phi in figure 7.1 is undefined).

3) After the tetrangle inequality has tightened a distance, the triangle and inverse triangle inequalities need to be reapplied to the upper and lower bound matrices. This can be achieved quickly by noting that if U(Pi,Pj) was changed, then the new U(Pk,Pm) is the old one or U(Pk,Pi)+U(Pi,Pj)+U(Pj,Pm) or U(Pk,Pj)+U(Pi,Pj)+U(Pi,Pm). A similar idea applies to the lower bound matrix.

In theory similar checks can be devised for pentangles, hexangles, etc., but the calculations involved become very difficult.

7.1.2 Finding Co-ordinates Which Satisfy The Constraints

After the two bound matrices have been tightened (by using the method of Section 7.1.1.1 followed by that of Section 7.1.1.2 and finally that of Section 7.1.1.3), a trial distance, TD(i,j), is chosen from each L(i,j) to U(i,j) interval by using a pseudo random number generator. Unfortunately, because there are less than n*(n-1)/2 degrees of freedom (where n is the number of points under consideration), the resulting entries in the trial distance matrix need not even obey the triangle inequality. Indeed a knowledge of all the inter-dependencies (correlations) of the distances is roughly equivalent to the whole problem, but the use of a more simple correlation index to cut down on the geometric violations rather than just using random numbers, has been put forward [Crip81].

7.1.2.1 Obtaining Approximate Co-ordinates

A basic result from linear algebra is that an n*n real symmetric matrix, RSM, has n real eigenvalues and n orthogonal eigenvectors. Therefore if E1,..,En are the eigenvalues in order of decreasing absolute value, $U_1,..,U_n$ are the corresponding unit length eigenvectors and V is the matrix whose i^{th} column is U_i, then

$$\mathrm{RSM} = V * \left| \begin{array}{ccccc} E1 & 0 & 0 & \dots & 0 \\ 0 & E2 & 0 & \dots & 0 \\ 0 & 0 & E3 & \dots & 0 \\ & & & & \\ & & & & \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & En \end{array} \right| * V\_transpose$$

Hence,

$$\mathrm{RSM}(i,j) = \quad Ek * U_k(i) * U_k(j)$$

But the only assumption about RSM was that it was a real symmetric matrix, so if RSM(i,j) is defined to be the scalar product of the vectors, R_i and R_j, from some origin to i and j, then

$$RSM(i,j) = \quad R_i(k) * R_j(k)$$

Equating terms in the two expressions for RSM(i,j) gives

$$R_i(k) = SQRT(Ek) * U_k(i).$$

As RSM is a distance matrix, it has a rank of at most three and so the co-ordinates of the the point i are given by the above equation with k taking the values 1, 2 and 3.

Using the law of cosines it can be shown that RSM(i,j) is equal to

$$(DIS(i,o)^2 + DIS(j,o)^2 - DIS(i,j)^2) / 2$$

where "o" is the origin and DIS(k,l) is the distance between points k and l. Furthermore [Have83] proves that if the centre of mass is taken as the origin, then the distance from the origin, DIS(i,o), is the square root of

$$(\sum_{j=1}^{n} DIS(i,j)^2) / n \quad - \quad (\sum_{j=1}^{n} \sum_{k=j+1}^{n} DIS(j,k)^2) / n^2$$

If instead of using actual distances the trial distances from TD are used, then more than three eigenvalues may be non-zero. However, an approximation to the co-ordinates of each point can be obtained by taking the three eigenvalues of largest absolute value (provided that they are all positive -if one of the three eigenvalues is negative then this method cannot be applied). These approximate co-ordinates can then be passed on to an optimization stage to try to make them obey the original upper and lower distance bounds.

The three largest eigenvalues of a matrix can easily be calculated by using the power method [Atki83] which relies on the fact that any vector can be written as a linear sum of the eigenvectors U_1 to U_n

$X = Q_1*U_1 ,+ Q_2*U_2 + \ldots\ldots + Q_n*U_n$

where Q_1 to Q_n are real numbers. Multiplying by the matrix just changes the coefficient of each term from Q_i to E_i*Q_i where E_i is the relevant eigenvalue, and so multiplying by the matrix m times leads to the coefficients being E_i to the m, times Q_i. Therefore, assuming that the largest eigenvalue has only one eigenvector and that the coefficient Q_1 is non-zero, repeatedly multiplying X by the matrix will converge on the eigenvector associated with the eigenvalue of largest magnitude. (Because of rounding errors on real number operations these two assumptions are likely to be valid, but if the absolute values of the eigenvalues are close together or if the original vector had a very small Q_1 value, then the convergence is likely to be very slow.) The other eigenvectors can be obtained using deflation [Atki83] which relies on the fact that if

$B = A - E_1 * U_1 *$ Z1transpose

where A is the original matrix with eigenvalues $E_1,..,E_n$ and eigenvectors $U_1,..,U_n$ and Z1 is any vector such that

Z1transpose $* U_1 =1$

then the eigenvalues of B are $0,E_2,..E_n$ and $U_2,..,U_n$ are given in terms of the eigenvectors $W_2,..,W_n$ of B by

$Ui = (E_i-E_1) * W_i + E_1 *$ (Z1transpose $* W_i) * U_1$

This result can be proved by substituting the expressions for $U2$ and A in $A*U2=E2*U2$.

The importance of the above result about taking the centre of mass as the origin is that in the approximation the distances from the origin get disproportionately weighted compared to the other inter-point distances.

7.1.2.2 Cyclic Co-ordinate Descent

After approximate values for the atomic co-ordinates have been obtained from the eigenvectors, they are refined by decreasing the value of the function E which is the sum (over 1 <= i < j <=n) of

((D(i,j)**2/U(i,j)**2)-1)**2 for D(i,j) > U(i,j)

((D(i,j)**2/L(i,j)**2)-1)**2 for D(i,j) < L(i,j).

As E is a quartic in each co-ordinate, differentiating with respect to a co-ordinate gives a cubic whose roots will give the minimum value for E obtained by varying this co-ordinate independently of the others. As the roots of a cubic can be found arithmetically, it is thus easy to apply the cyclic co-ordinate method [Baza79] to reduce E. This just cycles through the co-ordinates taking the root giving the lowest value of E until the roots chosen are all the same as the ones chosen on the last cycle through the roots.

7.1.2.3 Conjugate Gradient Method

The cyclic co-ordinate technique only provides a fairly rough and ready way of improving the co-ordinates, so after this stage the method of conjugate gradients [Baza79] is applied to the function F defined to be the sum (over 1 <= i < j <=n) of
((D(i,j)**2/U(i,j)**2)-1)**2 for D(i,j) > U(i,j)
((L(i,j)**2/D(i,j)**2)-1)**2 for D(i,j) < L(i,j).
The idea is to take the gradient of F and then to find lambda such that
F(Y + lambda * G) is a minimum
where lambda is greater than zero, Y are the current co-ordinates and G is the gradient. The new value of Y is then set equal to Y plus lambda times G, and the process is repeated. However, to avoid obtaining a "poor direction" on nearing a stationary point, the conjugate gradient method (as opposed to that of steepest descents) also takes into account the previous value of G when it calculates the new value of G.

The value of lambda was calculated using the golden section method [Baza79] which evaluates the function at two points inside the interval. Then it moves the nearer end point to the interior point which has the highest function value before repeating the process. The efficiency and the name of the algorithm stems from taking the interior points to be 0.618 times the length of the interval, from one end point. Hence when the interval is shortened one of the previous interior points remains as an interior point. Strictly speaking, the golden section method is designed to be used on intervals where the function is convex. F is not convex but [Have83] points out that F is likely to be "almost" convex in small neighbourhoods and the golden section method was found to be perfectly adequate in the work carried out below. An alternative and perhaps more efficient way of calculating lambda, would have been to use the Newton-Raphson method.

Other functions could be optimised rather than just the functions E and F described above if particular features wish to be incorporated in the final conformation. In fact [Have83] also uses a function C in order to preserve chiral centres but it is not described here as it was not used in the experimental work reported below.

7.2 A SEQUENTIAL ANALYSIS OF DISTANCE GEOMETRY

A FORTRAN 77 program was written to implement the basic distance geometry algorithm on a Prime 9950. The upper and lower distance bounds used were those for a molecule of size 18 given in [Weng82] and those for molecules of sizes 22, 23 and 36 obtained by applying the technique outlined in [Weng82]. This consists of specifying the upper and lower bounds to be the same if the two atoms are connected to each other or to a common third atom -the distances being found using standard bond lengths and angles. The distances between atoms for which the shortest path between them goes through two other atoms is given using standard bond lengths and angles and a dihedral angle of 0 or 180 degrees. The lower bounds between other atoms were set at 2.0 A whilst the upper bounds were set at 10 times the cube root of the number of atoms (this last figure was the value used in [Weng82]). However, in obtaining the limits for the above molecules the bond lengths and angles were obtained from the actual co-ordinates given in the Cambridge Crystallographic Database rather than from a set of tables or a molecular construction program. Also no account was taken of the extra rigidity constraints imposed by rings, and so the upper and lower bounds used were looser than they would normally be. This looseness led to it being difficult to find a set of random numbers for which the three eigenvalues of largest absolute value were positive for molecules of sizes 22 and 36. Therefore, as far as these molecules are concerned, only the results for the bound tightening stages are given below.

Two other sets of bounds were produced by firstly combining the molecules of sizes 18 and 23 to form a set of 41 points. This is useful for finding whether the original two molecules have a common pattern [Sher86] and is achieved by setting the lower bounds to zero when one atom is from one molecule and one from the other. The upper bounds are set to a high value in the same case except where the atoms concerned are conjectured to be "corresponding" atoms in the common pattern, when the upper bounds are set to a minimum tolerance value. In the present case, three atoms from a ring were chosen as the common pattern and the tolerance was set at 0.2 A. The second new set of bounds was obtained by repeating the process with the molecule of size 18 again. However, this last extension is clearly very artificial.

Table 7.1 shows the results of an analysis of the bound tightening stage of the algorithm. Dijkstra's and Floyd's algorithms for the triangle inequality are very similar performance wise but the inverse triangle inequality procedure given in [Have84] is significantly quicker than that of [Have83]. The performance of the tetrangle stage was very poor, but this is in line with [Have82] which says that "The TRNGL and TRINV algorithms [ie. the triangle and inverse triangle tightening algorithms using Dijkstra's algorithm] described there are quite efficient and completely reliable. Unfortunately, they are not capable of detecting the majority of violations of three dimensionality that can occur in the bounds. The TTRGL algorithm [the tetrangle algorithm] is capable of reliably detecting violations of an additional, and more significant, set of constraints but at its present state of development it is not an efficient algorithm. The EMBED algorithm [the production of eigenvalues and the smoothing of the resulting bound violations] is the only one we have that is capable of accounting for the complete set of constraints. It is neither highly efficient nor completely reliable."

The performance of the rest of the algorithm after the bound smoothing, was analysed by attempting to produce a series of conformations for the bound sets of sizes 18, 41 and 59. The conjugate gradient method was carried out for 50 iterations (or until the value of F was below 0.075). The results showing the amount of time spent in calculating lambda and in determining the gradient are shown in tables 7.2 and 7.4. The first two of these only deal with the cases where all 50 cycles were required; there were several instances for the structure of size 41 where F fell below the desired value before then, and the lowest time of these was 52.2 seconds.

Clearly the number of iterations employed by the power method in determining the eigenvalues and by the golden section algorithm in calculating lambda are a major factor in how much time is spent in each stage. The former was taken to have a maximum value of 100 (with the actual value being less if the eigenvalue had been determined to the desired accuracy). The latter was set at 25 and the starting interval was 0 to 0.1 A. Thus the timings given in tables 7.2 to 7.4 can only provide an illustration of the likely times for a "real life" application of distance geometry.

Bearing in mind that [Sher86] fixed the number of iterations in the conjugate gradient algorithm at 1000, the above results suggest that a multiprocessor system with each processor generating a possible conformation, could lead to a speed up for the conjugate gradient stage of about 12.9 when generating 13 solutions (the number used in [Sher86] was 25) for the data set of 59 points -the speed up figure being obtained by summing all the times and dividing by the largest.

7.3 A TRANSPUTER IMPLEMENTATION OF DISTANCE GEOMETRY

Although the results of the previous section show that most of the time taken by the algorithm is spent in steps that are difficult to write parallel code for, it was decided to go ahead with a transputer implementation. This was because of the ability to generate different conformations concurrently using different sets of random numbers. Once a serial occam version had been written (and its results checked against the FORTRAN 77 program as they both used the same pseudo random number generator [Knut81]), as many of the steps as possible were parallelised in order to examine the capabilities of transputers in dealing with a much finer grain parallelism than that met in Chapter 6 (and as a lesser goal to improve the overall performance of the algorithm). Despite the fact that the tetrangle inequality step is very time consuming and appears to be able to be split up into sections which can be executed concurrently, it was not implemented in the occam version of the algorithm as it would still have been a very time consuming stage whose net worth is far from clear (with it being likely to depend on the source of the original upper and lower bounds). Additionally, it is not used in the more recent accounts of work carried out using distance geometry [Have84, Sher86].

The bound tightening via the triangle and inverse triangle inequalities was carried out using a parallel version of Floyd's algorithm put forward in [Deo80]. The concurrency stems from the fact that all the j loops in figure 7.4 can be carried out simultaneously (that is in occam, instead of writing the line SEQ j=1 FOR ..., the line PAR j=1 FOR ... could be written). This is because a problem can only occur if one of the other processes going on in parallel changes U[j,i] or U[i,k]. However, U[j,i] can only be changed by the procedure under consideration, and U[i,k] can only be altered by setting j=i which in fact leads to no change. As can be seen from figure 7.4 though, the resulting parallelism is very fine grain in nature.

Parallelizing the conjugate gradient method is difficult because of the problem of trying to find the value of lambda which produces a minimum of the function. A concurrent determination of the gradient is much simpler as it is composed of 3*n (where n is the number of atoms) independent calculations, these being composed of tests to see whether the distances from each point to the point under consideration, are out of bounds and if they are, taking the relevant partial derivative. However, it did not prove possible to formulate a way of parallelizing the calculation of lambda. Unfortunately, the concurrent implementation of the conjugate gradient method in [Seag86] only deals with the special case of symmetric linear

```
SEQ  i  FROM  1  TO  number.of.points
  SEQ  j  FROM  1  TO  number.of.points
    IF  U[j,i] < maximum.value  THEN
      SEQ  k  FROM  1  TO  number.of.points
        IF  U[j,k] < (U[j,i] + U[i,k])  THEN
          U[j,k] := U[j,i] + U[i,k]
        END IF
      END SEQ
    END IF
  END SEQ
END SEQ
```

Figure 7.4 Pseudo Code For Floyd's Shortest Path Algorithm

systems.

It is also not clear how to parallelize the cyclic descent algorithm as splitting up the determination of which roots to use into batches of co-ordinates, could possibly lead to the situation where choosing a particular root for co-ordinate A reduces the value of the function E, and similarly for B, but using both these choices at the same time increases E. On the other hand, the initial step of calculating the various cubic roots could clearly be carried out in parallel as any calculation on a cubic is independent of calculations on the other cubics. However, this was not undertaken in the implementation.

Before giving the results of the attempts at a parallel implementation, attention should be given to the differences in the time taken in various stages of the algorithm between the FORTRAN 77 version run on the Prime 9950 and the occam version run on a T414A transputer. Tables 7.7 and 7.8 give the times for the stages of the algorithm when trying to determine a conformation for the sets of 41 and 59 points (as mentioned earlier, the times can be compared because using the same random number generator ensured that the calculations carried out were the same). Because of the similarities between the two programs (due to the occam version being derived very closely from the FORTRAN), the differences in the relative times stem from the different capabilities of the two computers. The T414A transputers which the program was executed, on carry out floating point operations by calling software subroutines, and so real number calculations are very slow. This was not a problem with the bound tightening via the triangle and inverse triangle inequalities as the bounds were represented as integers, but from the determination of the eigenvalues onwards, real numbers were used. However, T800 transputers have hardware for floating point operations, but they had not been commercially released at the time the above work was carried out (though they are expected to replace the T414 in becoming the most widely used member of the transputer family). Hence the Prime figures give a better indication of the relative expense of each stage.

The parallel pieces of code were executed on up to four transputers, with one of the transputers (the "root") executing the whole algorithm. On encountering a concurrent section of the program, it splits the section up into the number of transputers it is attached to, pieces and distributes them to these transputers. Unlike the program reported in Chapter 6, the root transputer did not carry out any of these parallel pieces of code. This was in order to try to minimise any overheads keeping in mind the fine grain nature of the parallelism.

7.4 CONCLUSIONS

In the principal example of distance geometry being used to find pharmacophoric patterns [Sher86], the large number of iterations of the conjugate gradient stage meant that this stage consumed the vast majority of the time spent executing the algorithm. The above results seem to indicate that a multiprocessor system geared for real number calculations can achieve substantial speed ups when the "natural" parallelism of generating a different conformation on a different processor is used. However, it appears very difficult to make a significant improvement on this speed up figure by introducing parallelism into the determination of each conformation.

The original version of distance geometry [Crip81] used an iterative method for both the triangle and inverse triangle inequality bound tightening procedures. This was expensive computationally and involved a large number of calculations which could be carried out in parallel. However the time spent in this stage can be reduced to a very low amount by using a shortest path algorithm, and so the more efficient implementation has reduced some of the scope for parallelism. The speed up (or rather the lack of it) obtained by using a parallel version of Floyd's shortest path algorithm was far below that reported in [Deo80] for Denelcor's HEP [Smit78, Hock85, Hiro86] where a speed up of between 7.8 and 6.5 was obtained for 40 node graphs. Some of the better performance by the HEP might be due to the different graphs used, however the more significant part follows on from its more tightly coupled architecture. This is based on a pipelined processor which is capable of having instructions from upto 8 different processes in the pipeline at a time, but with the restriction that there can never be two instructions from the same process in the pipeline. Access to memory is via a switch and thus several processors can be joined together to create a multiprocessor with data being rapidly "transferred" between processors by altering the switch.

The calculation of the gradient showed a speed up of 2.19 in both cases when four transputers were used, but using a transputer with hardware for floating point operations will clearly reduce this (by how much is obviously not certain). However, as it was not possible to parallelize the calculation of lambda stage, the above just serves to illustrate the relationship between speed up obtained and the granularity of the parallelism.

So far this thesis has described work involving the searching of a database for a given pharmacophoric pattern and techniques which could be used to discover such a pattern using as input a small number of molecules which are thought to act in the same way. The next chapter extends this by giving details of a system which allows a database to be searched to discover molecules which have a similar 3D structure to the query.

Step	Structure Size 18	22	23	36	41	59
TRNGL	0.1	0.1	0.1	0.5	0.7	2.1
FLOYD	0.1	0.2	0.2	0.6	0.8	2.0
TRINV	0.4	0.7	0.8	3.1	4.6	14.2
INVFLOYD	0.2	0.3	0.3	1.2	1.7	5.1
TETRAN	76.5	75.3	98.4	***	769.6	***

Table 7.1 Times(*) For The Various Stages Involved In Tightening The Upper And Lower Distance Bounds

* The times are in cpu seconds for a Prime 9950
TRNGL is the triangle inequality procedure using Dijkstra
FLOYD is the triangle inequality procedure using Floyd
TRINV is the inverse triangle inequality procedure given in [Have83]
INVFLOYD is the inverse triangle inequality procedure using the shortest paths approach in conjuction with Floyd
TETRAN is the tetrangle smoothing procedure

Calculating Lambda	Calculating The Gradient	Total Time
107.8	63.7	171.5
107.0	60.7	167.7
109.7	79.9	189.6
108.1	69.7	177.8
108.9	74.4	183.3
108.3	62.8	171.1
108.3	63.3	171.6
110.1	77.4	187.5
107.0	56.8	163.8
108.7	65.3	174.0
108.5	66.4	174.9
108.6	65.3	173.9
108.0	65.1	173.1
107.3	57.4	164.7

Table 7.2 Times(*) Taken For 50 Iterations Of The Conjugate Gradient Method On The Data Set Of 59 Points

* The times are in cpu seconds for a Prime 9950

Calculating Lambda	Calculating The Gradient	Total Time
53.8	40.7	94.5
53.1	36.6	89.7
53.8	43.1	96.9
53.1	36.7	89.8
53.0	34.7	87.7
54.3	45.6	99.9
54.0	39.3	93.3
53.6	35.3	88.9
54.4	41.6	96.0
54.1	39.1	93.2
54.1	39.7	93.8
54.3	39.7	94.0
53.7	38.5	92.2
50.6	33.8	84.4

Table 7.3 Times(*) Taken For 50 Iterations Of The Conjugate Gradient Method On The Data Set Of 41 Points

Calculating Lambda	Calculating The Gradient	Total Time	Number Of Iterations
4.2	3.9	8.1	15
6.1	5.5	11.6	22
3.9	3.5	7.4	14
3.9	3.5	7.4	14
6.2	6.2	12.4	22
6.2	5.9	12.1	22
4.8	4.5	9.3	17
7.3	7.2	14.5	26
13.6	13.0	26.6	50
13.9	14.3	28.2	50
4.2	3.8	8.0	15
3.6	3.3	3.9	13
6.4	6.0	12.4	23
4.8	4.4	9.2	17

Table 7.4 Times(*) Thirteen Conformational Calculations On The Data Set Of 18 Points

* The times are in cpu seconds for a Prime 9950

SIZE 41 Eigenvalue Determination	41 Cyclic Descent	59 Eigenvalue Determination	59 Cyclic Descent
2.04	3.54	7.60	17.9
3.01	2.48	4.84	10.8
2.21	4.95	3.66	8.2
2.42	2.92	5.45	11.8
2.49	3.75	5.01	10.1
2.35	2.82	5.72	8.6
1.85	9.73	7.40	15.4
2.43	3.46	5.19	16.0
2.93	3.23	5.31	14.6
2.77	3.43	3.66	9.7
3.10	5.25	5.51	10.6
2.76	5.49	5.70	9.1
2.71	3.29	6.96	8.9
1.54	2.24	5.52	12.7

Table 7.5 Times(*) Taken By Thirteen Instances Of The Eigenvalue Determination And Cyclic Descent Stages

Structure	Triangle Inequality			Inverse Triangle Inequality		
	Number Of Transputers			Number Of Transputers		
Size	1	2	3	1	2	3
18	10	29	30	17	86	72
22	16	41	42	30	121	104
23	19	45	45	32	134	117
36	60	105	101	128	296	234
41	85	138	129	190	388	316
59	232	301	261	555	802	630

Table 7.6 Times(+) Taken For The Bound Tightening Steps On A Transputer System

* These times are in cpu seconds for a Prime 9950
+ Transputer times are in units of 16 milliseconds

		Number Of Transputers		
Stage	FORTRAN 77	1	3	4
Triangle	0.8	85		
Inverse	1.7	190	The Same As	
Eigenvalue	2.0	1115	For One	
Cyclic	3.5	1504	Transputer	
Lambda	53.8	24625		
Gradient	40.7	10097	6185	4611

Table 7.7 Times For Finding A Conformation For The Data Set Of Size 41 Using 50 Iterations Of The Conjugate Gradient Method

		Number Of Transputers		
Stage	FORTRAN 77	1	3	4
Triangle	2.0	232		
Inverse	5.1	555	The Same As	
Eigenvalue	7.6	4131	For One	
Cyclic	17.9	11005	Transputer	
Lambda	107.8	50467		
Gradient	63.7	16346	9772	7471

Table 7.8 Times For Finding A Conformation For The Data Set Of Size 59 Using 50 Iterations Of The Conjugate Gradient Method

* These times are in cpu seconds for a Prime 9950
+ Transputer times are in units of 16 milliseconds
The figures for one transputer are for the serial algorithm
No figures are given for two transputers as this corresponds to the serial algorithm being carried out on one of the tip transputers
The figures for three and four transputers are when there were two and three tip transputers respectively (the root transputer carrying out no work)

CHAPTER 8

SEARCHING FOR THREE DIMENSIONALLY SIMILAR MOLECULES

Chapter 3 described a system for searching for known pharmacophoric patterns whilst in Chapter 4, two algorithms for comparing a set of molecules to find their common 3D substructures were compared. The current chapter reports on work involving the combining of the 3D screening system used in substructure searching with the clique finding approach to finding the maximum common substructure, so as to try to find an efficient way of searching the Cambridge Crystallographic Database for molecules with a similar 3D shape to a pattern molecule. Thus, rather than comparing two (or more) molecules to identify the maximum common substructure, a single target molecule is matched against all of the molecules in a database so as to identify those which are most similar to it. This could be of possible use for situations such as where a potential new drug has been identified by lead generation, and similar compounds are needed to attempt to improve the activity (and also give information on where the features leading to the activity are situated). Additionally, it could be of use in interpreting spectra by way of a facility to find similar structures and seeing whether they have similar spectra. A somewhat analogous system has been described for 2D browsing [Will86, Will87b] but a very different representation and measure of similarity were used.

8.1 THE THREE DIMENSIONAL SIMILARITY SEARCHING SYSTEM

In order to try to produce a system which could be used interactively, the search program was split up into two stages in an analogous way to the search systems described in Chapters 1 and 3. The first stage tried to remove molecules which were dissimilar to the pattern molecule by a computationally inexpensive check. The remaining molecules were then passed on to a full comparison stage so as to produce a measure of their similarity with the pattern molecule. The similarity measure used was the size of the largest common substructure mainly because it was suitable for the applications outlined above and the fact that it was easy to calculate using one of the algorithms of Chapter 4.

More explicitly, the first stage used the 3D screens developed by Jakes et al. [Jake86, Jake87a] so as to produce an upper bound for the similarity measure between a query molecule and the pattern. This was done by forming a graph of the same size as the pattern molecule. The connectivities were determined by taking each inter-atomic distance in the pattern molecule and checking whether the screen corresponding to this distance and the two relevant atomic types was set in the query molecule (hence the screens using unknown X atoms were not used, but a system interested in finding molecules of similar shape without regard for atomic types, could use them). If it was, then a 1 was entered in the graph at position (i,j) where i and j were the pattern atoms involved, otherwise a 0 was entered. After which the graph was examined to find the size of the largest clique. This value then placed a maximum upper bound on the size of a common substructure between the pattern molecule and the query structure. This was because if a substructure was in common between the two molecules then, for the atoms corresponding to this substructure in the pattern molecule, each inter-atomic distance must occur in the query. Therefore, the screens for these distances would be set in the query molecule's screen list and all the relevant pattern molecule's atoms would therefore be connected to each other in the graph.

To illustrate this process, consider the molecule BARGOQ shown in figure 8.1 and suppose that it is being compared with a hypothetical molecule A. If A's screen list does not contain a screen corresponding to the distance between the two oxygens, then they cannot both be present in a common substructure and the two oxygens are therefore unconnected in the graph. Hence, they cannot belong to the same clique. Alternatively, if the relevant oxygen-oxygen screen is set, then the two oxygens could possibly belong to the same common substructure, and so they are regarded as being connected, thus allowing them to both be present in the same cliques. Consequently, each common substructure must be contained in a clique of the screen-based graph.

If the upper bound value for the size of the maximum common substructure was above a threshold, expressed as some minimum number of atoms, then the query

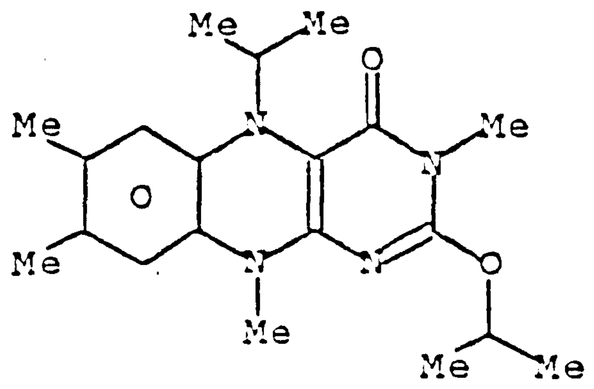


Figure 8.1 The Structure Diagram For The Molecule With Identifier BARGOQ

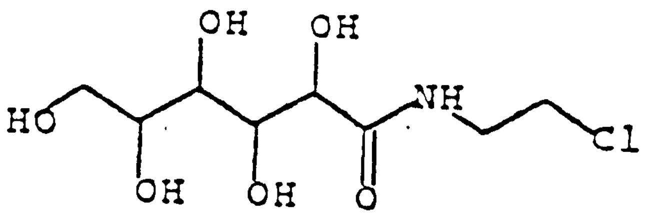


Figure 8.2 The Structure Diagram For The Molecule With Identifier CEGLCA

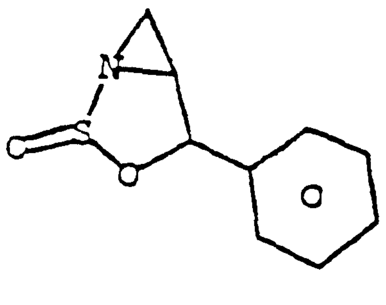


Figure 8.3 The Structure Diagram For The Molecule With Identifier BAGTOS

structure was analysed in detail using the efficient correspondence graph/clique finding approach of Chapter 4. As explained in detail there, this method basically just combines the graphs of the pattern and query structures in such a way that common substructures correspond to the cliques of the new graph. However, the times taken by clique finding algorithms increase very rapidly with the size of the graph, and so it was hoped that eliminating some compounds from consideration by carrying out a rough and ready check using a small graph, would lead to a significant improvement in performance.

The aim here, as elsewhere in this thesis, has been the development of efficient procedures for 3D structure matching; accordingly, our evaluation of this proposed best match searching system will be based upon its computational requirements, that is its efficiency, rather than the chemical nature of the molecules which are retrieved in the search.

All the times given in this chapter will be solely for the clique finding stages of the method unless otherwise stated; the time taken to set up the screens and the graphs will not be considered. This is because this latter time is likely to be small and to be heavily influenced by file access times (and the availability of the bit handling functions allowing the intersection and union of sets used in [Jake86, Jake87a]).

8.2 IMPLEMENTATION AND RESULTS

8.2.1 The Molecules And Clique Finding Algorithm Chosen

Nine hundred and ninety nine structures evenly spaced throughout the Cambridge Crystallographic Database (CCD) with an average size of 20.3 (non-hydrogen) atoms, were used as the basic data for the study. Where several sets of co-ordinates were available in the CCD for a molecule, only the first of these was used. The screens for the structures were stored as a list of TRUE/FALSEs whilst the co-ordinates were stored separately because of their bulk.

A variable tolerance was used when checking to see whether a pattern distance had a screen set for the query structure (the relevant screens to be considered being found using a table look up). The cliques of the resulting screen-generated graph were enumerated using three of the algorithms of Section 4.3. The algorithms chosen were the standard algorithm of Bron and Kerbosch [Bron73], the simple algorithm of Golender and Rozenblit [Gole83] and the maximal independent set algorithm of Loukakis and Tsouros [Louk81]. However, only Bron and Kerbosch's algorithm was used to analyse the correspondence graph formed for a "hit" as the other algorithms had already been shown to be inferior in the work reported in Chapter 4. Unfortunately, if the correspondence graph has more than about 1000 nodes, severe problems emerge when trying to list all the cliques (see Chapter 4). Therefore hits where the correspondence graph was of size greater than 1000 had to be ignored, but these were extremely rare and in fact did not occur in any of the runs reported below.

The above method was coded in FORTRAN 77 on an IBM 3083 and Bron and Kerbosch's algorithm was again found to be better overall than the other two clique finding methods. However, compared with Chapter 4, Golender and Rozenblit performed extremely badly while Loukakis and Tsouros performed much better sometimes being 3 to 4 times quicker than Bron and Kerbosch on graphs containing a large clique. This further illustrates the fact that the lack of "randomness" found in the graphs produced from three dimensional co-ordinate data can lead to algorithms which perform very well on randomly generated graphs, performing badly. The results of comparing two molecules whose (structure diagrams are given in figures 8.1 and 8.2) with sections of the database are given in table 8.1. It should be pointed out that Bron and Kerbosch's algorithm given in [Bron73] is recursive and so, as the FORTRAN 77 compiler used did not allow recursion, multiple copies of the subroutine were created. An alternative non-recursive formulation is given in [Kuhl84]. Additionally, the graphs were held as arrays of one byte logicals rather than the default of four bytes in order that the program should use less than 3 MBytes of core storage.

8.2.2 The Performance Of The Two Stage System

In fact, the above description of the algorithm was modified by recording which pattern atoms occurred in at least one relevant clique for the graph formed from considering the screen information. Then, when the correspondence graph was formed, only these atoms were allowed to be the first atoms in the pairs of atoms which made up the nodes of the correspondence graph. This stemmed from the fact that the maximum clique size that a pattern atom occurred in in the correspondence graph, could not be greater than its maximum clique size for the screen-generated graph. As an illustration of the improved performance possible from using this extra information, CEGLCA was searched against the section of the database from molecule 801 to molecule 900 with an error tolerance of 0.15 Å and a minimum clique size of 4. When the minimum number of non-carbons that had to be present in a clique was set to zero the time for the version of the algorithm using the extra information to reduce the size of the correspondence graph was 24.8 seconds of cpu time whilst that without the additional data was 25.2 seconds. However, specifying that at least two non-carbons had to be in every clique led to times of 15.9 seconds and 25.2 seconds respectively.

Molecules were chosen randomly from the intervals 1 to 200, 201 to 400 and 401 to 600 of the database (with their structure diagrams making up figures 8.3 to 8.5), and

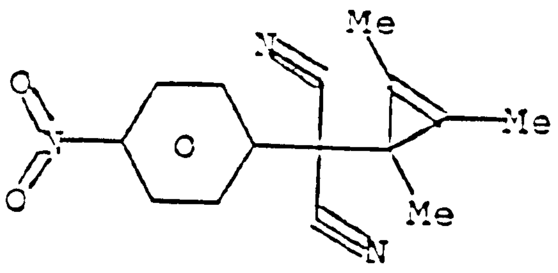


Figure 8.4 The Structure Diagram For The Molecule With
Identifier CAVROG

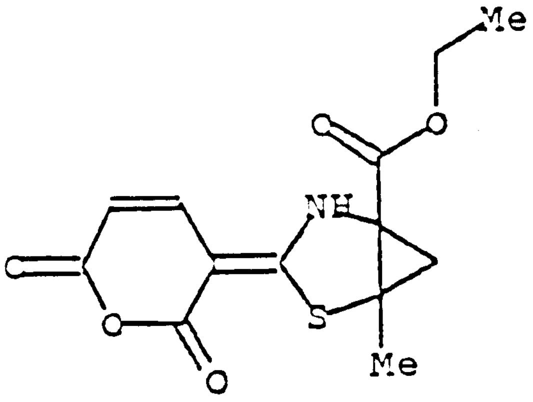


Figure 8.5 The Structure Diagram For The Molecule With
Identifier ETCOHX

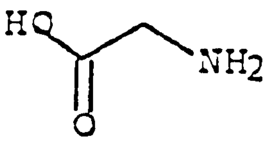


Figure 8.6 The Structure Diagram For The Molecule With
Identifier GLYCIN

the results of the searches are given in tables 8.2 and 8.3. Table 8.2 shows the effect of varying the error tolerance limits for the first of these molecules while table 8.3 gives the times for the three molecules when this limit was set to the value 0.15 A (and this value was also used for tables 8.4 to 8.9). As it stands, the algorithm just tends to locate rings because these are of a rigid 3D shape and usually contain more atoms than the other common 3D structures. Consequently, it was decided to allow the minimum number of non-carbons in the common substructure to be specified and the results of using this extra constraint are also included in table 8.3. Table 8.4 gives the number of molecules eliminated from consideration by stage one in the searches listed in table 8.3 (along with the total number of molecules having an appropriate substructure).

The results given in the various tables were obtained by using the standard FORTRAN compiler on the IBM 3083. However, an optimising compiler which carried out such things as register and branch optimization along with code-movement (the latter possibly leading to logic changes in the program) [Metc85], was also available and to give some indication of the improved performance in terms of speed that it gives molecule CAVROG was compared with the database using an error margin of 0.15 Angstroms, a minimum clique size of 4 and no restriction on the number of non-carbons. The resulting time of 398 cpu seconds was almost half the original time of 700 seconds.

As the times taken in table 8.3 are quite high, the three smaller molecules shown in figures 8.6 to 8.8 (each one having a reasonable percentage of non-carbons) were searched against the database. Table 8.5 shows that the times now taken are far less than with the larger molecules.

8.2.3 The Addition Of A Third Stage

The results of table 8.3 show that it is the time spent in the comparison stage which dominates that of the screening stage. Consequently, an extra, higher precision screening stage was added after the first screening stage in a similar manner to the "distance search" of the 3D substructure searching system of Section 3.1. As with the original screening stage, a graph of the same size as the pattern molecule was used. To determine whether the nodes i and j were connected in the graph, the distance between the i th and j th pattern atoms was formed. If one of the query structure's inter-atomic distances was the same as this distance and if the relevant atoms were of the correct types, the i and j nodes were connected, otherwise they were unconnected. This was determined by retrieving a sorted list of all the query's inter-atomic distances and then performing a binary search on this list. Thus the exact distance checking stage only connected two nodes if their pattern distance definitely occurred in the query structure. This contrasted with the stage based on the substructure searching screens which connected the two

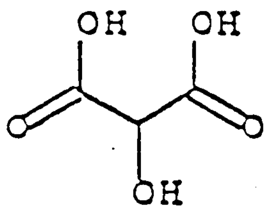


Figure 8.7 The Structure Diagram For The Molecule With Identifier HMALAC

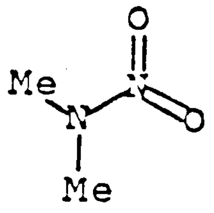


Figure 8.8 The Structure Diagram For The Molecule With Identifier METNAM

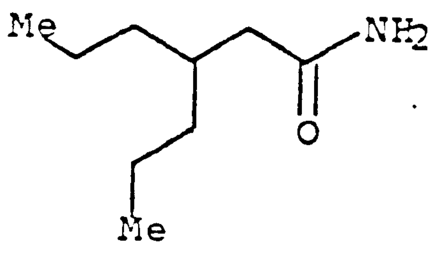


Figure 8.9 The Structure Diagram For The Molecule With Identifier DPPRAM

nodes if the screens allowed the pattern distance to be in the query.

Table 8.6 gives the results of tests using this two stage screening system with the figures in brackets being a comparison with the program when only a one stage screening system was used (the structure diagrams for the molecules are given in figures 8.3 to 8.11). The given times include the times for the binary searches. The times for the full search with no screening stage for molecules DPPRAM, NBENDC and PRPENC were 233.2, 259.2 and 241.3 seconds respectively while the times for the first part of the screening stage were 2.1, 2.7 and 2.6 seconds.

Whilst table 8.6 indicates that this extra stage could lead to a significant improvement when two non-carbons were specified as being required, it had very little effect when cliques containing only carbons were allowed. One reason for this lay in the fact that the screening system of Section 3.1 assigned many more screens to the carbon-carbon distance range than to those between other atomic types (for example, there were 19 screens for the oxygen-oxygen range and 61 for the nitrogen-carbon range, while the carbon-carbon range had 153 screens). Hence, the screening system could more accurately predict whether a particular carbon-carbon distance was present, as opposed to an oxygen-oxygen distance.

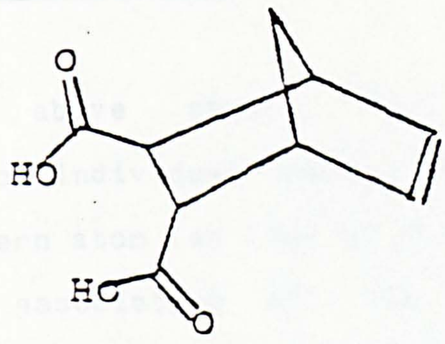


Figure 8.10 The Structure Diagram For The Molecule With Identifier NBENDC

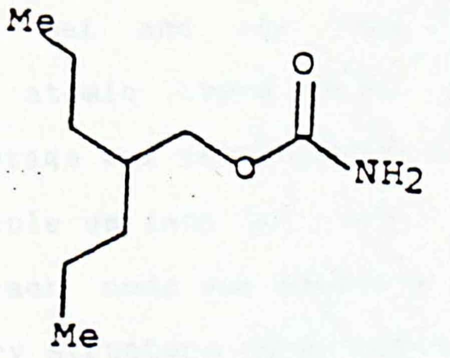


Figure 8.11 The Structure Diagram For The Molecule With Identifier PRPENC

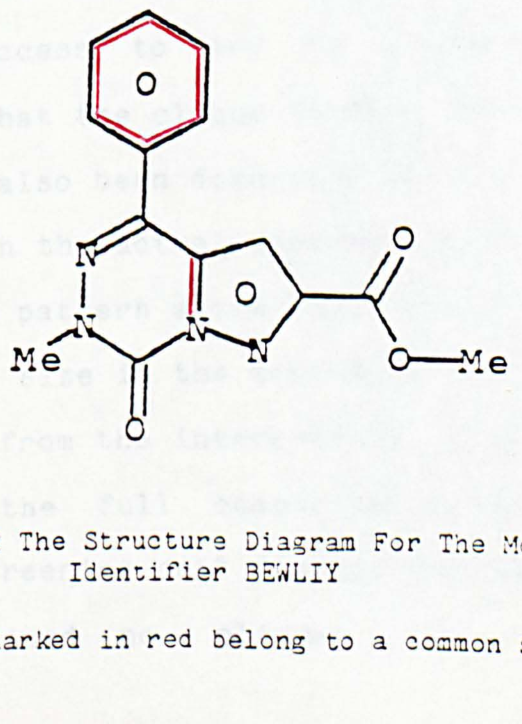


Figure 8.12 The Structure Diagram For The Molecule With Identifier BEWLIY

The atoms marked in red belong to a common substructure.

8.2.4 An Intermediate Stage

The above stages form a graph either by associating each individual query structure atom in turn with each pattern atom (as long as the atomic types are the same) or by associating all the query atoms with each pattern atom as in the second screening stage. Two nodes of the graphs so produced are connected if the distance between the pattern atoms is equal to one of the distances formed by taking a query structure atom from the first pattern atom's set and one from the second atom's set, subject to the atomic types being equal. Therefore an intermediate stage was developed by splitting the atoms of the query molecule up into two sets and then forming a graph where each node was composed of a pattern atom and one of the query structure sets (the graph being of twice the size of the pattern molecule). This idea was met previously in Section 4.6 where it was used with very limited success to try to increase the size of the molecules that the clique finding approach could deal with, and it has also been described by [Boll79].

In the actual implementation of this intermediate stage, only pattern atoms that occurred in cliques of a sufficient size in the screening stage were used as input. The output from the intermediate stage also reduced the input to the full comparison stage in a similar way, as well as "screening out" some of the query molecules because they contained no cliques of a sufficient size.

Additionally, it was possible for this stage to restrict in which half of the structure matches for the pattern atom should be sought from when setting up the correspondence graph for the final stage. The query structure was split into two sets by just taking the first half of its atoms as they occurred in the database. Clearly, much more elaborate methods could be devised and they could well have a significant effect on the overall performance. The results of runs using this intermediate stage are given in table 8.7 with the figures in brackets being those for when only a two stage screening system was used; the figures include the time taken for the binary searches but the three lists of inter-atomic distances for each query structure which were used by this stage, were assumed to have been presorted. It can be seen that the intermediate stage consumed a significant proportion of the total cpu time and this nearly always outweighed the gain in speed for the final stage. Although this result was disappointing, it ties in with the findings of Section 4.6 where trying to extend the clique finding method to deal with larger molecules led to similar results.

8.2.5 The Structures Retrieved By The System

So far only the efficiency of the system has been considered, and so figures 8.12 to 8.19 and figures 8.20 to 8.23 show some of the molecules retrieved for pattern structures of BAGTOS and DPPRAM respectively. The molecules

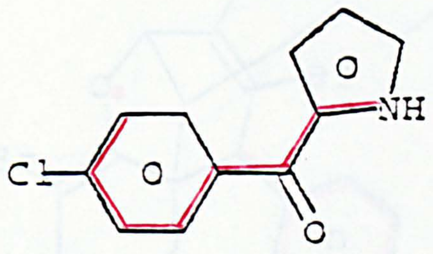


Figure 8.13 The Structure Diagram For The Molecule With Identifier BZPYRB

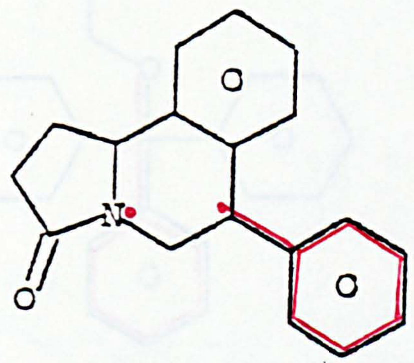
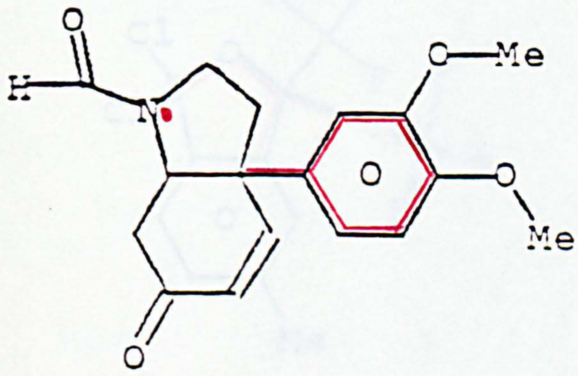


Figure 8.14 The Structure Diagram For The Molecule With Identifier CEDLUS



8.15 The Structure Diagram For The Molecule With Identifier DMFMES

The atoms marked in red belong to a common substructure.

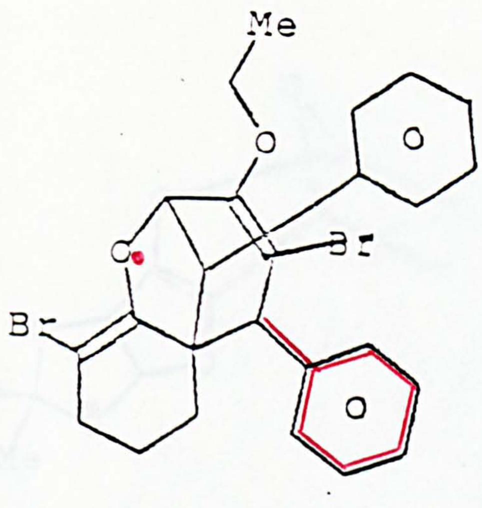


Figure 8.16 The Structure Diagram For The Molecule With Identifier EXBSUN

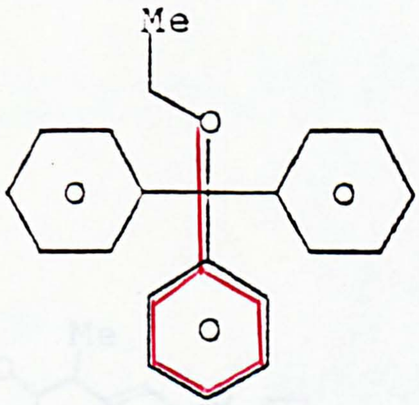
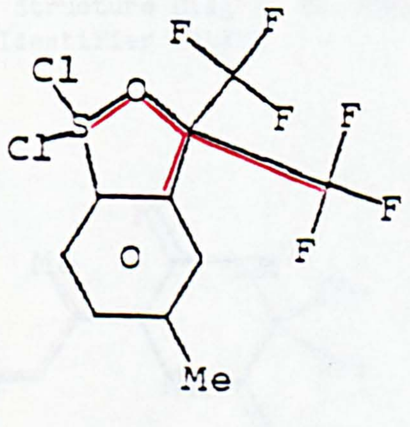


Figure 8.17 The Structure Diagram For The Molecule With Identifier PHETME



8.18 The Structure Diagram For The Molecule With Identifier CFMBXT

The atoms marked in red belong to a common substructure.

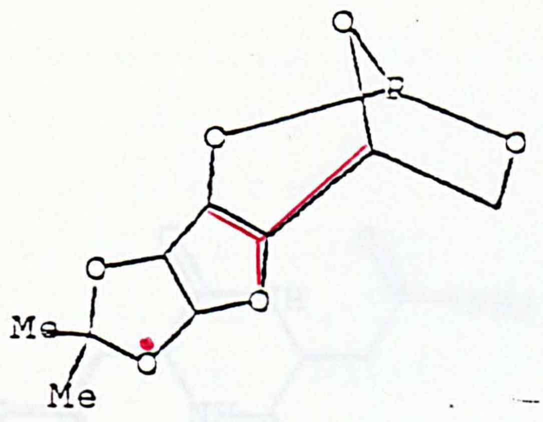


Figure 8.19 The Structure Diagram For The Molecule With Identifier PIPGFA

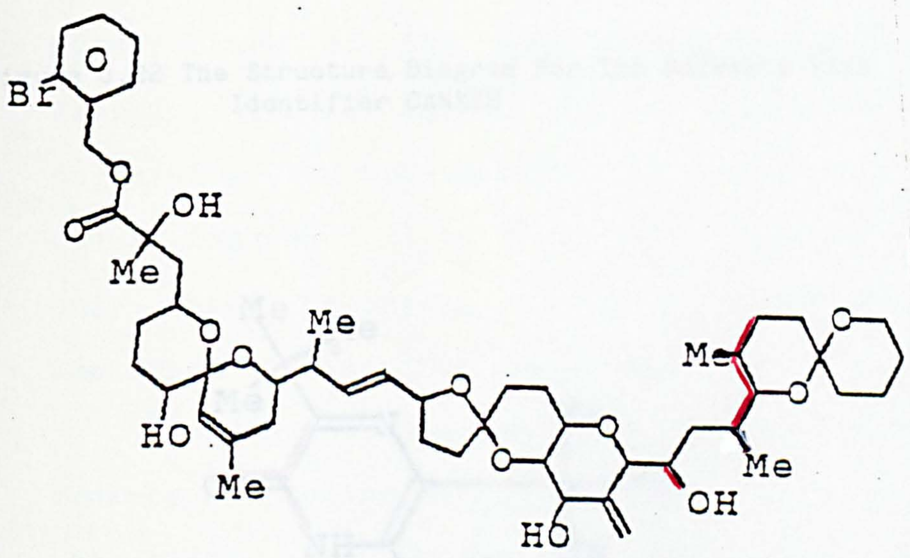


Figure 8.20 The Structure Diagram For The Molecule With Identifier BALKEE

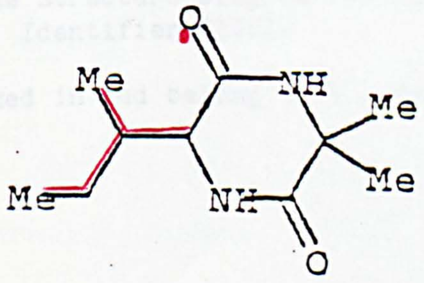


Figure 8.21 The Structure Diagram For The Molecule With Identifier BOCSOB

The atoms marked in red belong to a common substructure.

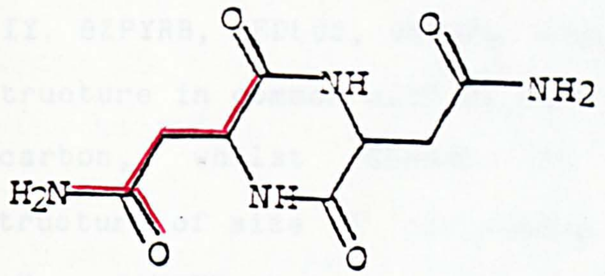


Figure 8.22 The Structure Diagram For The Molecule With Identifier CANKEH

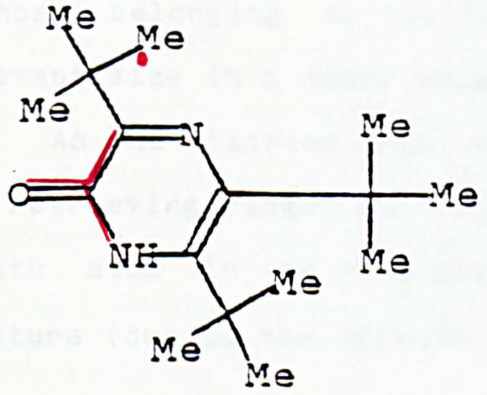


Figure 8.23 The Structure Diagram For The Molecule With Identifier TBDHZO

The atoms marked in red belong to a common substructure.

illustrated are those containing the largest common substructure with the pattern molecule when the minimum number of non-carbons was set to 1 and 2. More specifically BEWLIY, BZPYRB, CEDLUS, DMFMES, EXBSUN and PHETME have a substructure in common with BAGTOS of size 8 containing one non-carbon, whilst CFMBXT and PIPGFA have a common substructure of size 5 containing two non-carbons. For DPPRAM, CANKEH has a common substructure of size 6 containing two non-carbons while BALKEE and BOCSOB have a substructure of the same size containing one non-carbon. Finally, TBDHZO has a shared substructure of size 5 containing two non-carbons. The common atoms are marked in red in figures 8.12 to 8.23 (for reasons of legibility, only those belonging to the first common substructure of the relevant size in a query molecule are shown).

As the diagrams show, the the system has a bias towards retrieving rings, but figure 8.13 is of interest as the sixth atom in the ring does not belong to the common substructure (due to the effects of the error tolerance limit).

8.3 CONCLUSIONS

Part of the screen out obtained from the first of the screening stages could be provided by just checking that the query molecule does contain at least the required number of atoms of the correct type to meet the minimum number of non-carbon atoms in a common substructure condition. Table 8.8 gives this screen out figure for the molecules of table 8.6 (but with no check being carried out to see whether there are enough carbons in a molecule to allow clique sizes of, say, size 8 to be produced). Comparing the two tables seems to indicate that up to about half the screen out could be attributed to this factor, but generally it was significantly less than this. However, the initial screening stage takes relatively very little time, and so it is not unreasonable to eliminate molecules which do not contain enough atoms of the right type by using it.

The addition of an exact checking stage after the initial, screen-based stage in this work proved to be far less effective than this stage had been when used for 3D substructure searching (see Section 3.1) where it produced a significant improvement. This poorer performance was probably partly due to the pharmacophore s which were used in the substructure searching containing several non-carbons separated by several bond lengths. This meant that the distances between these atoms fell in regions where the screens were sparse.

To give some indication of the upper bound sizes the various stages produce and the actual common substructure sizes, tables 8.9 and 8.10 give some of the upper bounds produced by the exact distance checking stage and the intermediate stage along with the largest actual common substructure size when searching the database for structures similar to CAVROG with no restriction on the number of non-carbons. These show that even after the intermediate stage the upper bounds are quite often much greater than the largest common substructure size. (*It should be pointed out though that CAVROG was the compound from table 8.6 which led to the largest common substructures being found in the database.*)

Unfortunately, the net results produced by the searching system with its various screening stages, were not as large an improvement on using a system with no screening stage as had been hoped for, but as the size of the required common substructures increased, there was a significant improvement. However, a more efficient way of dealing with non-carbons would be to use a lexicographic clique finding algorithm where non-carbons are always chosen as the first elements of any potential clique, rather than the standard algorithm of Bron and Kerbosch.

It is not clear what structures would/should be used as a pattern, however the aim of this kind of best match searching is to provide an unbiased way of browsing through a database as an addition to the more usual searching facilities. This type of automated facility has the added value in three dimensions that humans find it very hard to visualize the extra dimension.

(*...*) A corrected version of this sentence is given in "Alterations".

Pattern And Its Position In The Database		Positions Compared With	Error Tolerance In A	Algorithm		
				BK	GR	LT
BARGOQ	101	1 to 100	0.05	1.81	20.06	3.19
CEGLCA	401	801 to 900	0.05	0.47	1.84	0.68
CEGLCA	401	801 to 900	0.15	0.49	2.45	0.62

Table 8.1 The Performance(*) Of The Three Clique Finding Algorithms On The Graph Produced From Considering Which Screens Are Set

BK is Bron and Kerbosch's algorithm
GR is Golender and Rozenblit's algorithm
LT is Loukakis and Tsouros' algorithm

Minimum Clique Size	Error Tolerance (In Angstroms)		
	0.05	0.15	0.25
4	213.0	239.9	259.3
6	192.4	232.2	257.1
8	142.7	218.0	250.4

Table 8.2 Times(*) For Searching The Database With No Restriction On The Number Of Non-Carbons And A Pattern Molecule Of BAGTOS

		Minimum Clique Size	Pattern Molecule		
			BAGTOS	CAVROG	ETCOHX
No Screening			240.5	701.5	543.8
Stage 1			2.8	8.6	8.2
STAGE TWO	No Restriction On Number Of Non-Carbons	4	239.9	700.1	540.4
		6	232.2	690.2	528.8
		8	218.0	672.3	502.4
	At Least One Non-Carbon	4	200.8	532.4	484.8
		6	194.6	524.4	476.7
		8	176.8	506.3	449.4
	At Least Two Non-Carbons	4	105.0	303.4	365.1
		6	103.4	302.5	361.7
		8	97.5	298.6	347.4

Table 8.3 Times(*) Taken For Searching The 999 Molecules Using An Error Tolerance Of 0.15 Angstroms

* The times are in cpu seconds for an IBM 3083

No Screening is the time when no screening stage is used.

	Minimum Clique Size	Pattern Molecule		
		BAGTOS	CAVROG	ETCOHX
No Restriction On Number Of Non-Carbons	4	24 (893)	19 (905)	11 (896)
	6	83 (488)	71 (503)	73 (137)
	8	272 (31)	161 (67)	222 (2)
At Least One Non-Carbon	4	102 (284)	117 (437)	76 (381)
	6	240 (22)	207 (91)	136 (28)
	8	410 (7)	321 (24)	278 (1)
At Least Two Non-Carbons	4	473 (19)	475 (107)	281 (117)
	6	575 (1)	520 (34)	325 (7)
	8	680 (1)	571 (22)	426 (1)

Table 8.4 The Number Of Molecules Eliminated From Consideration By Stage One

The number of molecules actually containing the required substructure are given in brackets.

		Pattern Molecule		
		GLYCIN	HMALAC	METNAM
	No Screening	14.6	40.0	15.1
	Stage One	0.6	1.2	0.7
STAGE TWO	No Restriction On The Number Of Non-Carbons	8.7	31.3	9.6
	At Least One Non-Carbon	8.7	31.2	9.5
	At Least Two Non-Carbons	8.7	25.4	9.5

Table 8.5 Times (*) For Comparing Smaller Molecules Against The Database With A Minimum Clique Size Of 4 And An Error Tolerance Of 0.15 Angstroms

* The cpu times are in seconds

No Screening is the time when no screening stage is used.

inimum Num Of Non Carbons	Minimum Clique Size	Time For 2nd Screening Stage	Time For Final Stage	Screen Out Total
BAGTOS				
0	6	4.6	231.1 (232.2)	112 (83)
1	6	4.1	184.4 (194.6)	295 (240)
2	6	2.0	42.7 (103.4)	833 (575)
0	8	3.5	210.3 (218.0)	331 (272)
0	10	1.9	111.3	735 (632)
CAVROG				
0	6	14.0	687.2 (690.2)	88 (71)
1	6	11.0	506.8 (524.4)	228 (207)
2	6	4.9	215.0 (302.5)	654 (520)
ETCOHX				
0	6	13.7	525.2 (528.8)	96 (73)
1	6	13.3	464.8 (476.7)	167 (136)
2	6	9.6	305.6 (361.7)	432 (325)
GLYCIN				
0	4	0.5	7.4 (8.7)	591 (500)
1	4	0.5	7.4 (8.7)	591 (500)
2	4	0.5	7.4 (8.7)	591 (500)
HMALAC				
0	4	2.0	30.4 (31.3)	286 (256)
1	4	2.1	30.4 (31.2)	286 (256)
2	4	1.6	23.3 (25.4)	515 (462)
METNAM				
0	4	0.7	7.6 (9.6)	588 (441)
1	4	0.7	7.6 (9.5)	588 (441)
2	4	0.7	7.6 (9.5)	588 (441)
DPPRAM				
0	5	3.7	225.6	124 (101)
1	5	2.9	169.3	223 (191)
2	5	0.8	26.2	822 (696)
NBENDC				
0	5	6.0	255.1	56 (43)
1	5	5.4	210.0	227 (215)
2	5	3.5	134.2	511 (427)
PRPENC				
0	5	4.5	235.0	92 (82)
1	5	4.0	190.3	198 (178)
2	5	2.2	102.4	550 (445)

Table 8.6 The Performance Of The Search System Using A Two Stage Screening System

The times are in cpu seconds for an IBM 3083.
The figures in brackets are for when only a one stage screening system was used (when the figures were available).

Minimum Num Of Non Carbons	Minimum Clique Size	Time For Intermediate Stage	Time For Final Stage	Screen Out Total
BAGTOS				
0	6	36.3	225.5 (231.1)	130 (112)
1	6	30.3	166.7 (184.4)	321 (295)
2	6	5.7	28.4 (42.7)	347 (333)
0	10	15.0	72.4 (111.3)	815 (735)
CAVROG				
0	6	360.6	679.2 (687.2)	106 (88)
1	6	274.2	454.9 (506.8)	260 (228)
2	6	104.2	143.8 (215.0)	662 (654)
ETCOHX				
0	6	160.4	515.8 (525.2)	116 (96)
1	6	154.7	440.3 (464.8)	189 (167)
2	6	107.4	256.6 (305.6)	466 (432)

Table 8.7 The Performance Of The Search System Using A Two Stage Screening System Along With The Intermediate Stage

The times are in cpu seconds for an IBM 3083.
The figures in brackets are for when only a two stage screening system was used.

	Minimum Number Of Non-Carbons	
	1	2
BAGTOS	68	161
CAVROG	70	197
ETCOHX	68	161
GLYCIN	70	227
HMALAC	178	313
METNAM	70	197
DPPRAM	70	557
NBENDC	178	313
PRPENC	70	227

Table 8.8 The Numbers Of Molecules Which Could Have Been Screened Out From The Searches Of Table 8.6 On An Analysis Of The Atomic Types That They Contain

Upper Bound For Size Of Common Substructure	Actual Size Of Substructure	Number Of Molecules
16	4	9
16	5	15
16	6	21
16	7	9
16	8	7
16	Above 9	3
17	4	2
17	5	9
17	6	5
17	7	15
17	8	4
18	8	1
18	9	2

Table 8.9 Some Of The Upper Bounds Produced By The Exact Distance Check Screening Stage For CAVROG With No Restriction On The Number Of Non-Carbons

Upper Bound For Size Of Common Substructure	Actual Size Of Substructure	Number Of Molecules
16	5	4
16	6	4
16	7	9
16	8	4
16	9	1
17	4	2
17	5	4
17	6	2
17	7	6
17	8	2
17	9	1

Table 8.10 Some Of The Upper Bounds Produced By The Intermediate Stage For CAVROG With No Restriction On The Number Of Non-Carbons

CHAPTER 9
SUMMARY

Two dimensional graph algorithms have a well established place in chemical information systems and as there is increasing interest in the use of three dimensions, there is a need for analogous algorithms here. This rising interest in 3D structural data has been based on the increased availability of 3D co-ordinates and improved computer performance generally, and especially in molecular graphics systems. This thesis has been concerned with techniques for handling 3D chemical information with a particular emphasis being placed upon algorithms for identifying and searching for pharmacophores. The stress throughout the thesis has been placed on the efficiency of the algorithms rather than their effectiveness in operational environments.

In greater detail, Chapter 3 described Jakes' screening system [Jake86] based on inter-atomic distances which allows 3D substructure searches to be carried out on the Cambridge Crystallographic Database. The structures which pass the screening stage are passed on to a partial matching stage and four algorithms for this stage were compared. On the results of the tests performed, Ullman's subgraph isomorphism algorithm [Ullm76] was substantially quicker than the other methods. Although this should probably be regarded as recommending subgraph isomorphism algorithms from the computer science literature (as opposed to specifically chemical algorithms) as a whole rather than as a an endorsement of Ullman's algorithm in particular. However, the patterns searched for were very artificial being obtained by taking some of the structure's atoms (mainly carbons) and distorting them slightly. Another major criticism that could be levelled at the work is that the performances of all of the algorithms were pretty good, and the time taken in this stage is always likely to be substantially less than the time taken by the disc accessing in the screening stage. As a follow on from this work two of the methods were used to search a macromolecule and the results implied that Lesk's algorithm [Lesk79] was the only one of the four techniques suitable for the task. (*However, its run times were very large and so work is currently being undertaken in the department to try to lessen this by using Lesk's algorithm to reduce the number of atoms being passed on to Ullman's algorithm and/or having an initial screening stage which restricts which structure atoms can match which pattern atoms [Davi87].*)

Chapter 4 compared two different methods for finding the 3D substructures in common between two molecules. The clique finding approach was found to be far more efficient, particularly when there was a sizeable common substructure. This method was extended to deal with more than two molecules at a time and the results suggested that the time taken was very roughly linearly related to the number of molecules. However, the same kind of criticism that was made above can be made again here in

(*...*) A corrected version of the marked lines is given in "Alterations."

that the structures being compared were very artificial. Additionally, the clique finding program was unable to cope with structures of size greater than about 35 atoms. Additional work could attempt to devise a method of coping with larger structures but finding a successful solution is likely to be difficult as can be seen from the poor performance of Bolles' suggestion [Boll79].

Following on from Chapter 4, the next two chapters considered a parallel implementation of a version of Crandell and Smith's algorithm first through a simulation and then through an actual implementation. The simulation was only intended as a fairly crude measure of the potential speed up that a multiprocessor system might offer, so as to give some indication as to whether a full implementation would be worthwhile. The main problems with it were difficulties over trying to estimate the distributions of the times each stage would take when they were split up into several jobs, the use of a Prime 9950 system clock rather than a system clock comparable with that of a transputer, and the fact that for simplicity, a pooled processor system with each processor carrying out identical programs was assumed. Therefore it was not possible to really correlate the actual figures produced by the simulation and those produced using transputers. The latter results showed that a near linear speed up could be produced for eleven transputers with regard to the comparison stage if, and only if, the stage involved enough computation. Unfortunately, the results are rather academic as this is often not the case and as the implemented version of Crandell and Smith's algorithm performed very badly in Chapter 4, both against a version which had an extra sorting stage and against the clique finding algorithm. However, a version of Crandell and Smith using exact distances rather than clustered distances could be a way of dealing with the problem of comparing larger molecules. This would lead to a computationally more expensive comparison stage but this is the stage which can be effectively parallelized. Additionally, if distance ranges are used, as suggested in [Cran83a], then the more efficient form of the algorithm, which sorts the grown structures before comparing them, becomes less practicable.

The academic nature of the parallel implementation of Crandell and Smith's algorithm led to interest in applying concurrency to try to discover pharmacophoric patterns via Crippen's distance geometry [Sher86]. This method generates a series of "conformations" of a "molecule" formed from the molecules under investigation by way of different sets of random numbers and then compares the proposed pharmacophoric regions using a least squares fitting routine. The computational expense of this approach can be seen in the time of 3.5 cpu hours for a VAX 11/780 quoted for finding 25 conformations by [Sher86], and so generating each conformation on a different processor was thought likely to lead to a significant improvement in performance. The work carried out in Chapter 7 seemed to confirm this but the main figures were produced by repeatedly running the algorithm on a serial computer rather than running the system once on a multiprocessor. A version of the algorithm was run on a transputer system but it performed very badly due to the floating point operations being carried out by software subroutines. This version also showed that it would be very difficult with the current algorithm to substantially improve the performance by using a cluster of transputers instead of a single one to generate each conformation. A way around this might be to use a different optimisation procedure other than the conjugate gradient method. Another potential area of interest would be to compare the distance geometry approach to pharmacophore identification with that proposed by Motoc et al. [Moto86, Laba86] (see Section 2.3.2) as they are both intended for use in the same kind of environment.

Finally, the 3D screening system of Chapter 3 was combined with the clique finding algorithm of Chapter 4 so as to try to create an efficient means of searching the Cambridge Crystallographic Database in order to find 3D structurally similar molecules to the starting molecule. The effectiveness of the "screening" stage was found to be heavily dependent on whether a minimum number of non-carbons was specified to be in the common substructure. In cases where this value was set to zero and the minimum common substructure size of interest was set to four, relatively little speed up was obtained, but this was very much a "worst case" situation.

To briefly summarize the main numerical results of this thesis, the tests reported in Chapter 3 indicated that Ullman's algorithm was generally at least twice as quick as the other algorithms when trying to find a pattern of size 5 in a molecule. This better preformance increased as the size of the pattern increased.

The comparison of the two algorithms for finding common substructures between molecules in Chapter 4 showed that the clique finding approach was at least twice as quick as the fastest version of Crandell and Smith's algorithm when finding a largest common substructure of size 7 between two distorted versions of the same molecule. This difference in performance very rapidly widened as the size of the largest common substructure increased. A similar situation occurred when more than two molecules were being considered. Finally, when a molecule was being compared with a collection of similar molecules, the clique finding approach was usually a minimum of 3 times quicker and sometimes had a far greater speed advantage than this. An additional advantage of using cliques was that the distance clustering stage (which caused severe problems for Crandell and Smith) was no longer necessary.

The simulation of a multiprocessor version of Crandell and Smith's algorithm in Chapter 5 predicted that when there was a largest common substructure of size 12 between two molecules, a speed up of 8 would be obtained when using 50 transputers (and assuming no processor overheads). In the same case but using 5, 10 and 20 transputers predicted speed ups were 2.5, 3.3 and 5.4

respectively. In the actual implementation, when there was a largest common substructure of size 12 between two molecules of size 14, the speed ups for 5 and 10 transputers were 4.98 and 7.72. These figures are probably artificially inflated because breaking up the comparison stage into pieces led to a reordering of the "growths" which in turn led to an increase in speed. Therefore the speed ups over the two transputer case could well be more realistic and these were 2.28 and 3.54 respectively. However, as the molecules became less similar the amount of processing needed decreased and the resulting speed ups became much smaller.

No overall speed up figures can reasonably be quoted for the implementation of distance geometry on transputers as the lack of transputers with floating point hardware and not being able to obtain the standard bond angles for the molecules used in [Sher86] meant no comparison with the 3.5 hours of cpu time used by a VAX 11/785 could be undertaken. However, the fact that using transputers to find all the shortest paths in a graph led to no speed up on the graphs considered whilst Denelcor's HEP has been reported as having very high speed ups (virtually linear for eight processors) for graphs of the same size [Deo86] seemed to indicate that transputers are not suited to very fine grain parallelism.

Chapter 8 involved comparing a molecule against 999 molecules in the Cambridge Crystallographic Database so as to find three dimensionally similar molecules. There was a large variation in the times taken using different pattern molecules. However, to give some indication of the general performance, ETCOHX (see figure 8.5) took about 480 cpu seconds on an IBM 3083 when using a cut off for the first screening stage of a predicted largest common substructure size of at least 6 atoms of which at least one had to be a non-carbon. However, this time could be almost halved if the optimise option is used on the FORTRAN 77 compiler.

Number Of	Overhead Added To A Process' Duration Time		
Processors	0	.001*I	.01*I
1	1.00	1.00	1.00
2	1.74	1.74	1.70
3	2.43	2.43	2.33
4	3.20	3.20	3.03
5	3.52	3.52	3.26
6	3.70	3.64	3.42
8	4.40	4.22	3.59
10	5.15	5.02	4.08
12	5.86	5.55	4.16
16	7.03	6.59	4.51
20	7.81	7.28	4.61
30	9.59	8.44	4.33
40	11.11	9.17	3.85
50	12.41	9.59	3.48

Table 5.6 The Speed Ups Obtained By The Simulation Of Comparing Three Molecules With A Common Substructure Size Of 11

where I is the number of processors

Apart from Chapter 7, all the work reported in this thesis has dealt with fixed (crystallographic) conformations and because of this drawback, can only be used at a very early stage in the computer assisted drug design process. However, future work might involve trying to incorporate the 3D comparison algorithms into one of the various QSAR methods, possibly along the lines suggested by Motoc [Moto81].

The work summarised above has indicated algorithms which appear to be sufficiently efficient for practical implementations in 3D chemical information systems. It is hoped that practical tests of these ideas will follow shortly.

REFERENCES

[Abol85] E.E.Abola, F.C.Bernstein, T.F.Koetzle; "The Protein Data Bank"; pp.139-144 in "The Role Of Data In Scientific Progress"; Editor: P.S.Glaesen; Proceedings Of The 9th International CODATA Conference Elsevier (North Holland), 1985

[Adam71] G.W.Adamson, M.F.Lynch, W.G.Town; "Analysis Of Structural Characteristics Of Chemical Compounds In A Large Computer Based File 2 Atom-Centered Fragments"; Journal Of The Chemical Society (C), (1971) pp.3702-3706

[Adam73a] G.W.Adamson, J.Cowell, M.F.Lynch, A.H.W.Mclure, W.G.Town, A.M.Yapp; "Strategic Considerations In The Design Of A Screening System For Substructure Searches Of Chemical Structure Files"; Journal Of Chemical Documentation, vol 13 num 3 (1973) pp.153-157

[Adam73b] G.W.Adamson, S.E.Creasey, M.F.Lynch; "Analysis Of Structural Characteristics Of Chemical Compounds In The Common Data Base"; Journal Of Chemical Documentation, vol 13 num 3 (1973) pp.158-162

[Adam73c] G.W.Adamson, V.A.Clinch, M.F.Lynch; "Relationship Between Query And Data Base Microstructure In General Substructure Search Systems"; Journal Of Chemical Documentation, vol 13 num 3 (1973) pp.133-136

[Adam74] G.W.Adamson, J.A.Bush; "Method For Relating The Structure And Properties Of Chemical Compounds"; Nature, vol 248 (1974) pp.406-407

[Adam77] G.W.Adamson, D.Bawden; "A Substructural Analysis Method For Structure-Activity Correlation Of Heterocyclic Compounds Using Wiswesser Line Notation"; Journal Of Chemical Information And Computer Sciences, vol 17 num 3 (1977) pp.164-171

[Aho74] A.Aho, J.E.Hopcroft, J.Ullman; "The Design And Analysis Of Computer Algorithms"; Addison-Wesley, 1974

[Alle79] F.H.Allen, S.Bellard, M.D.Brice, B.A.Cartwright, A.Doubleday, H.Higgs, T.Hummelink, B.G.Hummelink-Peters, O.Kennard, W.D.S.Motherwell, J.R.Rodgers, D.G.Watson; "The Cambridge Crystallographic Data Centre: Computer-Based Search, Retrieval, Analysis And Display Of Information"; Acta Crystallographica B, vol 35 (1979) pp.2331-2339

[Almo82] J.R.Almond, H.M.Welsh; "Chemical Substructure Searching - Industrial Applications And Commercial Systems"; Drexel Library Quarterly, vol 18 num 2 (1982) pp.84-105

[Ash75] J.E.Ash; "Connection Tables And Their Role In A System"; Chapter 11 in "Chemical Information Systems"; Editors: J.Ash, E.Hyde; Ellis Horwood, Chichester, 1975

[Ash85] J.Ash, P.Chubb, S.Ward, S.Welford, P.Willett; "Communication, Storage And Retrieval Of Chemical Information"; Ellis Horwood, Chichester, 1985

[Aspi84] D.Aspinall; "Closely Coupled Systems" (a) "Architecture", Chapter 15 pp.219-229, (b) "Cyba-M", Chapter 19 pp.267-276 in "Distributed Computing"; Editors: F.B.Chambers, D.A.Duce, G.P.Jones; Academic Press, London, 1984

[Atki83] L.V.Atkinson, P.J.Harley; "An Introduction To Numerical Methods With Pascal"; Addison Wesley, London, 1983

[Aust84] V.Austel; "Drug Design: Principles And Techniques"; Chapter 24 pp.441-460 in "X-Ray Crystallography And Drug Action"; Editors: A.S.Horn, C.J.De Ranter Oxford University Press, Oxford, 1984

[Avid82] V.Avidon, I.A.Pomerantsev, V.E.Golender, A.B.Rozenblit; "Structure-Activity Relationship Oriented Languages For Chemical Structure Representation"; Journal Of Chemical Information And Computer Sciences, vol 22 num 4 (1982) pp.207-214

[Bari81] L.Barino; "Use Of A Least-Squares Best Molecular Fit Routine In A Steric Comparison Of Flexible Molecules"; Computers And Chemistry, vol 5 num 2-3 (1981) pp.85-90

[Barr76] H.G.Barrow, R.M.Burstall; "Subgraph Isomorphism, Matching Relational Structures And Maximal Cliques"; Information Processing Letters, vol 4 num 4 (1976) pp.83-84

[Barr81] H.G.Barrow, J.M.Tenebaum; "Computational Vision"; Proceedings Of The IEEE, vol 69 num 5 (1981) pp.572-595

[Barr86] I.M.Barron; "The Transputer And Occam" in "Information Processing 86"; Editor: H.-J. Kugler; Elsevier Science Publishers B.V. (North-Holland) IFIP (pp.259-265 in Participants Edition) 1986

[Bawd81] D.Bawden, J.T.Catlow, T.K.Devon, J.M.Dalton, M.F.Lynch, P.Willett "Evaluation And Implementation Of Topological Codes For Online Compound Search And Registration"; Journal Of Chemical Information And Computer Sciences, vol 21 num 2 (1981) pp.83-86

[Bawd83] D.Bawden; "Computerized Chemical Structure-Handling Techniques In Structure-Activity Studies And Molecular Property Prediction"; Journal Of Chemical Information And Computer Sciences, vol 23 num 1 (1983) pp.14-22

[Baza79] M.S.Bazaraa, C.M.Shetty; "Nonlinear Programming: Theory And Algorithms"; Wiley, Chichester, 1979

[Bern77] F.C.Bernstein, T.F.Koetzle, G.J.B.Williams, E.F.Meyer, M.D.Brice, J.R.Rodgers, O.Kennard, T.Shimanouchi, M.Tasumi; "The Protein Data Bank : A Computer-Based Archival File For Macromolecular Structures"; Journal Of Molecular Biology, vol 112 num 3 (1977) pp.535-542

[Birt73] G.M.Birtwistle, O-J.Dahl, B.Myhrhaug, K.Nygaard; "SIMULA Begin"; Auerbach Publishers Inc., Philadelphia, 1973

[Boll79] R.C.Bolles; "Robust Feature Matching Through Maximal Cliques"; Proceedings Of The Society Of Photo-Optical Instrument Engineers, vol 182 (1979) pp.140-149

[Boll82] R.C.Bolles, R.A.Cain; "Recognising And Locating Partially Visible Objects: The Local-Feature-Focus Method"; International Journal Of Robotics Research, vol 1 num 3 (1982) pp.57-82

[Boyd82] D.B.Boyd, K.B.Lipkowitz; "Molecular Mechanics: The Method And Its Underlying Philosophy"; Journal Of Chemical Education, volume 59 num 4 (1982) pp.269-274

[Boyd83] D.B.Boyd; "Quantum Mechanics In Drug Design: Methods And Applications"; Drug Information Journal, vol 17 (1983) pp.121-131

[Bron73] C.Bron, J.Kerbosch; "Algorithm 457 Finding All Cliques Of An Undirected Graph [H]"; Communications Of The ACM, vol 16 num 9 (1973) pp.575-577

[Bruc87] P.Bruck; "Retrieval Of Substructures From Very Large Files Using Tree-Structured Databases"; in "Proceedings Of Chemical Structures: The International Language Of Chemistry"; Chemical Structures Association (In Preparation)

[Burg75] A.S.V.Burgen, G.C.K.Roberts, J.Feeney; "Binding Of Flexible Ligands To Macromolecules"; Nature, vol 253 (1975) pp.753-755

[Buse83] B.Busetta, I.J.Tickle, T.L.Blundell; "DOCKER, An Interactive Program For Simulating Protein Receptor And Substrate Interactions"; Journal Of Applied Crystallography, vol 16 (1983) pp.432-437

[Cahn79] R.S.Cahn, C.O.Dermer; "Introduction To Chemical Nomenclature"; 5th Edition, Butterworths, London 1979

[Caro84] A.Carotti, C.Hansch, M.M.Mueller, J.M.Blaney; "Actinidin Hydrolysis Of Substituted- Phenyl Hippurates: A Quantitative Structure- Activity Relationship And Graphics Comparison With Hydrolysis By Papain"; Journal Of Medicinal Chemistry, vol 27 num 11 (1984) pp.1401-1405

[Chen81] J.K.Cheng, T.S.Huang; "A Subgraph Isomorphism Algorithm Using Resolution"; Pattern Recognition, vol 13 num 5 (1981) pp.371-379

[Cohe79] N.C.Cohen; "Beyond The 2-D Chemical Structure"; Chapter 18 pp.371-381 in "Computer Assisted Drug Design"; Editors: E.C.Olson, R.E.Christoffersen; American Chemical Society, Washington D.C., 1979

[Cohe85] N.C.Cohen; "Drug Design In Three Dimensions"; Advances In Drug Research, vol 14 (1985) pp.41-145

[Cone77] M.M.Cone, R.Venkataraghavan, F.W.McLafferty; "Molecular Structure Comparison Program For The Identification Of Maximal Common Substructures"; Journal Of The American Chemical Society, vol 99 num 23 (1977) pp.7668-7671

[Crai75] P.N.Craig; "Structure/Property Correlations"; Chapter 16 pp.259-268 in "Chemical Information Systems"; Editors: J.Ash, E.Hyde; Ellis Horwood, Chichester, 1975

[Cram74] R.D.Cramer, G.Redl, C.E.Berkoff; "Substructural Analysis: A Novel Approach To The problem Of Drug Design"; Journal Of Medicinal Chemistry, vol 17 num 5 (1974) pp.533-535

[Cran83a] C.W.Crandell, D.H.Smith; "Computer-Assisted Examination Of Compounds For Common Three-Dimensional Substructures"; Journal Of Chemical Information And Computer Sciences, vol 23 num 4 (1983) pp.186-197

[Cran83b] C.W.Crandell; "Computers In Chemistry"; Ph.D. Thesis, University Of Stanford, California, 1983

[Crip79a] G.M.Crippen; "Distance Geometry Approach To Rationalizing Binding Data"; Journal Of Medicinal Chemistry, vol 22 num 8 (1979) pp.988-997

[Crip79b] G.M.Crippen; "Distance Constraints On Macromolecular Conformation 1 The Effectiveness Of The Experimental Studies On The Tobacco Mosaic Virus Protein"; International Journal Of Peptide And Protein Research, vol 13 (1979) pp.320-326

[Crip80] G.M.Crippen; "Quantitative Structure-Activity Relationships By Distance Geometry: A Systematic Analysis Of Dihydrofolate Inhibitors"; Journal Of Medicinal Chemistry, vol 23 num 6 (1980) pp.599-606

[Crip81] G.M.Crippen; "Distance Geometry And Conformational Calculations"; Research Studies Press, Wiley, New York, 1981

[Dahl72] O.-J.Dahl, E.W.Dijkstra, C.A.R.Hoare; "Structured Programming"; Academic Press, London, 1972

[Danz85] D.J.Danziger, P.M.Dean; "The Search For Functional Correspondences In Molecular Structure Between Two Dissimilar Molecules"; Journal Of Theoretical Biology, vol 116 num 2 (1985) pp.215-224

[Das78] S.R.Das, C.L.Sheng, Z.Chen; "An Algorithm For Finding All Maximal Complete Subgraphs And An Estimate Of The Order Of The Computational Complexity"; Comput. And Elect. Engineering, vol 5 (1978) pp.365-368

[Davi81] L.S.Davis, A.Rosenfeld; "Cooperating Processes For Low-Level Vision: A Survey"; Artificial Intelligence, vol 17 (1981) pp.245-263

[Davi87] H.Davies; M.Sc. Thesis, Department Of Information Studies, University Of Sheffield, (In Preparation)

[Denn86] P.J.Denning; "Parallel Computing And Its Evolution"; Communications Of The ACM, vol 29 num 12 (1986) pp.1163-1167

[Deo74] N.Deo; "Graph Theory With Applications To Engineering And Computer Science"; Prentice-Hall, Englewood Cliffs, New Jersey, 1974

[Deo80] N.Deo, C.Y.Pang, R.E.Lord; "Two Parallel Algorithms For Shortest Path Problems"; Proceedings Of The 1980 International Conference On Parallel Processing (IEEE), New York, pp.244-253, 1980

[DeRa84] C.J.De Ranter; "Crystals, X-ray Crystallography, And Drugs"; Chapter 1 pp.1-22 in "X-ray Crystallography And Drug Action"; Editors: A.S.Horn, C.J.De Ranter; Oxford University Press, Oxford, 1984

[Dijk68] E.W.Dijkstra; "Go To Statement Considered Harmful"; Communications Of The ACM, vol 11 num 3 (1968) pp.147-148

[Ditt83] P.G.Dittmar, N.A.Farmer, W.Fisanick, R.C.Haines, J.Mockus; "The CAS ONLINE Search System 1 General System Design And Selection, Generation And Use Of Search Screens"; Journal Of Chemical Information And Computer Sciences, vol 23 num 3 (1983) pp.93-102

[Drey69] S.E.Dreyfus; "An Appraisal Of Some Shortest-Path Algorithms"; Operations Research, vol 17 num 3 (1969) pp.395-412

[Duch79] D.J.Duchamp; "Molecular Mechanics And Crystal Structure Analysis In Drug Design"; Chapter 3 pp.79-102 in "Computer Assisted Drug Design"; Editors: E.C.Olsen, R.C.Christoffersen; American Chemical Society, Washington D.C., 1979

[Elde84] M.Elder, P.Machin, S.E.Hull; "CDA: An Interactive Program For The Comparative Analysis Of Crystal Structure Co-ordinate Data"; Journal Of Molecular Graphics, vol 2 num 3 (1984) pp.70-78

[Esak82] T.Esaki; "Quantitative Drug Design Studies 5 Approach To Lead Generation By Pharmacophoric Pattern Searching"; Chemical And Pharmaceutical Bulletin, vol 30 num 10 (1982) pp.3657-3661

[Esak83] T.Esaki; "Fundamental Studies On Quantitative Drug Design"; Ph.D.Thesis, Nagoya University, Japan, 1983

[Faug82] O.D.Faugeras; "Image Understanding And Graph Matching"; Proceedings IEEE International Conference On Acoustics, Speech And Signal Processing, vol 2 (1982) pp.1162-1165

[Feld75] A.Feldman, L.Hodes; "An Efficient Design For Chemical Structure searching, 1. The Screens"; Journal Of Chemical Information And Computer Sciences, vol 15 num 3 (1975) pp.147-152

[Feld78] R.J.Feldman, D.H.Bing, B.C.Furie, B.Furie; "Interactive Computer Surface Graphics Approach To Study Of The Active Site Of Bovine Trypsin"; Proceedings Of The National Academy Of Science USA, vol 75 num 11 (1978) pp.5409-5412

[Feld79] A.Feldman, L.Hodes; "Substructure Search With Queries Of Varying Specifity"; Journal Of Chemical Information And Computer Sciences, vol 19 num 3 (1979) pp.125-129

[Figu72] J.Figueras "Substructure Search By Set Reduction" Journal Of Chemical Documentation, vol 12 num 4 (1972) pp.237-244

[Fish85] A.J.Fisher; "Occam On The Prime - Users' Manual (Revised)"; Department Of Computer Science, University of Hull, 1985

[Fish86] A.J.Fisher; "A Multiprocessor Implementation Of Occam"; Software Practice And Experience, vol 16 num 10 (1986) pp.875-892

[Floy62] R.W.Floyd; "Algorithm 97 Shortest Path"; Communications Of The ACM, vol 5 num 6 (1962) p.345

[Fran84] R.Franke; "Theoretical Drug Design Methods"; Elsevier, Oxford, 1984

[Gare79] M.R.Garey, D.S.Johnson; "Computers And Intractability: A Guide To The Theory Of NP-Completeness"; Freeman, New York, 1979

[Gati79] G.Gati; "Further Annotated Bibliography On The Isomorphism Disease"; Journal Of Graph Theory, vol 3 num 2 (1979) pp.95-109

[Gaud86] J.L.Gaudiot, M.Dubois, L.T.Lee, N.G.Tohme; "The TX16: A Highly Programmable Multi-Processor Architecture"; IEEE Micro, vol 6 num 5 (1986) pp.18-31

[Gerh79] L.Gerhards, W.Lindenberg; "Clique Detection For Nondirected Graphs: Two New Algorithms"; Computing, vol 21 (1979) pp.295-322

[Ghos85] A.K.Ghose, G.M.Crippen; "Geometrically Feasible Binding Modes Of A Flexible Ligand Molecule At The Receptor Site"; Journal Of Computational Chemistry, vol 6 num 5 (1985) pp.350-359

[Gill86] V.J.Gillet, S.M.Welford, M.F.Lynch, P.Willett, J.Barnard, G.M.Downs, G.Manson, J.Thompson; "Computer Storage And Retrieval Of Generic Chemical Structures In Patents 7 Parallel Simulation Of A Relaxation Algorithm For Chemical Substructure Search"; Journal Of Chemical Information And Computer Sciences, vol 26 num 3 (1986) pp.118-126

[Gole83] V.Golender, A.Rozenblit; "Logical And Combinatorial Algorithms For Drug Design"; Research Studies Press, Letchworth, 1983

[Gost81] R.W.Gostick; "Software And Hardware Technology For The ICL Distributed Array Processor"; Australian Computer Journal, vol 13 num 1 (1981) pp.1-6

[Gour84] D.R.H.Gourley; "Receptors- A Review Of Recent Progress"; Chapter 6 pp.95-112 in "X-ray Crystallography And Drug Action"; Editors: A.S.Horn, C.J.De Ranter; Oxford University Press, Oxford, 1984

[Gund74] P.Gund, W.T.Wipke, R.Langridge; "Computer Searching Of A Molecular Structure File For Pharmacophoric Patterns"; Proceedings International Conference Computers In Chemical Research And Education, Ljubljana, July 12-17 1973, vol 3 (1974) pp.33-38

[Gund77] P.Gund; "Three-Dimensional Pharmacophoric Pattern Searching"; Progress In Molecular And Subcellular Biology, vol 5 (1977) pp.117-143

[Gund79] P.Gund; "Pharmacophoric Pattern Searching And Receptor Mapping"; Annual Reports In Medicinal Chemistry, vol 14 chapter 29 (1979) pp.299-308

[Gund80] P.Gund, J.D.Andose, J.B.Rhodes, G.M.Smith; "Three Dimensional Molecular Modelling And Drug Design"; Science, vol 208 num 4451 (1980) pp.1425-1431

[Gurd85] J.R.Gurd, C.C.Kirkham, I.Watson; "The Manchester Prototype Dataflow Computer"; Communications Of The ACM, vol 28 num 1 (1985) pp.34-52

[Gurd86] J.R.Gurd, C.C.Kirkham; "Dataflow: Achievements And Prospects" in "Information Processing 86"; Editor: H.-J. Kugler; Elsevier Science Publishers B.V. (North-Holland) IFIP (pp.61-68 in Participants Edition) 1986

[Hans73] C.Hansch, J.M.Clayton; "Lipophilic Character And Biological Activity Of Drugs .II. The Parabolic Case"; Journal Of Pharmaceutical Science, vol 62 num 1 (1973) pp.1-21

[Hans82] C.Hansch, Ren-li Li, J.M.Blaney, R.Langridge; "Comparison Of The Inhibition Of Escherichia Coli And Lactobacillus Casei Dihydrofolate Reductase By 2,4-Diamino- 5- (Substituted- Benzyl) Pyrimidines: Quantitative Structure- Activity Relationships, X-ray Crystallography, And Computer Graphics In Structure- Activity Analysis"; Journal Of Medicinal Chemistry, vol 25 num 7 (1982) pp.777-784

[Hans84] C.Hansch; "On The State Of QSAR"; Drug Information Journal, vol 18 (1984) pp.115-122

[Hass85] C.H.Hassel; "Computer Graphics As An Aid To Drug Design"; Chemistry In Britain, vol 21 num 1 (1985) pp.39-46

[Have79] T.F.Havel, G.M.Crippen, I.D.Kuntz; "Effects Of Distance Constraints On Macromolecular Conformation 2 Simulation Of Experimental Results And Theoretical Predictions"; Biopolymers, vol 18 num 1 (1979) pp.73-81

[Have82] T.F.Havel; "The Combinatorial Distance Geometry Approach To The Calculation Of Molecular Conformation"; Ph.D. Thesis, University Of California, Berkely, 1982

[Have83] T.F.Havel, I.D.Kuntz, G.M.Crippen; "The Theory And Practice Of Distance Geometry"; Bulletin Of Mathematical Biology, vol 45 num 5 (1983) pp.665-720

[Have84] T.F.Havel, K.Wuthrich; "A Distance Geometry Program For Determining The Structures Of Small Proteins And Other Macromolecules From Nuclear Magnetic Resonance Measurements Of Intramolecular 1H-1H Proximities In Solution"; Bulletin Of Mathematical Biology, vol 46 num 4 (1984) pp.673-698

[Haye86] J.P.Hayes, T.Mudge, Q.F.Stout, S.Colley, J.Palmer; "A Microprocessor-Based Hypercube Supercomputer"; IEEE Micro, vol 6 num 5 (1986) pp.6-17

[Hayn82] L.S.Haynes, R.L.Lau, D.P.Siewiorek, D.W.Mizeu; "A Survey Of Highly Parallel Computing"; Computer, vol 15 num 1 (1982) pp.9-24

[Hend86] J.B.Hendrickson, A.G.Toczko; "Systematic Synthesis Design By Computer" Chapter 9 pp.86-97 in "Mathematics And Computational Concepts In Chemistry"; Editor: N.Trinajstic; Ellis Horwood, Chichester, 1986

[Hiro86] R.Hiromoto; "Some Issues In Parallel Processing As Encountered On The Denelcor HEP"; Parallel Computing, vol 3 num 2 (1986) pp.111-127

[Hock85] R.W.Hockney; "MIMD Computing In The USA -1984"; Parallel Computing, vol 2 num 2 (1985) pp.119-136

[Hode77] L.Hodes, G.F.Hazard, R.I.Geran, S.Richman; "A Statistical Heuristic Method For Automated Selection Of Drugs For Screening"; Journal Of Medicinal Chemistry, vol 20 num 4 (1977) pp.469-475

[Hode81] L.Hodes; "Computer-Aided Selection Of Compounds For Antitumour Screening: Validation Of A Statistical-Heuristic Method"; Journal Of Chemical Information And Computer Sciences, vol 21 num 3 (1981) pp.128-132

[Hopf80] A.J.Hopfinger; "A QSAR Investigation Of Dihydrofolate Reductase Inhibition By Baker Triazines Based Upon Molecular Shape Analysis"; Journal Of The American Chemical Society, vol 102 num 24 (1980) pp.7196-7206

[Hopf85] A.J.Hopfinger; "Computer-Assisted Drug Design"; Journal Of Medicinal Chemistry, vol 28 num 9 (1985) pp.1133-1139

[Hori86] H.Horikoshi, Y.Inagami; "Dataflow: From Its Practical Viewpoints" in "Information Processing 86"; Editor: H.-J. Kugler; Elsevier Science Publishers B.V. (North-Holland) IFIP (pp.69-72 in Participants Edition) 1986

[Horn84] A.S.Horn; "Conformationally Restricted Analogues Of Neurotransmitters And Drugs"; Chapter 14 pp.235-255 in "X-ray Crystallography And Drug Action"; Editors: A.S.Horn, C.J.De Ranter; Oxford University Press, Oxford, 1984

[Humb80] C.Humblet, G.R.Marshall; "Pharmacophoric Identification And Receptor Mapping"; Annual Reports In Medicinal Chemistry, vol 15 chapter 28 (1980) pp.267-276

[Humb81] C.Humblet, G.R.Marshall; "Three-Dimensional Computer Modelling As An Aid To Drug Design"; Drug Development Research, vol 1 num 4 (1981) pp.409-434

[Hwan84] K.Hwang, F.A.Briggs; "Computer Architecture And Parallel Processing"; McGraw-Hill, New York, 1984

[INMO84] INMOS Limited; "Occam Programming Manual"; Prentice-Hall, London, 1984

[INMO85] INMOS Limited; "Transputer Reference Manual"; 1985

[Isen86] D.K.Isenor, S.G.Zaky; "Fingerprint Identification Using Graph Matching"; Pattern Recognition, vol 19 num 2 (1986) pp.113-122

[Iwas85] K.Iwase, K.Komatsu, S.Hirono, S.Nakagawa, I.Moriguchi; "Estimation Of Hydrophobicity Based On The Solvent-Accessible Surface Area Of Molecules"; Chemical And Pharmaceutical Bulletin, vol 33 num 5 (1985) pp.2114-2121

[Jake86] S.E.Jakes, P.Willett; "Pharmacophoric Pattern Matching In Files Of 3D Chemical Structures: Selection Of Interatomic Distance Screens"; Journal Of Molecular Graphics, vol 4 num 1 (1986) pp.12-20

[Jake87a] S.E.Jakes, N.J.Watts, P.Willett, D.Bawden, J.D.Fisher; "Pharmacophoric Pattern Matching In Files Of 3D Chemical Structures: Evaluation Of Search Performance"; Journal Of Molecular Graphics, vol 5 num 1 (1987) pp.41-48

[Jake87b] S.E.Jakes; Ph.D. Thesis, Department Of Information Studies, University Of Sheffield, (In Preparation)

[Jame75] T.L.James; "Nuclear Magnetic Resonance In Biochemistry"; Academic Press, New York, 1975

[Kao85] J.Kao, C.Eyermann, L.Watt, R.Maher, D.Leister; "A Versatile, Efficient, And Interactive Program To Build Molecular Structures For Theoretical Calculations And Chemical Information Systems"; Journal Of Chemical Information And Computer Sciences, vol 25 num 4 (1985) pp.400-410

[Katz72] L.Katz, C.Levinthal; "Interactive Computer Graphics And Representation Of Complex Biological Structures"; Annual Review Of Biophysics And Bioengineering, vol 1 (1972) pp.465-504

[Kier85] L.B.Kier; "A Shape Index For Molecular Graphs"; Quantitative Structure-Activity Relationships, vol 4 (1985) pp.109-116

[Kitc82] L.Kitchen, E.V.Krishnamurthy; "Fast, Parallel Relaxation Screening For Chemical Patent Data-Base Search"; Journal Of Chemical Information And Computer Sciences, vol 22 num 1 (1982) pp.44-48

[Klei86] T.E.Klein, C.Huang, T.E.Ferrin, R.Langridge, C.Hansch; "Computer-Assisted Drug Receptor Mapping Analysis"; Chapter 13 pp.147-158 in "Artificial Intelligence Applications In Chemistry"; Editors: T.H.Pierce, B.A.Hohne; American Chemical Society Symposium Series, vol 306, American Chemical Society, 1986

[Knut74] D.E.Knuth; "Structured Programming With Go To Statements"; Computing Surveys, vol 6 num 4 (1974) pp.261-301

[Knut81] D.E.Knuth; "The Art Of Computer Programming Volume 2 Seminumerical Algorithms"; Addison Wesley, Second Edition, London, 1981

[Koll84] P.Kollman; "Drug-Receptor Binding Forces"; Chapter 4 pp.63-82 in "X-ray Crystallography And Drug Action"; Editors: A.S.Horn, C.J.De Ranter; Oxford University Press, Oxford, 1984

[Kuck86] D.J.Kuck, E.S.Davidson, D.H.Lawrie, A.H.Sameh; "Parallel Supercomputing Today And The Cedar Approach"; Science, vol 231 num 4741 (1986) pp.967-974

[Kuhl84] F.S.Kuhl, G.M.Crippen, D.K.Friesen; "A Combinatorial Algorithm For Calculating Ligand Binding"; Journal Of Computational Chemistry, vol 5 num 1 (1984) pp.24-34

[Kung82] H.T.Kung; "Why Systolic Architectures?"; Computer, vol 15 num 1 (1982) pp.37-46

[Kunt82] I.D.Kuntz, J.M.Blaney, S.J.Oatley, R.Langridge, T.E.Ferrin; "A Geometrical Approach To Macromolecule-Ligand Interactions"; Journal Of Molecular Biology, vol 161 num 2 (1982) pp.269-288

[Laba86] J.Labanowski, I.Motoc, C.B.Naylor, D.Mayer, R.A.Dammkoehler; "Three-Dimensional Quantitative Structure-Activity Relationships 2 Conformational Mimicry And Topographical Similarity Of Flexible Molecules"; Quantitative Structure-Activity Relationships, vol 5 (1986) pp.138-152

[Lang81] R.Langridge, T.E.Ferrin, I.D.Kuntz, M.L.Connolly; "Real-Time Color Graphics In Studies Of Molecular Interactions"; Science, volume 211 number 4483 (1981) pp.661-666

[Lawl85] E.L.Lawler, J.K.Lenstra, A.H.G.Rinnoy Kan, D.B.Shmoys; "The Traveling Salesman Problem"; Wiley, Chichester, 1985

[Lesk79] A.M.Lesk; "Detection Of Three-Dimensional Patterns Of Atoms In Chemical Structures"; Communications Of The ACM, vol 22 num 4 (1979) pp.219-224

[Levi66] C.Levinthal; "Molecular Model-Building By Computer"; Scientific American, vol 214 num 6 (1966) pp.44-52

[Levi72] G.Levi; "A Note On The Derivation Of Maximal Common Subgraphs Of Two Directed Or Undirected Graphs"; Calcolo, vol 9 (1972) pp.341-352

[Lori75] H.Lorin; "Sorting And Sort Systems"; Addison-Wesley, London, 1975

[Louk81] E.Loukakis, C.Tsouros; "A Depth First Search Algorithm To Generate The Family Of Maximal Independent Sets Of A Graph Lexicographically"; Computing, vol 27 num 4 (1981) pp.349-366

[Louk83] E.Loukakis; "A New Backtracking Algorithm For Generating The Family Of Maximal Independent Sets Of A Graph"; Computers And Mathematics With Applications, vol 9 num 4 (1983) pp.583-589

[Luks80] E.M.Luks; "Isomorphism Of Graphs Of Bounded Valence Can Be Tested In Polynomial Time"; Proceedings Of The 21st IEEE Foundations Of Computer Science Symposium, IEEE, New York, pp.42-49, 1980

[Lync75] M.F.Lynch; "Screening Large Chemical Files"; Chapter 12 in "Chemical Information Systems"; Editors: J.Ash, E.Hyde; Ellis Horwood, Chichester, 1975

[Lync81] M.F.Lynch, J.M.Barnard, S.M.Welford; "Computer Storage And Retrieval Of Generic Chemical Structures In Patents 1 Introduction And General Strategy"; Journal Of Chemical Information And Computer Sciences, vol 21 num 3 (1981) pp.148-150

[Lync87] M.F.Lynch, G.A.Manson, P.Willett, G.A.Wilson; "The Application Of Reconfigurable Microprocessors To Information Retrieval Problems"; British Library Research And Development Report, March 1987

[Mars79] G.R.Marshall, C.D.Barry, H.E.Bosshard, R.A.Dammkoehler, D.A.Dunn; "The Conformational Parameter In Drug Design: The Active Analog Approach"; Chapter 9 pp.205-226 in "Computer Assisted Drug Design"; Editors: E.C.Olson, R.E.Christoffersen; American Chemical Society, Washington D.C., 1979

[Mars84] G.R.Marshall; "Computer-Aided Drug Design"; Chapter 1 pp.3-20 in "Computer-Aided Molecular Design"; Conference Transcript, London, October 1984; Oyez Scientific And Technical Services Limited, London, 1985

[Mart81] Y.C.Martin; "A Practitioner's Perspective Of The Role Of Quantitative Structure-Activity Analysis In Medicinal Chemistry"; Journal Of Medicinal Chemistry, vol 24 num 3 (1981) pp.229-237

[Matt75] R.J.Matthews; "A Comment On Structure-Activity Correlations Obtained Using Pattern Recognition Methods"; Journal Of The American Chemical Society, vol 97 num 4 (1975) pp.935-936

[May84] D.May, R.Taylor; "Occam - An Overview"; Microprocessors And Microsystems, vol 8 num 2 (1984) pp.73-79

[McCa85] J.T.McCall, J.G.Tront, F.G.Gray, R.M.Haralick, W.M.McCormack; "Parallel Computer Architectures And Problem Solving Strategies For The Consistent Labelling Problem"; IEEE Transactions On Computers, vol 34 num 11 (1985) pp.973-980

[McCo82a] W.M.McCormack, F.G.Gray, R.M.Haralick; "A Simulation Model Of A Multi-Computer System Solving A Combinatorial Problem"; Proceedings Of The 1982 Winter Simulation Conference, vol 1 (1982) pp.261-266

[McCo82b] W.M.McCormack, F.G.Gray, J.G.Tront, R.M.Haralick, G.S.Fowler; "Multi-Computer Parallel Architectures For Solving Combinatorial Problems"; pp.431-451 in "Multicomputers And Image Processing : Algorithms And Programs"; Editors: K.Preston, L.Uhr; Academic Press, New York, 1982

[McGr79] J.J.McGregor; "Relational Consistency Algorithms And Their Application In Finding Subgraph And Graph Isomorphisms"; Information Sciences, vol 19 (1979) pp.229-250

[McGr81] J.J.McGregor, P.Willett; "Use Of A Maximal Common Subgraph Algorithm In The Automatic Identification Of Ostensible Bond Changes Occurring In Chemical Reactions"; Journal Of Chemical Information And Computer Sciences, vol 21 num 3 (1981) pp.137-140

[McGr82] J.J.McGregor; "Backtrack Search Algorithms And The Maximal Common Subgraph Problem"; Software Practice And Experience, vol 12 num 1 (1982) pp.23-34

[McLa82] A.D.McLachlan; "Rapid Comparison Of Protein Structures"; Acta Crystallographica, vol A 38 (1982) pp.871-873

[Metc85] M.Metcalf; "FORTRAN Optimization"; Academic Press, London, (Revised Edition) 1985

[Morg65] H.L.Morgan; "The Generation Of A Unique Machine Description For Chemical Structures: A Technique Developed At Chemical Abstracts Service"; Journal Of Chemical Documentation, vol 5 (1965) pp.107-113

[Moto81] I.Motoc; "An Improved Version Of The Steric Difference Method"; Drug Research, vol 31-2 num 1 (1981) pp.290-293

[Moto86] I.Motoc, R.A.Dammkoehler, D.Mayer, J.Labanowski; "Three-Dimensional Quantitative Structure-Activity Relationships 1 General Approach To The Pharmacophore Model Validation"; Quantitative Structure-Activity Relationships, vol 5 num 3 (1986) pp.99-105

[Murr84] P.Murray-Rust; "Databases Of Molecular Structure"; Chapter 11 pp.187-194 in Computer-Aided Molecular Design Conference transcript, London, October 1984; Oyez Scientific And Technical Services Limited, London, 1985

[Palm83] R.A.Palmer, J.H.Tickle, I.J.Tickle; "Acetylcholine Receptor Site: A Proposed Model"; Journal Of Molecular Graphics, vol 1 num 4 (1983) pp.94-106

[Papa82] C.H.Papadimitriou, K.Steiglitz; "Combinatorial Optimization: Algorithms And Complexity"; Prentice-Hall, New Jersey, 1982

[Polt82] D.J.Polton; "Installation And Operational Experiences With MACCS (Molecular Access System)"; Online Review, vol 6 num 3 (1982) pp.235-242

[Poun86] D.Pountain; "A Tutorial Introduction To Occam Programming"; INMOS Limited, July 1986

[Pric85] K.E.Price; "Relaxation Matching Techniques -A Comparison"; IEEE Transactions On Pattern Analysis And Machine Intelligence, vol 7 num 5 (1985) pp617-623

[Read77] R.C.Read, D.G.Corneil; "The Graph Isomorphism Disease"; Journal Of Graph Theory, vol 1 num 4 (1977) pp.339-363

[Redl74] G.Redl, R.D.Cramer, C.E.Berkoff; "Quantitative Drug Design"; Chemical Society Reviews, vol 3 (1974) pp.273-292

[Seag86] M.K.Seager; "Parallelizing Conjugate Gradient For The CRAY X-MP"; Parallel Computing, vol 3 num 1 (1986) pp.35-47

[Seit85] C.L.Seitz; "The Cosmic Cube"; Communications Of The ACM, vol 28 num 1 (1985) pp.22-33

[Shea82] D.C.S.Shearn; "PASSIM - A Pascal Simulation System"; Division Of Economic Studies, University Of Sheffield, 1982

[Shei81] B.A.Sheil; "The Psychological Study Of Programming"; Computing Surveys, vol 13 num 1 (1981) pp.101-120

[Sher86] R.P.Sheridan, R.Nilakantan, J.S.Dixon, R.Venkataraghavan; "The Ensemble Approach To Distance Geometry: Application To The Nicotinic Pharmacophore"; Journal Of Medicinal Chemistry, vol 29 num 6 (1986) pp.899-906

[Sing69] R.C.Singleton; "Algorithm 347 An Efficient Algorithm For Sorting With Minimal Storage [M1]"; Communications of the ACM, vol 12 num 3 (1969) pp.185-187

[Smit78] B.J.Smith; "A Pipelined Shared Resource MIMD Computer" IEEE Proceedings 1978 International Conference On Parallel Processing (1978) pp.6-8

[Stew87] M.Stewart, P.Willett; "Nearest Neighbour Searching In Binary Search Trees: Simulation Of A Multiprocessor System"; Journal Of Documentation, (In Press)

[Stob85] R.E.Stobaugh; "Chemical Substructure Searching"; Journal Of Chemical Information And Computer Sciences, vol 25 num 3 (1985) pp.271-275

[Stup76] A.J.Stuper, P.C.Jurs; "ADAPT:A Computer System For Automated Data Analysis Using Pattern Recognition"; Journal Of Chemical Information And Computer Sciences, vol 16 num 2 (1976) pp.99-105

[Stup79] A.J.Stuper, W.E.Brugger, P.C.Jurs; "Computer-Assisted Studies Of Chemical Structures And Biological Function"; Wiley, New York, 1979

[Sund74] K.Sundaram, S.Mahajan, R.K.Mishra; "A Quantitative Approach To The Comparison Of Biomolecular Topographies"; Physiological Chemistry And Physics, vol 6 (1974) pp.469-478

[Suss65] E.H.Sussenguth; "A Graph-Theoretic Algorithm For Matching Chemical Structures"; Journal Of Chemical Documentation, vol 15 num 5 (1965) pp.36-43

[Taba87] D.Tabak; "RISC Architecture"; Research Studies Press, Letchworth, 1987

[Toll84] J.P.Tollenare, H.Moereels, L.A.Raymaekers; "Molecular Modelling By Computer Techniques And Pharmacophore Identification"; Chapter 25 pp.461-483 in "X-ray Crystallography And Drug Action"; Editors: A.S.Horn, C.J.De Ranter; Oxford University Press, Oxford, 1984

[Topl83] J.G.Topliss (Editor); "Quantitative Structure-Activity Relationships In Drugs"; Academic Press, New York, 1983

[Town85] W.G.Town, M.F.Lynch, P.Willett, G.C.Ford, D.W.Rice; "Advanced Computational Support For Biotechnology Research"; Commission Of The European Communities X11/600/85-EN ITTTF/CUBE, May 1985

[Ullm76] J.R.Ullman; "An Algorithm For Subgraph Isomorphism"; Journal Of The ACM, vol 23 num 1 (1976) pp.31-42

[Vand74] G.G.Vander Stouw, P.M.Elliott, A.C.Isenberg; "Automated Conversion Of Chemical Substance Names To Atom-Bond Connection Tables"; Journal Of Chemical Documentation, vol 14 num 4 (1974) pp.185-193

[Vark79] T.H.Varkony, Y.Shiloach, D.H.Smith; "Computer-Assisted Examination Of Chemical Compounds For Structural Similarities"; Journal Of Chemical Information And Computer Sciences, vol 19 num 2 (1979) pp.104-111

[Vint85] J.G.Vinter; "Molecular Graphics For The Medicinal Chemist"; Chemistry In Britain, vol 21 num 1 (1985) pp.32-38

[Vlad87] G.E.Vladutz; "A Hyperstructure Of Superposed Compound And Reaction Diagrams For Joint Reaction/Compound Storage And Retrieval"; in "Proceedings Of Chemical Structures: The International Language Of Chemistry"; Chemical Structures Association (In Preparation)

[Voll83] J.Vollmer; "WLN.An Introduction"; Journal Of Chemical Education, vol 60 num 3 (1983) pp.192-196

[VonS84] A.von Scholley; "A Relaxation Algorithm For Generic Chemical Structure Screening"; Journal Of Chemical Information And Computer Sciences, vol 24 num 4 (1984) pp.235-241

[Wah85] B.W.Wah, G.J.Li, C.F.Yu; "Multiprocessing Of Combinatorial Search Problems"; Computer, vol 18 num 6 (1985) pp.93-108

[Walt84] D.E.Walters, A.J.Hopfinger; "Applications Of Molecular Shape Analysis To QSAR"; pp.279-286 in "QSAR In Design Of Bioactive Compounds"; Proceedings of the 1st Telesymposium on Medicinal Chemistry February 29, 1984; J.R.Prous International Publishers, Barcelona, 1984

[Watt84] N.Watts; "Creation Of A Base Of Pharmacophores For Use In 3D Substructure Searching Of Computer-Based Chemical Information Systems"; M.Sc. Thesis, Department Of Information Studies, Sheffield University, 1984

[Weng82] J.C.Wenger, D.H.Smith; "Deriving Three-Dimensional Representations Of Molecular Structure From Connection Tables Augmented With Configuration Designations Using Distance Geometry"; Journal Of Chemical Information And Computer Sciences, vol 22 num 1 (1982) pp.29-34

[Will77] R.J.P.Williams; "Flexible Drug Molecules And Dynamic Receptors"; Angewandie Chemie International Edition In English, vol 16 num 11 (1977) pp.766-777

[Will86] P.Willett, V.Winterman, D.Bawden; "Implementation Of Nearest Neighbour Searching In An Online Chemical Structure Search System"; Journal Of Chemical Information And Computer Sciences, vol 26 num 1 (1986) pp.36-41

[Will87a] P.Willett; "A Review Of Chemical Structure Retrieval Systems"; Journal Of Chemometrics, (In Press)

[Will87b] P.Willett; "Similarity And Clustering In Chemical Information"; Research Studies Press, Letchworth, 1987

[Wipk74] W.T.Wipke, T.M.Dyott; "Stereochemically Unique Naming Algorithm"; Journal Of The American Chemical Society, vol 96 num 15 (1974) pp.4834-4842

[Wipk84] W.T.Wipke, D.Rogers; "Rapid Subgraph Search Using Parallelism"; Journal Of Chemical Information And Computer Sciences, vol 24 num 4 (1984) pp.255-262

[Wold83] S.Wold, W.J.Dunn; "Multivariate Quantitative Structure-Activity Relationships (QSAR): Conditions For Their Applicability"; Journal Of Chemical Information And Computer Sciences, vol 23 num 1 (1983) pp.6-13

[Wong83] A.K.C.Wong, F.A.Akinniyi; "An Algorithm For The Largest Common Subgraph Isomorphism Using The Implicit Net"; IEEE Proceedings International Conference On Systems, Man And Cybernetics, 29 December 1983 to 7 January 1984, Bombay and New Delhi, India, pp.197-201

[Wool84] K.R.H.Woolridge; "The Virtues Of Present Strategies For Drug Discovery"; Chapter 11 pp.209-216 in "Drug Design: Fact Or Fantasy ?"; Editors: K.R.H.Woolridge, G.Jolles; Academic Press, London, 1984

[Wyke87] A.Wyke; "Pharmaceuticals"; The Economist, 7 February 1987 pp.3-18

[Zakh84] V.Zakharov; "Parallelism And Array Processing"; IEEE Transactions On Computers, vol 33 num 1 (1984) pp.45-78

Alterations

Several paragraphs and sentences in the main body of the thesis have been marked as requiring clarification. This section gives the amended versions with which they should be replaced.

Page 136 The following sentences should be added to the end of the first paragraph of Section 5.6.2

" For example, although it was not possible in view of the cpu times involved to obtain the durations of each process by running a serial version as in [Stew87], some limited form of statistical test such as chi squared could have been undertaken to analyse how closely the actual distributions correlated with the estimated ones. However, this would have led to considerably more work being involved."

Page 151 The following paragraph should be added to the bottom of the page

"As the times recorded were the lowest times, the speed ups obtained when using several processors as opposed to a single one, are higher than would normally be obtained in practice where the partition factors have to be determined beforehand. Some indication of the spread of times is given in Section 6.4 but because of the sheer volume of data, the other less optimal figures are not included here. (However, they can be obtained by directly contacting the author)."

Page 191 The sentence starting on line 2 should read

"It should be mentioned though that CAVROG was chosen for this analysis because it was the compound from table 8.6 which led to the largest common substructures being found in the database."

Page 193 Lines 14 to 19 should be replaced by

"This was because the other three algorithms ran into storage problems. However, its run times were very large and so work is currently being undertaken in the department [Davi87] to try to lessen the overall time taken by a search by using Lesk's algorithm as a screening stage. The atoms which pass this stage are then passed on to Ullman's (quicker) algorithm."