THEORETICAL & PRACTICAL APPROACHES TO THE MODELLING OF CRYSTAL & MOLECULAR STRUCTURES

A Thesis presented to the University of Surrey for the degree of Doctor of Philosophy in the School of Physical Sciences

By
Colin Richard BALDWIN

Structural and Computational Chemistry Group
Department of Chemistry
University of Surrey

October 1997
ABSTRACT

A genetic algorithm has been proposed as a computational method for producing molecular mechanics force field parameters, using input data from the Cambridge Structural Database. The method has been applied initially to simple test data and to a coordination compound under various conditions and the results have been analysed in an attempt to determine the most suitable operating parameters. Finally, several possible approaches, both software and hardware, aimed towards improving the algorithm's performance, are discussed.

Two approaches for extending the performance of a PC have been considered, namely upgrading the computational power and the graphics capabilities using state-of-the-art hardware solutions. Both of these features can be considered essential for crystal modelling. Conclusions have then been drawn regarding the applicability of these approaches to a modern, top-of-the-range PC.

Finally, a variety of software modules are proposed, aimed at the 'engineering' of known crystal structures. Many of these techniques are graphical in nature, enabling the visualisation and manipulation of the inherent symmetry these systems display.
ACKNOWLEDGEMENTS

I would like to take this opportunity to thank the people who have enabled this research to be carried out.

Firstly, I wish to express my sincere gratitude to my supervisor, Dr. David Povey, for his constant help, guidance and encouragement throughout the period of this research. Thanks are also due to Dr. Brendan Howlin, for his assistance with the provision of the molecular fitting routine used in the genetic algorithm. I would also like to thank Mr. Graham Wright for his maintenance of the computing equipment used in this project.

I would like to acknowledge the two organisations who provided equipment during the project - Dr. S Ramdas at BP Sunbury for providing the Silicon Graphics Crimson workstation and MicroWay Ltd., Kingston-Upon-Thames, Surrey for the Number Smasher-860 board.

Finally, I would like to thank my family for the support and encouragement they have shown me throughout my studies.
CONTENTS

Abstract i
Acknowledgements ii
Contents iii

INTRODUCTION
1.1 Introduction 2
1.2 Molecular Modelling 3
   1.2.1 Introduction 3
   1.2.2 Molecular Graphics 4
   1.2.3 Why Attempt To Predict Structure? 5
   1.2.4 Tools For Predicting Chemical Structures 6
   1.2.5 Prediction of Properties 13
   1.2.6 Limitations of Molecular Modelling Techniques 17
   1.2.7 Summary 23
1.3 Crystal Modelling 23
   1.3.1 An Approach to Modelling Inorganic Molecular Crystals 25
1.4 Layout of Thesis 27

A GENETIC ALGORITHM FOR GENERATING MOLECULAR MECHANICS FORCE FIELD PARAMETERS
2.1 Introduction 31
2.2 Genetic Algorithms 31
   2.2.1 Principles of Genetic Algorithms 32
   2.2.2 Genetic Algorithms in Chemistry 49
2.3 Molecular Mechanics 49
   2.3.1 The Standard Force Field 52
   2.3.2 Additional Terms in the Energy Expression 59
   2.3.3 Specialised Force Fields 63
   2.3.4 Energy Minimisation Techniques 67
   2.3.5 Parameterisation Methods 70
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4</td>
<td>Application of Molecular Mechanics to Coordination Compounds</td>
<td>71</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Introduction</td>
<td>71</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Problems With Conventional Force Fields</td>
<td>72</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Methods Developed</td>
<td>75</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Examples</td>
<td>79</td>
</tr>
<tr>
<td>2.5</td>
<td>Automatic Parameterisation of Molecular Mechanics Force Fields</td>
<td>83</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Introduction</td>
<td>83</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Examples</td>
<td>83</td>
</tr>
<tr>
<td>2.6</td>
<td>A Genetic Algorithm for Determining Force Field Parameters</td>
<td>87</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Program Design</td>
<td>87</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Initial Investigations: Straight-Chain Alkanes</td>
<td>97</td>
</tr>
<tr>
<td>2.6.3</td>
<td>Optimising the GA</td>
<td>105</td>
</tr>
<tr>
<td>2.6.4</td>
<td>Investigation of Optimum Mutation Rate</td>
<td>111</td>
</tr>
<tr>
<td>2.6.5</td>
<td>Determination of Optimum Convergence Threshold</td>
<td>138</td>
</tr>
<tr>
<td>2.6.6</td>
<td>The Effect of Including Energy in the Score</td>
<td>141</td>
</tr>
<tr>
<td>2.6.7</td>
<td>Two Point Crossover</td>
<td>148</td>
</tr>
<tr>
<td>2.6.8</td>
<td>Application of the GA to a Coordination Complex</td>
<td>154</td>
</tr>
<tr>
<td>2.6.9</td>
<td>Future Developments</td>
<td>161</td>
</tr>
<tr>
<td>2.6.10</td>
<td>Conclusions</td>
<td>164</td>
</tr>
</tbody>
</table>

HARDWARE CONSIDERATIONS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>167</td>
</tr>
<tr>
<td>3.2</td>
<td>Enhancing A PC’s Processing Power</td>
<td>167</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Available Processors</td>
<td>168</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Enhancing Computational Power</td>
<td>172</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Enhancing Graphics Capability</td>
<td>181</td>
</tr>
<tr>
<td>3.3</td>
<td>PC Hardware Incompatibilities</td>
<td>184</td>
</tr>
<tr>
<td>3.4</td>
<td>Recent Developments</td>
<td>186</td>
</tr>
<tr>
<td>3.5</td>
<td>Conclusions</td>
<td>187</td>
</tr>
</tbody>
</table>
DESIRABLE FEATURES IN A CRYSTAL MODELLING PACKAGE

4.1 Introduction

4.1.1 Ligand Browser 190
4.1.2 CSD On-line Link 197
4.1.3 Symmetry Operators Overlay 203
4.1.4 Manipulation of Symmetry Related Fragments 206
4.1.5 Interactive Modification of Unit Cell Parameters 206
4.1.6 Symmetry Operator Replacement 206
4.1.7 Molecular Volume Calculations 207
4.1.8 Lattice Slicing Algorithm 210
4.1.9 Single Crystal X-ray Diffraction Pattern 210
4.1.10 Ligand MO/Electrostatic Calculations 210
4.1.11 Structure & Property Prediction Algorithms 210
4.1.12 Partial Display of Structure 211
4.1.13 Data File Input/Output 211

4.2 Conclusions 210

CONCLUSIONS

5.1 Conclusions 214

REFERENCES 219

APPENDIX: Source Code 228
INTRODUCTION

1.1 Introduction

1.2 Molecular Modelling
   1.2.1 Introduction
   1.2.2 Molecular Graphics
   1.2.3 Why Attempt To Predict Structure?
   1.2.4 Tools For Predicting Chemical Structures
      1.2.4.1 Classical Mechanics
      1.2.4.2 Quantum Mechanics
      1.2.4.3 Solid State Modelling
   1.2.5 Prediction of Properties
      1.2.5.1 Electronic Isopotential Maps
      1.2.5.2 Quantitative Structure Activity Relationships (QSARs)
   1.2.6 Limitations of Molecular Modelling Techniques
      1.2.6.1 Organic Chemistry
      1.2.6.2 Inorganic Chemistry
   1.2.7 Summary

1.3 Crystal Modelling
   1.3.1 An Approach to Modelling Inorganic Molecular Crystals

1.4 Layout of Thesis
1.1 Introduction

The aims of this project are to increase the range of computational tools available for studying metal coordination complexes and to investigate whether such modelling programs could be implemented on affordable hardware. More explicitly, the objectives can be summarised as follows:

- To develop a technique which automates the derivation of force field parameters for use in molecular mechanics calculations.

- To evaluate whether an IBM compatible personal computer may be enhanced in terms of hardware capabilities to such an extent that it rivals the performance of graphics workstations.

- To investigate the software tools which may be developed or used for the molecular modelling of metal coordination complexes.

This chapter introduces the reasons for attempting to predict chemical structure and discusses the techniques which have been developed to perform these calculations. The application of these techniques to different classes of compound is then reviewed. This is followed by a discussion of some of the techniques available for modelling a molecule’s properties.
1.2 Molecular Modelling

1.2.1 Introduction
The term ‘molecular modelling’ can be applied equally to more elementary studies using mechanical models or even pen and paper, as well as to the more modern computational techniques. Indeed, the American chemist G. N. Lewis can be credited with molecular modelling when he proposed the ‘dot’ notation (which greatly simplified the writing of atomic structures) and its subsequent use in the prediction of molecular topology (DeKock, 1980). Other molecular models which are still in use today, incorporated within computational packages, include the commonly used display techniques - ball and stick, cylinder, van der Waals’ surface or spacefill and protein backbone or ribbon.

However, in modern times, molecular modelling has come to be regarded as a ‘catch all’ phrase, which is used to describe a wide variety of computational techniques within chemistry. Texts by Hinchcliffe (1996) and Leach (1996) both provide an overview of many of these techniques. They fall into two main areas - the prediction of a compound’s structure (molecular or otherwise) and the prediction of some form of its behaviour.

Whilst computational chemistry is still a relatively new concept, its origins are based in the principles of theoretical and quantum chemistry which were proposed long before computers were even dreamt of. Computers only provide the number-crunching capability to solve these problems on a realistic time scale. The rapid uptake of computational chemistry within the scientific community is entirely due to the dramatic advances in computational power which have arisen since the 1960s. One of the most important advances which has led to this increased popularity is the development of high quality computer graphics.
1.2.2 Molecular Graphics

Molecular graphics actually plays quite a minor role in the overall picture of molecular modelling, acting mainly as an intermediary between the user and the computational algorithm which has generated the results being displayed. In fact, it could be argued that for a ‘snapshot’ of a molecule within time, computer graphics has very few advantages over a mechanical model. Both systems allow exploration by the rotation of the molecule, and the mechanical model is inherently three-dimensional, whereas mathematical techniques such as depth-cueing have to be applied to the computational model to give the impression of a three-dimensional molecule.

The advantages of molecular graphics in this scenario include the speed with which the model can be generated from a set of coordinates and the variety of different display styles which can be applied to the model to reveal more information. Mechanical models of large molecules can also be time consuming to construct. A computer can be used to generate such a model very rapidly and, because the resulting model can appear extremely cluttered, most molecular graphics systems include the facility to switch on or off the display of certain elements of the structure. For example, in protein modelling one may wish to display the backbone or structural units such as helices, α-turns or β-sheets: all of this functionality may be available almost instantaneously.

However, molecules are not static entities, and chemists are obviously interested in how they move, as this information can be used to help predict, for example, low energy conformations, changes in electronic charge distribution over the molecule’s surface and how a particular compound will behave in a particular environment. Using a mechanical model to display this type of information would
be extremely time consuming at best and, more likely, impossible. Computer
graphics are ideally suited to displaying this type of information.

1.2.3 Why Attempt To Predict Structure?
At the most fundamental level, chemistry depends upon the behaviour of the
electrons around the nuclei of a molecule, and what happens to these electrons
during the process of a chemical reaction. By modelling the electronic structure of
a molecule, chemists hope to gain an insight into how a compound is likely to react.

Chemists are also preoccupied with the structure of molecules because the
structure or conformation of a molecule is inextricably linked to the compound’s
properties. It is these properties which will decide whether a particular compound
is suitable for its intended use. The properties of interest may be physical
(conductivity, processability etc.) or chemical (reactivity etc.). Methodologies
have been developed to attempt to predict rationally some of these properties, for example:

- Conductivity - band gap
- X-ray diffraction patterns
- The behaviour of molecules adsorbing onto a surface or through a zeolite

However, the one common factor linking all of these property-predicting
techniques is that they all rely upon the availability of an accurate initial structure.
Several techniques have been implemented on computers which make attempts
(with varying degrees of accuracy) at predicting the structure of a variety of
classes of compound.
1.2.4 Tools For Predicting Chemical Structures

Of course, the only methods available for unambiguously determining the three-dimensional structure of a molecule are the use of diffraction methods and possibly NMR. Whilst the chemist can make some predictions about the structure of a new molecule by studying the structures of similar molecules, there are no hard and fast rules of how this should be done. The only solution is the synthesis of the compound.

For true prediction of chemical structure (and specifically molecular structure), the most commonly used computational tools available fall into one of two categories - those based upon classical mechanics and those on quantum mechanics. A third category of calculations is required when molecules in the solid state (specifically non-molecular compounds) are considered.

1.2.4.1 Classical Mechanics

The two main tools which use classical mechanics methods are molecular mechanics and molecular dynamics.

Molecular mechanics is concerned with refining a molecule’s structure to a low energy conformation. Molecular dynamics uses similar techniques, but allows the molecule’s movements to be followed through time: due to the complexity and intensive nature of the calculation, the time scale which is commonly chosen is of the order of tens of picoseconds \((10^{-12} \text{ seconds})\).

In both cases, the molecule is described by an energy expression which usually has terms related to bond strain, angle strain, torsional strain, charge interactions and van der Waals’ effects. Each of these terms is described using a classical mechanics principle. For example, bond and angle strain are commonly
approximated by Hooke’s Law for simple harmonic motion. The bond energy term is therefore:

\[ E_{\text{bond}} = \frac{k_b}{2} (b - b_0)^2 \]

where \( E_{\text{bond}} \) is the bond energy, \( k_b \) is the force constant for the particular bond in question, \( b \) is the bond length in the molecule and \( b_0 \) is the equilibrium or ideal bond length. This collection of parameters for all structural elements in the molecule is known as the force field.

The major drawback of classical mechanics techniques is the requirement of
i) a suitable energy function to describe the molecule, and
ii) a force field which contains parameters which will accurately model the molecule under investigation. The development of such parameter sets can be extremely time consuming.

If one of these elements is missing, the results obtained can be wildly inaccurate. However, for systems which are well defined, classical mechanics techniques can produce an extremely accurate model of the molecule. The main advantage of classical techniques is that they are considerably less intensive computationally than methods which use quantum mechanics, so results can be obtained much more rapidly.

A more detailed discussion of the principles involved in molecular mechanics can be found in Chapter Two.

1.2.4.2 Quantum Mechanics

The principles of quantum mechanics were first proposed many years before the development of the first computers. Theoretically, all the properties of matter may
be predicted by the solution of the Schrödinger Equation. The early applications of
the technique were restricted to small (atomic or diatomic) systems with few
electrons, as these could be solved approximately using pen and paper. With the
advent of the computer, chemists were quick to realise that the power they now
possessed would enable them to tackle larger systems, and a series of algorithms
were developed which implemented the principles of quantum mechanics.

Quantum mechanics describes the structure of a molecule using a series of
delocalised molecular orbitals, based upon simpler orbitals derived from the
hydrogen atom. Since the technique explicitly deals with the electrons in the
system, quantum mechanics can be used to derive properties which depend upon
the electronic distribution within the molecule. This information can also be used
to study the breaking and formation of bonds occurring during a chemical reaction.

These calculations are extremely complex, and increase in complexity with each
extra electron within the system. Whilst an ab-initio approach to the problem will
often provide a very accurate solution, for many systems it is uneconomical to
attempt such a calculation due to the length of computational time required to
perform the calculation. Even if enough computational power is available, the
operation of the programs can be very complex, making their implementation far
from straightforward.

In an attempt to solve this problem, chemists have developed a series of
semi-empirical quantum mechanics methods. These algorithms provide only an
approximate solution to the Schrödinger Equation - however their demands
computationally are much lower than the ab-initio approach, so larger molecules
can be considered. As with classical mechanics methods, parameter sets derived
from experimental data are used in these calculations and considerable care must
be taken to ensure that the parameters used are appropriate for the molecule under consideration.

There are several different semi-empirical computational techniques available to the chemist - most have undergone a great deal of revision and refinement since they were first introduced. These NDO (Neglect of Differential Overlap) methods differ in the way that they treat the overlap between pairs of different orbitals. Examples include CNDO (Complete Neglect of Differential Overlap) and INDO (Intermediate NDO) proposed by Pople and co-workers (1965 and 1967 respectively), and MNDO (Modified NDO) from Dewar (1970).

1.2.4.3 Solid State Modelling

In more recent times, attention has turned to the development of new materials which display desirable properties in the solid state. Whilst many of these materials are more appropriately described as non-molecular inorganic compounds, the prediction of their structure and properties is still covered by the term molecular modelling.

In terms of structure prediction, algorithms have been developed which may be used to predict the arrangement of the atoms (or, more commonly, ions) within the lattice, defect properties and the structure of the material's surface.

One of the techniques commonly used to study this type of material is Perfect Lattice Modelling. This principle is embodied in computer codes such as CASCADE (Cray Automated System for the Calculation of Defect Energies) (Leslie, 1982).
To accurately model solid state systems one needs a description of the energy of the system as a function of its atomic coordinates. Most inorganic solids are 'polar solids' - there are appreciable charges present on the atoms (even if they are not totally integral). The Interatomic Potential Model has three main features:

- **Fully Ionic (Born) Model** - Long range electrostatic term. The energy contribution is given by:
  \[
  \text{Energy} = \sum \frac{Z_i Z_j e^2}{r_{ij}}
  \]
  where \(Z_i\) and \(Z_j\) represent the charges on the species, \(e\) is the electron charge and \(r_{ij}\) is the distance between \(i\) and \(j\).

- **Short Range Interactions** - A two body potential, often represented by a simple analytical function such as the Buckingham Potential, which has the form:
  \[
  \text{Energy} = A e^{-\left(\frac{r_{ij}}{\rho}\right)} - \frac{c}{r_{ij}^6}
  \]
  where \(A\), \(\rho\) and \(c\) are parameters of the potential and \(r_{ij}\) is the distance between \(i\) and \(j\). This function is shown graphically in Figure 1.1.
This approach is suitable for many ionic solids - however for those compounds which display semi-ionic/semi-covalent character, such as Quartz ($\alpha$-SiO$_2$) and other silicates such as Aluminosilicates (Zeolites), a two body potential is not suitable. In these cases, the model must be extended to include a three body ‘angle-bending’ term.

$$\text{Energy} = \frac{k_\theta}{2} (\theta - \theta_0)^2$$

where $k_\theta$ is the angle bending constant and $\theta_0$ is the equilibrium bond angle between the three bodies.

- **Treatment of Ionic Polarisation** - Ionic polarisation arises from the distortion of the electron charge clouds around ions. In an isolated ion, the electron charge cloud is symmetrically distributed around the nucleus. In a solid lattice, this is clearly not the case. Polarisation and short range forces are therefore coupled.
These terms are dealt with using the Shell Model. This concept treats the atom or ion as two distinct units:

i) The core, consisting of the nucleus and the core electrons. This accounts for all of the mass of the atom or ion, and is immobile.

ii) The shell consists only of the valence shell electrons, and has no mass.

The core and shell are coupled by an harmonic spring of force constant $k_s$ (Figure 1.2).

![Diagram of the Shell Model for Ionic Polarisability](image)

**Figure 1.2 - Diagram of the Shell Model for Ionic Polarisability**

The polarisability, $\alpha$, is given by:

$$\alpha = \frac{Y^2}{k_s}$$

where $Y$ is the charge on the shell.

This is usually derived empirically from crystallographic data - a high value of $k_s$ implies a rigid ion, (i.e. one that is not easily polarised), and therefore a rigid structure.
1.2.5 Prediction of Properties

The whole thrust of molecular modelling is to predict the structure of a molecule or material which displays a desirable property. Chemists have developed a variety of computational techniques which allow them to attempt to predict a number of properties and some are outlined below.

1.2.5.1 Electronic Isopotential Maps

For pharmaceutical companies, one of the most important properties they wish to predict is the ability of a molecule to dock into a receptor site, in order for it to display some sort of biological activity. In this case, the two main requirements are:

- The physical ability of the molecule to fit into the receptor site.
- An appropriate match between a variety of characteristics over the surfaces of the molecule and the active site which will allow the molecule to approach and bind. Typical characteristics include charge distribution, hydrophobicity/philicity, lipophobicity/philicity or an ability to form hydrogen bonds.

The first of these requirements is obviously a structural problem which may be satisfied simply by producing an accurate determination of the reactive site - usually from analytical techniques such as diffraction data or NMR, or from theoretical methods such as homology modelling, and the prediction of a realistic model of the target molecule. The volumes of the modelled molecule and the void within the receptor molecule may then be calculated and compared.

The second requirement requires the production of surface maps such as, in the case of charge distribution, electronic isopotential maps of both the receptor site and the binding molecule. This is done by rolling a probe entity such as a proton
around the molecule and calculating the interaction energy between the probe and the molecule on a specific grid of points. The surface is usually calculated within a cut-off sphere of, for example, around 8\AA. The grid is then analysed to show contours of equal potential - usually displayed in colour. In order to perform this type of calculation, the partial charges of the atoms in the molecule must be determined. This information may be obtained via a variety of different techniques with varying degrees of accuracy. A variety of other surface maps may also be generated by the use of other probe molecules - for example, by using a solvent molecule such as water, one obtains the solvent accessible area.

Obviously the docking procedure is also an important feature in this type of work. Docking may be carried out interactively or by using an automatic docking algorithm, such as the DOCK program proposed by Kuntz et al (1982). One important consideration must be the effect of the docking procedure on the conformations of both entities. Computational techniques applied to conformational analysis include genetic algorithms, molecular dynamics, simulated annealing and systematic searching. Many of these methods have been used in docking algorithms at some time - at least for the docking molecule. In most cases, the structure of the receptor site is usually considered to be rigid.

Obviously, a more realistic approach to the docking problem would be to perform a molecular dynamics simulation of the molecule as it approaches and binds to the substrate - however the calculations are currently too intensive computationally, and molecular dynamics is only used as a final refinement for a docked molecule.
1.2.5.2 Quantitative Structure Activity Relationships (QSARs)

Often the relationship between a drug's structure and its biological activity in vivo is complex, and quite different from its behaviour in vitro. By studying structure-activity relationships, one can determine which of the structural features which a molecule possesses gives rise to these properties of interest, enabling the chemist to make modifications to a parent compound which may enhance its properties.

A Quantitative Structure Activity Relationship (QSAR) is a computational approach to structure activity relationships. It uses a mathematical model to relate the molecule's activity to numerical values associated with its structure. QSARs have proved to be effective in many cases which have been reported in the literature.

QSARs have their origins in the physical organic chemist’s Hammett Equation:

$$\log \left( \frac{k}{k_0} \right) = \rho \sigma$$

where \( k \) is the rate constant or equilibrium constant, \( k_0 \) is a reference compound’s rate or equilibrium constant (the reference compound is usually a hydrogen substituted compound), \( \rho \) is a reaction constant which is fixed for a particular reaction under specific conditions and \( \sigma \) is a substituent parameter. Hammett’s work was concerned with compounds such as substituted benzoic acids - in this case the substituent parameter depends upon the nature of the substituent group and whether it is meta or para to the carboxyl group.

The equation is rearranged to allow a graph of \( \log k \) to be plotted against \( \sigma \). This graph should result in a straight line of gradient \( \rho \), which allows the reaction constant for other compounds to be read off from the graph.
QSARs are a more complicated form of this principle - taking into account a large number of structural features. The basic equation which defines all QSARs is defined as:

\[ v = f(\rho) \]

where \( v \) is the property the molecule should possess and \( f(\rho) \) represents functions of numerical values which relating to properties of the molecule's structure.

In order to create a QSAR, the first step must be the collection of the required data. This is collated from reference sources or from the synthesis and determination of the biological activity of a number of compounds. The molecules which are used are usually related to each other structurally, although this need not necessarily be the case. It is important that the compounds used should display a wide variance for each of the properties under consideration, and various measures can be taken to test that this is the case. Typical properties which may be used in a QSAR include representations of electronic structure, molecular shape and commonly, partition coefficients.

Once the data has been obtained, the relationship between the properties and structure must be determined. A technique known as Multiple Linear Regression is usually used to calculate the values for the QSAR coefficients which best model the relationship. Once this relationship has been created, the structures of new compounds can be input into the QSAR, and a numerical value for their expected activity can be obtained.

Other techniques for determining the QSAR relationship, including the use of Neural Networks (Andrea, 1991) and Genetic Algorithms (Rogers, 1994) have also been proposed.
1.2.6 Limitations of Molecular Modelling Techniques

Whilst there have been major advances in the many fields of molecular modelling, the techniques which have been developed are not a universal panacea to the problems of structure or property prediction. The techniques have particular strengths and weaknesses when applied to the two main synthetic branches of chemistry, namely organic and inorganic chemistry.

1.2.6.1 Organic Chemistry

Organic chemistry is the area where most development work in molecular modelling has been focused. The pharmaceutical companies realised at an early stage that molecular modelling could be used to guide their research into new biologically active molecules. Their belief that computational chemistry could cut the cost and time needed to discover, develop and launch new drugs has led directly to the wide variety of chemical modelling tools available today. Currently most, if not all, major pharmaceutical, petrochemical and agrochemical companies employ a team of computational chemists.

- **Small Molecules** - For the prediction of structure in most areas of organic chemistry, the technique which is initially chosen is classical mechanics - both molecular mechanics and molecular dynamics. Whilst the techniques are not as accurate as the various quantum mechanics methods available, they are extremely fast. It has been discovered that the accuracy of the predicted structure may be improved by employing a set of parameters which has been developed specifically for the particular types of structures under investigation. A typical example is the AMBER force field, (Weiner, 1984, 1986), developed for modelling proteins and nucleic acids. Classical Mechanics calculations can provide information relating to molecular geometry, enthalpy, vibrational properties and entropy - but only for molecules for which the program has been
adequately parameterised. It should also be noted that static structural calculations are usually carried out on an isolated gaseous molecule at 0K, and the results obtained reflect these conditions.

Quantum mechanics calculations are ideally suited to small organic molecules. Because the structures under investigation have few atoms, and these atoms are usually from the first row of the periodic table, the number of electrons under consideration is relatively small - allowing the generation of accurate results in a short time scale.

- **Polymers** - For modelling the monomer units which make up a polymer, quantum mechanics calculations are again ideal. However, when one considers the polymer chain, the quantum calculations, even with the highest level of approximations, are still too intensive computationally to be performed on a routine basis. Much time has been spent on developing methodologies in this area and commercial polymer modelling software packages have been available for a number of years.

One important tool for polymer modelling is the use of classical mechanics - this is the only technique which is realistically able to perform calculations on the polymerised system. By performing molecular dynamics simulations on a periodic system, bulk structural and physical properties of the polymer may be predicted.

- **Proteins** - The interest in protein structure again arises from the pharmaceutical companies - most drugs act by binding to a protein molecule, so a knowledge of its structure is of paramount importance. X-ray crystallography and NMR spectroscopy are the two methods used to gain information about protein
structures, and the Brookhaven Protein Database contains information on about 6,500 structures the vast majority of which are proteins. Unfortunately, the determination of a protein’s structure is extremely time consuming - lagging far behind the rate with which new proteins are sequenced. This has led to the development of a range of protein modelling methodologies.

One approach to the problem of predicting a protein’s structure is to perform conformational analysis on the sequence of amino acids from first principles (i.e. with no prior knowledge of the protein’s structure). A variety of different methods have been employed to perform the conformational analysis. However, whilst the technique has been employed successfully to predict the conformations of small peptide molecules, it has so far only been able to predict the general folding present in proteins.

Protein molecules contain areas of secondary structure, such as α-helices and β-strands. These structures are present in most proteins, and are commonly formed by similar sequences of amino acids, even if the proteins themselves have completely different functionality. A ‘Rule-Based Approach’ can be used to first predict the structures of the secondary features, prior to predicting how these features pack together to form the whole protein. Statistical information is commonly used to predict whether a particular section of the protein sequence is likely to form an element of secondary structure, and if so, what that structure is likely to be (Chou, 1978). This is done by identifying the amino acids which commonly initiate secondary features and then analysing four or five subsequent amino acids to see if they will continue or terminate the structure. Once these elements have been identified, a series of rules derived by Cohen (1982) can be applied to predict how these structural elements are assembled.
Finally, having generated at least one proposed solution, an energy minimisation calculation can be carried out to refine the structure(s).

A variety of other methods have also been proposed for the prediction of a protein's secondary structure - however the success rate of even the best method is only 65-70%. If all the secondary structures were present in equal numbers, a success rate of 33% should be achieved purely by chance. One reason for this poor performance is that these methods only consider effects due to amino acids which are close together in the chain. Often, secondary structure is caused by interactions which are due to residues that are some distance apart in the sequence, but which are spatially close together. Obviously, the accuracy of the overall protein structure generated is heavily dependent upon the results of the secondary structure prediction. This type of approach has been used to predict a variety of protein structures - with varying degrees of success.

The most popular method for predicting protein structures involves homology (or comparative) modelling. Throughout proteins, similar three-dimensional structures appear frequently. Whilst this may be expected for proteins which have similar function, it is also possible for proteins with quite different biological functions to display similar structural features and conformations. Homology modelling uses a known protein structure to act as a template for the unknown one. The four steps involved in this technique are:

- Select the protein to be used as a template.
- Match up the amino acids between the two sequences.
- Generate/delete the polypeptide loops which join the aligned sequences.
- Minimise the energy of the completed structure, ensuring that the structure isn’t distorted too far from the initial proposal.
The accuracy of this approach obviously depends upon the closeness of the match between the target and the template molecule. The larger the inserted loops, the further the predicted structure can be from reality.

Despite the time and resources spent by a variety of researchers on the protein modelling problem, there is not yet available a predictive method which can be applied with complete confidence to a modelling problem. However, progress is being made, and the best results generated can be extremely close to the experimentally determined structure.

1.2.6.2 Inorganic Chemistry
In the same way that organic chemistry can be split up into a variety of classes of compound, so can inorganic chemistry. In this case, however, the division is based more upon the structural elements involved in the compound concerned, rather than size.

- **Non-Molecular Compounds** - These compounds are now commonly subjected, with some success, to solid-state modelling methodologies similar to those already outlined. The growth in these techniques can be attributed both to an increased interest in this type of compound in terms of, for example, heterogeneous catalysis, magnetic properties or conductivity, and also to the advances in computer power and a greater knowledge of the interatomic potentials involved. Typical classes of compounds these techniques are applied to include zeolites and high temperature superconductors.

- **Metal Complexes** - Until recent years, this area of inorganic chemistry has been relatively neglected in terms of molecular modelling. There are a variety of reasons why this is the case, the most important one being that, historically,
there has been less commercial interest in this class of molecule when compared to the organic pharmaceuticals market. More recently, this has begun to change as interest in materials chemistry has grown.

The use of quantum mechanics methods is hampered by the complexity of the calculations resulting from the increased numbers of electrons associated with metal containing compounds. The main problems lie in the need for descriptions of the wave functions for the transition metal d-orbitals which are not computationally prohibitive. When this problem is addressed, semi-empirical quantum mechanics methods have been shown to be of use. With the advent of improved computational power, the situation is improving, but it is still fair to say that the quantum mechanics capabilities are still far behind those available for organic molecules of similar numbers of atoms.

A more common approach has been the application of molecular mechanics methods to the prediction of this type of unit. However, this is also not without difficulties. These compounds present huge problems for molecular mechanics in terms of their inherent geometry, even before distortions from regular arrangements are considered. An equally important problem arises when one attempts to parameterise force fields for these systems. These issues are explored more fully in Chapter Two.

Whilst these problems are not insurmountable, it is fair to say that they have seriously hampered the routine application of molecular mechanics to compounds containing transition metal atoms.
1.2.7 Summary
Research into molecular modelling has provided the chemist with a series of tools which enable him to predict both molecular structure and some properties. As research proceeds and computational power increases whilst costs decrease, the expectation must be that further advances towards improved predictive tools will be made.

However, there is no single computer program which may be applied universally to any problem. A certain amount of background knowledge about the limitations of each program and the molecular system under investigation is vital for the correct tool to be chosen. Even with this knowledge, further modifications may be necessary in order to use the program to effectively predict the chosen molecule’s structure.

In recent years, the focus of research into molecular modelling techniques has shifted from prediction of structure towards property prediction (although, of course, the latter does depend upon the former), and from organic molecules to crystalline organic and inorganic materials. One of the remaining hurdles for the computational chemist is the prediction of the three-dimensional structure of crystalline materials.

1.3 Crystal Modelling

In 1988, John Maddox stated “One of the continuing scandals in the physical sciences is that it remains in general impossible to predict the structure of even the simplest crystalline solids from a knowledge of their chemical composition.” Eight years on, an article in the same journal, (Ball, 1996), states that “in large part the
scandal remains”, and this is why the crystalline materials widely used in technological applications have been discovered more by accident than by design.

However, it would be wrong to say that computational chemists have not risen to the challenge, and a variety of techniques are available to tackle certain aspects of crystal design. The drawback is that, in order to develop most of these techniques, large areas of the periodic table have been discounted. In common with the modelling of isolated molecules, the area which causes most difficulty is inorganic chemistry, and in particular, metal complexes.

More success has been seen with the prediction of organic crystals. By careful design, organic ‘building blocks’ may be proposed such that there are at most only a few (and preferably only one) way in which they can join together. For example, one could envisage a diamondoid lattice constructed from template molecules designed to self assemble in a tetrahedral geometry. They should then self-assemble, precipitating from solution as the crystal structure of choice. However, even this approach is not without difficulties, and problems of interpenetrating networks, polymorphism and the inherent lack of stability (thermal, mechanical and/or chemical) associated with organic crystals still need to be resolved.

Given the problems associated with ab-initio crystal design, computational chemists have shifted their emphasis towards ‘Crystal Engineering’. This approach involves the modification of known compounds in small ways (e.g. changing metal centres, minor modifications to ligands or counter ions). The new crystal is then allowed to ‘relax’ into a minimum energy structure, assuming the space group remains unaltered. This approach is obviously not without problems - not least of which is the assumption that the new structure retains the space group of its
parent. Cole (1992) discovered that, in terms of packing energy, many 'local minima' exist in close proximity to each other - even though, in reality, the compound may show extremely clear preferences towards one particular configuration.

Whilst there must be some relationship between molecular and crystal structure, its nature still remains undiscovered. Until a reliable model for this relationship is proposed, the difficulties associated with predicting molecular crystals will remain.

1.3.1 An Approach to Modelling Inorganic Molecular Crystals

Modelling inorganic molecular crystals can be treated as a series of discrete steps:

1. Model the molecular unit(s) within the crystal (i.e. the asymmetric unit).
2. Propose which crystallographic space group the compound will crystallise in.
3. Predict the values of the unit cell parameters.

Modelling the Molecular Unit

Methods for modelling the molecular unit of an inorganic compound have been touched upon in this chapter and will be explored more fully in Chapter Two.

A more fundamental problem is the prediction of which compound should be modelled. In organic chemistry, reaction mechanisms are extremely well defined, and Computer Aided Organic Synthesis (CAOS) programs have been developed which are capable of proposing synthetic routes for a given target molecule. In comparison, transition metal chemistry is less clearly understood. It can be difficult to predict which ligands in a reaction mixture will coordinate to a particular metal centre, and more importantly, in which arrangement. This step is obviously of vital importance - unless the chemist can accurately propose a
compound which may be synthesised, any attempts to optimise its structure and predict its properties are worthless.

A prime example of the differing properties this type of isomerism can cause is exhibited by the square-planar [PtCl₂(NH₃)₂] complexes. The cis- has the best clinical properties of hundreds of platinum compounds in terms of antitumour activity. The trans- isomer, however, is essentially ineffective (Cotton, 1980).

Predicting the Space Group
Once the structures of the molecular unit(s) have been proposed, the next step is to decide how they will be arranged within the crystal lattice. This involves predicting how the molecules will orient themselves in relation to one another — especially important if there is more than one entity in the structure, such as a counter-ion. Once this arrangement has been decided upon, the crystallographic space group must be chosen. The only available source of information for this process is existing crystal structures. Structurally similar molecules may provide a starting point for predicting the space group, but this method is by no means infallible.

Optimising Unit Cell Parameters
Once again, structurally similar molecules can provide an idea of the values of the unit cell parameters. There are obviously minimum values that the parameters can be assigned due to the physical size of the unit cell contents. The lattice energy can then be monitored as the cell parameters are altered in an attempt to find an energy minimum.
Summary
At present, the tools required to predict much of this information are not available to the chemist - it is not known whether rules exist which will allow such information to be gleaned from sources such as the Cambridge Structural Database. Work by Wright (1994) suggests that artificial intelligence is not viable for predicting this kind of information, as the AI rules which must be defined are so rigid that one effectively defines each individual entry explicitly. Of course, this poses the question as to whether the ‘correct’ rules were postulated. What can be concluded is that there is a vast amount of chemical structural information available which currently cannot be used rationally to predict crystal structures.

1.4 Layout of Thesis
The layout of the remainder of this thesis is as follows:

• Chapter 2 describes how molecular mechanics can be applied to metal complexes and covers the development and investigation of a new technique, the application of a genetic algorithm, to the problem of generating force field parameters for unparameterised structures.

• Chapter 3 is concerned with some of the modifications that may be made to improve an IBM compatible personal computer in terms of computational power and graphics performance.

• Chapter 4 discusses the techniques which are available or desirable for modelling the structure and properties of molecular compounds in the crystalline state.

• Chapter 5 offers some concluding comments regarding the work presented in the preceding chapters.
A Genetic Algorithm for Generating Molecular Mechanics Force Field Parameters

2.1 Introduction
2.2 Genetic Algorithms
  2.2.1 Principles of Genetic Algorithms
    2.2.1.1 Introduction
    2.2.1.2 A Simple GA
    2.2.1.3 Initialisation
    2.2.1.4 Evaluation
    2.2.1.5 Reproduction
    2.2.1.6 Termination
    2.2.1.7 Summary
  2.2.2 Genetic Algorithms in Chemistry
2.3 Molecular Mechanics
  2.3.1 The Standard Force Field
    2.3.1.1 Bond Stretch and Angle Bend Energies
    2.3.1.2 Torsion Angle Energy
    2.3.1.3 Non-Bonded Interactions - van der Waals’ Strain
    2.3.1.4 Non-Bonded Interactions - Electrostatic Interactions
  2.3.2 Additional Terms in the Energy Expression
    2.3.2.1 Bond Stretching
    2.3.2.2 Hydrogen Bonds
    2.3.2.3 Out-of-Plane Bending
    2.3.2.4 Cross Terms
  2.3.3 Specialised Force Fields
    2.3.3.1 MM2 and Derivatives
    2.3.3.2 Biological Force Fields - AMBER and CHARMM
    2.3.3.3 Dreiding Force Fields
    2.3.3.4 Burchart
    2.3.3.5 Universal Force Field
2.3.4 Energy Minimisation Techniques
2.3.4.1 Non-Derivative Optimisation - Simplex
2.3.4.2 First Derivative Optimisation - Steepest Descents
2.3.4.3 First Derivative Optimisation - Conjugate Gradients
2.3.4.4 Second Derivative Optimisation - Newton-Raphson

2.3.5 Parameterisation Methods

2.4 Application of Molecular Mechanics to Coordination Compounds
2.4.1 Introduction
2.4.2 Problems With Conventional Force Fields
2.4.2.1 Parameterisation
2.4.2.2 Geometrical Difficulties
2.4.3 Methods Developed
2.4.3.1 The SHAPES Force Field
2.4.3.2 MM2MX-RIPS
2.4.3.3 Modified MM2
2.4.4 Examples
2.4.4.1 Zirconocene Ziegler-Natta Catalysts
2.4.4.2 Conducting Complexes
2.4.4.3 Applications of MM2MX-RIPS

2.5 Automatic Parameterisation of Molecular Mechanics Force Fields
2.5.1 Introduction
2.5.2 Examples

2.6 A Genetic Algorithm for Determining Force Field Parameters
2.6.1 Program Design
2.6.1.1 Introduction
2.6.1.2 Overview
2.6.1.3 Outline of the Algorithm
2.6.1.4 Summary
2.6.2 Initial Investigations: Straight-Chain Alkanes
2.6.3 Optimising the GA
2.6.3.2 Hardware
2.6.3.3 Software
2.6.3.4 Conclusions
2.6.4 Investigation of Optimum Mutation Rate
2.6.4.1 Determination of Convergence
2.6.4.2 Results
2.6.4.3 Conclusions
2.6.5 Determination of Optimum Convergence Threshold
2.6.5.1 Results
2.6.5.2 Conclusions
2.6.6 The Effect of Including Energy in the Score
2.6.6.1 Results
2.6.6.2 Conclusions
Chapter 2: A GA for Generating Force Field Parameters

2.1 Introduction

The preferred method for accurately modelling the molecular structure of metal containing compounds is one of the variations of quantum mechanics. However, as previously discussed, this technique is currently unsuitable for routine use due to the limitations of computational hardware. Work is proceeding in this area to improve the techniques and basis sets (Hansen, 1990; Li, 1992; Bock, 1993, Broo, 1996, Couty, 1996, Thomas, 1997) and the number and complexity of these applications will steadily increase with computational advances.

Until appropriate computer hardware is commonly available many computational chemists have turned to employing molecular mechanics as a substitute. Whilst the basic technique has inherent problems when coordination compounds are considered, by careful modifications to the algorithm and judicious selection of the systems studied, the method can be successfully applied to this type of problem.

The results from any molecular mechanics calculation are highly dependent upon the parameter set or force field used to model the molecule. The derivation of these parameters can be extremely time consuming - increasing rapidly as molecular structure becomes more complex. An automated approach to force field generation is therefore a highly desirable tool for the molecular mechanics modeller.

2.2 Genetic Algorithms

A variety of methods can be used to optimise a solution to a problem computationally. Typical examples include hill descending (or climbing) techniques and simulated annealing. A technique proposed by John Holland in the
early 1970s that has steadily gained popularity, especially since the early 1990s, is
the genetic algorithm (GA) - a variety of references about the subject have been
It was proposed that such an algorithm could be used to automatically generate or
optimise molecular mechanics force fields.

2.2.1 Principles of Genetic Algorithms

2.2.1.1 Introduction
The concept of GAs is based upon the Darwinian principles of evolution and
survival of the fittest. The basing of an optimisation technique on natural
phenomena draws similarities with neural networks and simulated annealing.

A genetic algorithm attempts to solve a problem by first randomly generating a set
of ‘solutions’ to the problem. Each ‘solution’ is tested - the better ones contribute
to the next cycle. Eventually after many cycles, the GA should converge on what it
‘believes’ is the most appropriate solution. Standard GAs have no real knowledge
of the problem - that is dealt with in the evaluation (test) routine.

Much of the terminology associated with GAs has been imported from the field of
genetics. The ‘solutions’ of the problem are known as the population and each
cycle is known as a generation. Individuals are known as chromosomes and part of
a chromosome which forms a ‘building block’ towards the solution is known as a
schema. The value denoting an individual’s chance of contributing to the next
population is known as its fitness. When an individual contributes to the next
generation, it often reproduces with other solutions, exchanging its ‘genetic’
material. Solutions are also prone to mutation.
Generally accepted features of evolution which apply to GAs include:

- Evolution acts upon chromosomes, not upon the entire entity.
- Natural selection links chromosomes to the performance of their decoded structures. It causes the chromosomes which encode successful features to reproduce more frequently than those that do not.
- Evolution occurs via reproduction. During reproduction, the recombination of genetic material from two parents may create chromosomes in the children which are quite different. Further differences may arise due to mutations.
- Biological evolution has no memory. All knowledge about producing new individuals that are well adapted to their environment is contained within the gene pool of the current population.

2.2.1.2 A Simple GA

A simple GA as defined by Davis, (1991) is illustrated in Figure 2.1.

Figure 2.1 - Schematic of a General GA
Although most GAs exhibit most, if not all, of the procedures outlined above, their implementations may be extremely different. This is because there are no hard and fast rules in the development of GAs - it is usual for the principle to be tailored to suit the particular optimisation problem. These points are explored below:

**2.2.1.3 Initialisation**

The first step in the operation of the genetic algorithm is the initialisation of the population. There are several important points which must be considered:

One of the most fundamental problems is the way the population is going to be encoded. The first genetic algorithms operated upon simple strings of binary digits. Whilst this approach works well if the solution to the problem is relatively simple (for example, a single integer or floating point number), representing complex solutions in this way can be more difficult.

Bit string representations do have some advantages - they are simple to initialise and concepts such as crossover and mutation can be applied easily. In principle, one standard GA could be applied to any problem, as long as the proposed solutions are binary encoded. When binary strings are used, the string length must be decided since this dictates the precision of the result.

An alternative approach is to use real numbers. From a practical point of view this can have several advantages. Firstly, a real number representation is likely to be the one used in reality. It is also easier for the user to understand how the GA is proceeding as potentially complicated decoding is not necessary. This can also result in time savings, as the computational overhead of decoding the solutions is decreased.
This type of representation, known as a hybrid genetic algorithm, has been shown to be faster, more consistent between runs and provides higher precision for some problems (Janikow, 1991). If a real number representation is used, however, the reproduction operators (crossover and mutation) must be modified, abandoning the concept of problem-independence. Researchers who apply GAs to real world problems see this as an advantage since they may be able to design operators which are more specific to their problem.

Once the encoding has been decided, the population can be initialised. If a binary string of length n digits is being used, initialisation is simply a case of choosing 1s or 0s at random for each position on the string. The initialisation of floating point representations involves generation of a random number between set limits.

The final decision is the size of the population. If too large, the population can be slow to evolve, whilst a small population may converge too quickly. If run time is important, the correct balance between number of members and length of run must be achieved. Goldberg (1989b) reports on the theory of optimum population size. Unfortunately, much of this work is based upon binary populations, although they may in theory be applied to other encodings.

2.2.1.4 Evaluation
The evaluation section requires knowledge about the problem - each member of the population is evaluated and a score is allocated. This score may be equal to the fitness for each member (known as fitness-is-evaluation). However there are a variety of reasons why this may be inappropriate and in these cases a scaling factor is required. One of the most fundamental reasons for scaling the evaluation occurs when the individual’s fitness is inversely related to its evaluation (i.e. the fittest members of the population have the lowest score).
Other reasons for scaling relate to the relative scores of the population’s members - GAs may give rise to two phenomena - the Super Individual and the Close Race:

- A Super Individual is a member of the population whose evaluation far exceeds that of its nearest rival. With fitness-is-evaluation, this solution rapidly dominates the population within a couple of generations. However, this individual may be far removed from the optimum solution to the problem. If it dominates the population, it will have little chance to exchange information with other individuals, leading to the population’s rapid convergence.

- When several individuals have similar evaluations the population experiences a Close Race. If fitness is allocated directly according to evaluation in this case, there will be very little bias towards the best individuals. GA’s often experience a Close Race towards the end of a run.

The effects of both of these phenomena on a GA’s population can be reduced either by creating a simple mathematical relationship between the evaluation and fitness, (for example, letting fitness = evaluation$^2$), or through the use of fitness normalisation methods, such as Windowing or Linear Normalisation:

In order to apply Windowing, the individual with the minimum evaluation must first be identified. This minimum value is then subtracted from all members of the population - the remainder is taken as the individual’s fitness. A direct consequence of this is that the weakest individual has a fitness of zero - hence it has no chance of contributing to the next generation. If required, a higher value can be applied with any individuals falling below this value being assigned it, ensuring they have a chance of reproducing.

Linear Normalisation operates by ordering the individuals by decreasing evaluation. The fitness of the best individual is assigned as a constant value. The next individual has a fitness that is a fixed amount less than the best, and so on -
the initial constant value and the rate of decrement are required as parameters of this technique. Once again, an optional minimum value can be assigned, thus ensuring that all members of the population will be given a chance of reproducing.

The effects of applying fitness-is-evaluation, Windowing and Linear Normalisation to a population experiencing a Super Individual and a Close Race are illustrated in Table 2.1. The evaluations for six individuals (A-F) are presented. The columns labelled Fit represent the fitness of the individual calculated by each fitness technique, and those labelled P give the probability that an individual will contribute to the next generation (calculated by dividing the individual’s fitness by the sum of the population’s fitnesses). Two examples of Windowing and Linear Normalisation are given, each using different parameters.

**Table 2.1 - Effect of fitness operators on a GA’s population.**

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness is Evaluation</td>
<td>25</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0.463</td>
<td>0.167</td>
<td>0.148</td>
<td>0.130</td>
<td>0.074</td>
<td>0.019</td>
</tr>
<tr>
<td>Windowing No Minimum</td>
<td>24</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.500</td>
<td>0.167</td>
<td>0.146</td>
<td>0.125</td>
<td>0.063</td>
<td>0.000</td>
</tr>
<tr>
<td>Windowing Minimum = 5</td>
<td>24</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>0.436</td>
<td>0.145</td>
<td>0.127</td>
<td>0.109</td>
<td>0.091</td>
<td>0.091</td>
</tr>
<tr>
<td>Linear Norm. Max. = 60, Decrement = 1</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>0.174</td>
<td>0.171</td>
<td>0.168</td>
<td>0.165</td>
<td>0.162</td>
<td>0.159</td>
</tr>
<tr>
<td>Linear Norm. Max. = 60, Decrement = 10</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>0.286</td>
<td>0.238</td>
<td>0.190</td>
<td>0.143</td>
<td>0.095</td>
<td>0.048</td>
</tr>
</tbody>
</table>

In this example, the original evaluations for each member of the population display the two phenomena previously discussed - A is a super individual and B, C and D exhibit a close race. This is reflected by the high probability achieved by A and the approximately equal probabilities shown by B, C and D in the
Fitness-is-Evaluation treatment. When Windowing is applied, both phenomena still exist, but their effects are reduced, especially when a minimum value is applied. Linear Normalisation decreases the dominant selection for the Super Individual - although it obviously still has the largest fitness score. This technique also heightens competition in a close race. In this method, the value of the decrement parameter is of prime importance - the larger the decrement, the greater the selection pressure in favour of the Super Individual.

Once the fitnesses of the members of the population have all been evaluated, the next generation can be created.

2.2.1.5 Reproduction

The first decision which must be made in terms of reproduction is the number of new solutions required. Simple GAs replace every member of the current population each generation. This can cause some problems; due to the random nature of the reproduction methods, the best member(s) of the population may not reproduce. Alternatively, the reproduction operators (crossover, mutation and/or inversion) may alter the ‘good’ sequences of information in the parents. Either way, information stored within the best members of the population will be lost. One way to counter this is to use an elitist strategy, where the best individual is always copied unaltered to the next generation, ensuring its survival.

Even with an elitist strategy, many of the best solutions may well be lost with each generation. Steady-state genetic algorithms, (Syswerda, 1989), only allow a small number of new individuals to be created each generation. The individuals which are lost are selected through inverse ranking (i.e. the worst individuals are removed). The benefits of this technique are:
1. Good members of the population float towards the top of the list, where they won’t be deleted (parents are never deleted).

2. Poor individuals sink to the bottom, where they are likely to be removed. However, they are not prevented from being parents and, if this is the case, they will not be deleted.

These characteristics automatically provide elitism and allow for a higher error rate (i.e. a more aggressive learning rate - for example, higher mutation), whilst protecting what is good in the population.

An extension of this technique is steady-state without duplicates (Davis, 1991). In this strategy, each new individual is compared with existing members of the population. If it is a duplicate it is rejected and another individual is created, thus guaranteeing greater genetic diversity. Although this techniques has an additional computational overhead, the time spent checking for duplicates is usually negligible when compared to the overall evaluation time for a real-world problem.

**Roulette-Wheel Selection**

Once the number of new solutions is established, the parents of these solutions must be selected - the most common method being roulette-wheel selection. In order to use roulette-wheel selection, the fitness scores for solution must first be summed to give the total fitness. A random number is then generated between zero and the total fitness. The first individual whose fitness added to the fitnesses of the preceding members is greater than or equal to the random number is returned. This process is repeated once for each new individual that is going to be created. The chosen solutions, or parents, are collectively known as the breeding pool. The fitness-is-evaluation data in Table 2.1 is used in Table 2.2 and Figure 2.2 to illustrate the procedure.
Table 2.2 - Fitness data used for roulette-wheel selection.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness</td>
<td>25</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Sum of Fitness and Preceding Fitnesses</td>
<td>25</td>
<td>34</td>
<td>42</td>
<td>49</td>
<td>53</td>
<td>54</td>
</tr>
</tbody>
</table>

Figure 2.2 - The roulette-wheel resulting from the data in Table 2.2.

Stochastic remainder selection (Cartwright, 1993) ensures that all solutions with above average scores contribute whilst still relying to some extent upon chance to create the breeding pool. The stages in stochastic remainder selection are:

1. Scale all chromosomes such that the average fitness is 1.0.

2. For each string whose fitness is above average, make copies equal to the integer part of the string into the breeding pool, i.e. a solution with a score of 2.3 will have two copies made. Subtract the integer part off of the fitness value for each solution giving all chromosomes in the population a remainder < 1.
3. Select strings from the old population at random. Each time a string is selected, a random number (between 0 and 1) is also generated. If the random number is less than the fitness remainder, copy the string to the breeding pool and set the fitness to zero, ensuring that no further copies can be made. If the random number is greater than the remainder, choose another string.

4. Continue until the correct number of strings are in the breeding pool.

A variety of operators can then be applied to the parents in order to exchange or alter their information, creating the next generation. The three best known operators are mutation, crossover and inversion (Holland, 1975).

**Mutation**

The mutation operator takes a probability of mutation occurring as a parameter. Each bit is considered in turn - a random number between 0 and 1 is generated and if this value is less than the mutation probability, the bit mutates. When mutation occurs, there are two possibilities: the value of the bit may be automatically flipped - a one becomes a zero and vice versa or the new mutated value may be randomly generated, effectively halving the mutation rate.

Real number mutation is equivalent to bit string mutation - the difference being that the whole number is changed rather than simply one binary digit. It changes fewer chromosomes since only one probability test is applied to each real number. An alternative operator is Real Number Creep. If a chromosome is reproducing it is likely that it is in the vicinity of the “best” solution. Real number mutation could destroy these numbers. Instead, if the mutation test is passed, a small, randomly generated value is added to (or subtracted from) the number. The mutation rate and the range of the creep value are parameters of this operator.
Crossover

One of the most important and distinguishing components of a GA is the crossover operator. In crossover, information from two (or more) individuals is exchanged to form new individuals. All other GA operators operate upon a single chromosome.

Crossover will therefore combine good features in individual chromosomes - a process which is likely to produce even better solutions. It is far less likely that these features will appear on the same chromosome through mutation.

There are a variety of different crossover operators - all combine schema from different individuals but the methods of combination vary.

- **One-Point Crossover** is the simplest form of crossover. Two parents are randomly selected from the breeding pool. A position along the length of the chromosome is chosen, also at random. The information contained on each parent beyond this cut is then swapped. This is illustrated in Figure 2.3.

  Parent A: 1 1| 1 1 1 1  Child A: 1 1| 0 0 0 0

  Parent B: 0 0| 0 0 0 0  Child B: 0 0| 1 1 1 1

  \[ \Rightarrow \]

  **Figure 2.3 - One-point crossover.**

Whilst one-point crossover is an effective way of combining features which occur upon distinct chromosomes, there are some schemata, which are impossible to combine, as shown in Figure 2.4. Other forms of crossover have been developed to deal with this problem.
Figure 2.4 - Two chromosomes with schemata which one-point crossover cannot combine. The important features are marked in bold type (e.g. 1).

- **Two-Point Crossover** also works on two parents, but two cut points are selected instead of one. The information between the two cuts is swapped to form the new children, as shown in Figure 2.5.

  
  \[
  \text{Parent A: } 1 \ 1 \ 1\vert \ 1 \ 1\vert \ 1 \ 1 \quad \text{Child A: } 1 \ 1 \ 1\vert \ 0 \ 0\vert \ 1 \ 1 \\
  \text{Parent B: } 0 \ 0 \ 0\vert \ 0 \ 0\vert \ 0 \ 0 \quad \Rightarrow \quad \text{Child B: } 0 \ 0 \ 0\vert \ 1 \ 1\vert \ 0 \ 0
  \]

  Figure 2.5 - Two-point crossover

However, certain schemata still cannot be combined (Figure 2.6).

\[
\text{Chromosome A: } 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\
\text{Chromosome B: } 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0
\]

Figure 2.6 - Chromosomes with schemata which neither one- or two-point crossover can combine. Important features are marked in bold type (e.g. 1).

- **Uniform Crossover** can combine good schemata on two parents regardless of where on the chromosomes this information lies (as long as the information isn’t coincident). Two parents are selected from the breeding pool as before, but in uniform crossover, each bit position on the chromosome is considered in turn to randomly determine whether information for child A should be taken from parent A or parent B. Child B receives the opposite information. This principle is shown in Figure 2.7.
Whilst uniform crossover can combine some features which cannot be combined by one- or two-point crossover, it can also be extremely destructive to schemata which are compactly encoded. However, Syswerda (1989) and Spears and De Jong (1991) have shown that for some problems, the ability to combine features no matter where they are located on the chromosome can outweigh the global devastation possible.

More recent versions of crossover allow the possibility of orgies, (Michalewicz, 1996), where several parents contribute to each child. A variety of techniques have been reported, (Mühlénbein and Voigt, 1995, Eiben et al., 1994), however these operators have yet to gain wide acceptance.

- **Average Crossover**, (Davis, 1991), is an alternative form of crossover which is possible when the chromosomes use real number encoding. This operator requires two parents, and takes the average of each field, producing one child.

**Inversion**

Inversion operates on one chromosome by inverting the order of elements between two randomly selected points. Whilst it may be useful when each chromosome contains the same set of elements and the object of the optimisation is to determine
the 'best' order, (i.e. a travelling salesman problem), it has not generally been found useful in GA practice. Some researchers believe that it will become more important as the lengths of chromosomes increase (Davis, 1991).

**Operator Parameter Values**

Many of the operators require parameters to be defined within the GA. These values can have a profound effect upon the GA's performance. Most GAs are, at least initially, parameterised according to the judgement of the developer. As the GA is used these parameters may be hand optimised to give improved results.

Sophisticated techniques for optimising parameter values have been developed, including the use of another GA. However, Davis notes that the time taken to find optimal parameter settings for a particular problem may be orders of magnitude greater than the time required to run the GA.

As a GA proceeds, the nature of its population changes, suggesting that operator parameters should adapt. In the early stages, a high crossover probability is essential to gather useful pieces of information from the different solutions. Mutations may also create good schema but are also likely to destroy the only copy of good schema - for this reason the mutation operator needs to be kept low. Later, the population converges into a small section of search space. Mutation enables searching for improved solutions in the region of the best current individuals, so the probability of mutation should increase as a run proceeds.

The simplest way to interpolate operator fitnesses throughout the run is linearly. However, it has been shown (Davis, 1989) that the optimum values actually follow a curved trajectory as the run proceeds. A more complex system of adaptive operator fitness tracks the performance of operators over the period of the GA,
rewarding operators which ‘set the stage’ for improvements. Periodically, the operators’ parameters are updated, allowing them to track the trajectory.

2.2.1.6 Termination
Various criteria may be used to terminate the GA. The simplest and most common is to stop after a fixed number of generations. A similar criterion is that the program terminates after a set number of evaluations - giving a different result if the GA is operating under a steady-state strategy.

These termination criteria require knowledge of the characteristics of the problem which influence the length of the optimisation. A better approach could be for the algorithm to determine when no further significant improvements are likely and hence when it should terminate.

Another strategy is to check for convergence in each generation. Each field on the chromosome can be checked, and if a predetermined percentage of the chromosomes are identical (or similar in real number encoding), then that field can be said to have converged. If the percentage of converged fields exceeds another predetermined value the GA is terminated. An alternative approach measures the progress made by the algorithm in a predefined number of generations. If the progress is less than a set value, the search terminates.

2.2.1.7 Summary
The field of genetic algorithms has been growing rapidly since their inception in the early 1970s. Much effort has been spent researching GAs and their application to a wide range of theoretical and real world problems. The general strategies applied by genetic algorithms practitioners are rapidly diverging as workers tailor the principles of evolution and survival of the fittest to the solution of their own
problems. It is unlikely that the problem-independent (i.e. non-evaluation) part of any two independently developed GAs will follow exactly the same procedures.

Genetic algorithms are significantly different from other optimisation methods. Major differences include:

Genetic algorithms have many potential solutions searching. However, unlike performing several hill-climbing optimisations from different starting positions, the GA’s solutions are not independent. Crossover ensures that when one solution is searching in a better environment, this information is disseminated through the other chromosomes.

Genetic algorithms attempt to solve a problem by relying upon chance. Hill climbing algorithms follow set rules, moving up the gradient until a maximum is reached. Whilst this works well on simple surfaces, severe difficulties are encountered when the surface contains many maxima. A hill climbing algorithm also always follows the same path if started from the same point. Because the GA relies upon random numbers in its operation it is less likely to continue falling into the same trap. Even if it starts converging in the wrong place there is a chance that mutation or crossover will produce an individual in the correct search area which will rapidly lead the whole algorithm in the right direction. Each run of the GA should achieve the same result (approximately for real-number encodings), but the route taken to get there should always be different.

Unlike many other methods, the GA doesn’t need to know any information about the search space. This enables the GA to search surfaces which are not clearly defined, such as those which contain a significant amount of noise. A hill-climbing algorithm could have severe difficulties on this type of surface.
Genetic algorithms are ideally suited to parallelisation. For example, the program could be separated in the evaluation of each individual as this is independent of the other evaluations in a generation. An alternative approach which has found consistently better solutions than the standard GA is the distributed genetic algorithm (Tanese, 1989). The population is divided into a number of sub-groups - each sub-group is then evolved for a number of generations before the best individuals in each group are migrated to other populations. This technique can obviously be applied equally on a single- or multi-processing system. As genetic algorithms are employed to optimise increasingly complicated problems, the use of parallel implementations will become vital in maintaining reasonable run times.

A final comparison between hill-climbing, simulated annealing and GAs appeared on the Internet (Sarle, 1993) and has been reproduced in Michalewicz (1996):

"Notice that in all [hill-climbing] methods discussed so far, the kangaroo can hope at best to find the top of a mountain close to where he starts. There's no guarantee that this mountain will be Everest, or even a very high mountain. Various methods are used to try to find the actual global optimum.

In simulated annealing, the kangaroo is drunk and hops around randomly for a long time. However, he gradually sobers up and tends to hop up hill.

In genetic algorithms, there are lots of kangaroos that are parachuted into the Himalayas (if the pilot doesn't get lost) at random places. These kangaroos do not know that they are supposed to be looking for the top of Mt. Everest. However, every few years, you shoot the kangaroos at low altitudes and hope that the ones who are left will be fruitful and multiply."
2.2.2 Genetic Algorithms in Chemistry

Genetic algorithms have found a variety of applications in chemistry, including:

- The analysis of the distribution of airborne pollution, (Cartwright, 1992) - allowing the solution of many-source/many-receptor problems with results significantly better than previous models.
- The solution of chemometrics problems through the development of GATES (Genetic Algorithm Toolbox), (Lucasius, 1989, 1994a, 1994b).
- Protein structure prediction and ligand docking (Khimasia, 1997; Dandekar, 1997 and Oshiro, 1995).
- Determination of crystal structures from powder diffraction data by fitting experimental data to calculated patterns (Shankland, 1997, Paszkowicz, 1996).
- Generation of candidate structures for inorganic crystals (Bush, 1995). These candidates are then minimised using standard lattice energy methods.

This list is far from exhaustive; GAs are finding uses in all major branches of chemistry.

2.3 Molecular Mechanics

Many texts have been published which discuss molecular mechanics in great detail (Rappé, 1997, Leach, 1996, Bowen, 1991, Burkert, 1982). Most commercial molecular modelling packages also include details of their implementations of molecular mechanics. The basic theory is straightforward - by treating atoms as rigid spheres and bonds as springs, a molecule’s energy may be described by a series of mathematical functions based upon classical mechanics - a concept first proposed as early as 1930 (Andrews, 1930). This collection of mathematical expressions is known as the energy expression and the parameters pertaining to the functions for each particular atom type are known as the force field.
Chapter 2: A GA for Generating Force Field Parameters

Molecular mechanics is an empirical technique and it is not suggested that the equations employed to model each component of the potential are accurate descriptions of the molecule’s energy. For this reason, the form of the energy expression vary from force field to force field. However in general, all the equations include terms describing the energy associated with bonds, bond angles, torsion (or dihedral) angles and non-bonded interactions.

Even the simplest molecular mechanics force field (i.e. one derived for a set of molecules containing a limited number of elements) contains relatively large amounts of information.

![Figure 2.8 - Chloroethane](image)

If a simple molecule such as chloroethane, (Figure 2.8), is considered, parameters for the following are required (Table 2.3):

<table>
<thead>
<tr>
<th>Bonds</th>
<th>Angles</th>
<th>Torsions</th>
<th>Non-Bonded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cl - C</td>
<td>Cl - C - H</td>
<td>Cl - C - C - H</td>
<td>Cl - C</td>
</tr>
<tr>
<td>C - C</td>
<td>Cl - C</td>
<td>H - C - C - H</td>
<td>Cl - H</td>
</tr>
<tr>
<td>C - H</td>
<td>H - C - C</td>
<td>H - C - H</td>
<td>Cl - Cl</td>
</tr>
<tr>
<td>H - C - H</td>
<td></td>
<td></td>
<td>C - C</td>
</tr>
</tbody>
</table>

Table 2.3 - N.B. All Carbon atoms are assumed sp^3 hybridised. Non-bonded parameters other than Cl - H and H - H are only applicable between molecules, since intramolecular non-bonded interactions are in general only considered when atoms are separated by three or more bonds.
If an oxygen atom is inserted to form 2-Chloroethanol, (Figure 2.9), the number of force field parameters required to model the molecule grows as shown (Table 2.4).

Table 2.4 - N.B. All Carbon and Oxygen atoms are assumed sp³ hybridised. Non-bonded parameters other than Cl - H, Cl - O, H - O, C - H and H - H are only applicable between molecules since intramolecular non-bonded interactions are generally only considered for atoms separated by three or more bonds, since the electrostatic and van der Waals' interactions for closer atoms are implicitly included in the bond stretching and angle bending parameters.

<table>
<thead>
<tr>
<th>Bonds</th>
<th>Angles</th>
<th>Torsions</th>
<th>Non-Bonded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cl - C</td>
<td>Cl - C - H</td>
<td>Cl - C - C - H</td>
<td>Cl - C</td>
</tr>
<tr>
<td>C - C</td>
<td>Cl - C - C</td>
<td>Cl - C - C - O</td>
<td>Cl - H</td>
</tr>
<tr>
<td>C - H</td>
<td>H - C - C</td>
<td>H - C - C - H</td>
<td>Cl - O</td>
</tr>
<tr>
<td>C - O</td>
<td>H - C - H</td>
<td>H - C - C - O</td>
<td>C - O</td>
</tr>
<tr>
<td>O - H</td>
<td>H - C - O</td>
<td>C - C - O - H</td>
<td>C - H</td>
</tr>
<tr>
<td></td>
<td>C - O - H</td>
<td>H - C - O - H</td>
<td>H - H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>H - O</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cl - Cl</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C - C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>O - O</td>
</tr>
</tbody>
</table>

Parameters must be derived for each element in each hybridisation state. For example, a force field which is parameterised for all forms of carbon must contain information to deal with its three hybridisation states - sp³, sp² and sp. A separate atom type may also be defined to deal with aromatic carbons.

By evaluating the energy expression and its first derivatives, the chemist can gain an insight into the energy of the model and the forces acting upon it.
2.3.1 The Standard Force Field

A typical energy expression resembles the following (Equation 2.1), although there are likely to be some modifications.

\[
\text{Energy} = \sum_{\text{bonds}} \frac{k_b}{2} (b - b_o)^2 + \sum_{\text{angles}} \frac{k_\theta}{2} (\theta - \theta_o)^2 \\
+ \sum_{\text{torsions}} \frac{V_n}{2} (1 + \cos(n\phi - \gamma)) \\
+ \sum_{\text{van der Waals' strain}} 4\epsilon \left[ \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] \\
+ \sum_{\text{charge interactions}} \frac{q_i q_j}{\varepsilon_0 r_{ij}}
\]

The individual terms can be explained as follows:

2.3.1.1 Bond Stretch and Angle Bend Energies

\[
\text{Bond Stretch} = \frac{k_b}{2} (b - b_o)^2 \\
\text{Angle Bend} = \frac{k_\theta}{2} (\theta - \theta_o)^2
\]

Bond stretch and angle bend energies are both described using a simple harmonic function. This is appropriate for small deformations from the equilibrium value - for larger deformations, more advanced force fields use modified functions.

In the equations above, \(k_b\) and \(k_\theta\) are the force constant for the bond or angle, \(b\) and \(\theta\) refer to the observed bond length or angle in the molecule, and \(b_o\) and \(\theta_o\) refer to the equilibrium for the bond length and angle. The force constants are
commonly obtained from infra-red and/or Raman spectroscopic data. The equilibrium values for the bond length and angle are calculated by averaging the values from numerous crystallographic studies of the bond or angle. $k_b$ values are lower than $k_\theta$ because angles are easier to deform than bonds.

### 2.3.1.2 Torsion Angle Energy

$$\text{Torsion Energy} = \frac{V_n}{2} (1 + \cos(n\phi - \gamma))$$

The torsion energy term is described by a periodic (cosine) function. Variations in torsion angle are one of the most significant reasons for differing geometries - indeed conformational searching of a molecule is carried out by altering the values of torsion angles.

Various mathematical expressions are used to describe the torsion energy term, but most of them are actually equivalent. In the above equation, $V_n$ is a constant relating to the barrier to rotation for the torsion angle, $n$ is the multiplicity of the torsion (how many minima the torsion experiences if it is rotated through 360°), $\gamma$ (the phase factor or origin) is the position where the torsion angle passes through its minimum value and $\phi$ represents the torsion angle itself. Figure 2.10 shows how different torsion angles may be described by using different values of $n$ and $\gamma$. 
Figure 2.10 - Variation of torsion angle energy for different values of $n$ and $\gamma$.

Double bond: $n = 2$, $\gamma = 180^\circ$, Single bond: $n = 3$, $\gamma = 0^\circ$.

In the graph, the torsional barrier constant, $V_n$, is ignored. The value of this constant gives the maximum height of the energy barrier - therefore $V_n$ is significantly higher for multiple bonds than for single bonds.

In an attempt to reduce the number of parameters required to describe the torsion angles present within a molecule, some force fields disregard the two outer atoms. This method can also be useful as a 'catch all' to assign roughly correct parameters to any torsions that are not explicitly defined.

2.3.1.3 Non-Bonded Interactions - van der Waals' Strain

\[
\text{van der Waals' Strain} = 4\varepsilon \left[ \left( \frac{\sigma_y}{r_{gy}} \right)^{12} - \left( \frac{\sigma_y}{r_{gy}} \right)^{6} \right]
\]
The van der Waals' interactions present within a molecule are commonly modelled using a Lennard-Jones potential - usually the 12-6 form. (Figure 2.11). The equation may be in the form shown, or expressed in terms of the interatomic separation at minimum energy, $r_m$.

**Figure 2.11 - The 12-6 form of the Lennard-Jones potential**

The Lennard-Jones equation has two parameters in the force field for each possible pair of interacting atoms. The collision diameter, $\sigma$, is the atomic separation at which the energy of interaction is zero, and the well depth, $\varepsilon$, is the value of the energy at the potential's minimum.
The Lennard-Jones potential is composed of two parts - an attractive part proportional to \( r^{-6} \) and a repulsive term proportional to \( r^{-12} \). The \( r^{-12} \) term has been found to be reasonable for rare gases and has the added advantage for large systems in that it can be easily calculated by squaring the \( r^{-6} \) term. For this reason, this form of the equation is commonly employed. However, it is inappropriate in some cases and other powers (e.g. 9 or 10) have been used for the repulsive term.

It should be noted that the tables of required force field parameters for chloroethane and chloroethanol are somewhat misleading. Since the determination of van der Waals’ parameters can be extremely time consuming, it is commonly assumed that parameters for interactions between unlike atom types may be obtained by mixing the parameters for ‘pure’ atoms. This reduces the parameters required to describe van der Waals’ interactions in a system containing \( N \) atom types from \( N(N+1)/2 \) to \( N \). In a further attempt to reduce the number of parameters needed, the same set of parameters are often used for all instances of a particular atom - regardless of its hybridisation.

\[ \text{Electrostatic Potential} = \frac{q_i q_j}{\varepsilon_0 r_{ij}} \]

Even though the overall charge on a molecular unit may be zero, electronegative atoms attract more electrons, giving rise to an unequal charge distribution. A common solution is to assign fractional point charges throughout the molecule,
then apply Coulomb’s law to each pair of atoms to calculate the interactions. In the equation shown, $q_i$ and $q_j$ refer to the charges on atoms i & j, and $\varepsilon_0$ is the dielectric constant (1 for a vacuum).

Various methods have been used to assign point charges: a number have been compared by Wiberg and Rablen, (Wiberg, 1993). The simplest method is to assign an approximate formal charge to each atom type. More accurate methods rely upon initially performing a single point quantum mechanics calculation on the molecule. The electron density over the molecule may be calculated from the resultant wavefunction. Population analysis is then used to assign a value, (not necessary integral), to each nucleus corresponding to the number of electrons associated with it. The first method to be developed, Mulliken analysis (Mulliken, 1955) assigns the charges rapidly, but has some shortcomings: some orbitals appear to contain a negative number of electrons and others more than two. More recent methods (Löwdin, 1970; Bader, 1985) produce results closer to the values a chemist would expect from experience.

The problem with these techniques is that they rely upon data generated from the initial structure. This may be far removed from the optimised structure so the charges may become incorrect as energy minimisation proceeds. The solution is to repeat the charge assignment procedure during the optimisation process - however the methods discussed rely upon performing a single-point quantum mechanics calculation which may be too time consuming to repeat.

This has lead to research into faster methods for calculating atomic charges. The Gasteiger and Marsili approach (Gasteiger, 1980), uses an iterative method to achieve partial equalisation of orbital electronegativity. In this calculation, the molecule’s conformation is unimportant - the method only considers the atoms
present and how they are connected. Account is taken, however, of hybridisation state as electronegativity varies with hybridisation.

An alternative approach implemented in Cerius² is the Charge Equilibration (QEq) method (Rappé, 1991). This method takes into account the topology of the molecule, the electronegativities of the constituent atoms and the molecular conformation; again the process is iterative. Parameter sets have been derived for systems containing metal ions, and it is therefore recommended for the treatment of organometallics. However it has been noted that using this method on molecules in high energy conformations can lead to unrealistic charges so a brief energy minimisation is recommended prior to performing QEq.

Electrostatic interactions are critical in predicting the structure of inorganics and in calculating the intermolecular behaviour (e.g. docking or packing) of organic molecules. Coulombic potentials operate over a long range - for a molecule containing N atoms there are N(N-1)/2 interactions which require evaluation. It is easy to see how this could be prohibitive even for the smallest protein or polymer structure. To circumvent this, many implementations apply a cut off function (8Å - 15Å) for atom pairs which are considered to be too far apart to be of significance. The cut-off distance must be selected with care such that the excluded interactions are negligible - a pair of unit charges separated by 100Å would still contribute 3.3 kcal/mol to the overall potential.

If even the most appropriate molecular mechanics force field is having difficulties predicting a molecule’s structure, there are three alternatives available:

- Add additional terms to the energy expression to take into account important interactions in the molecule (the additional force field parameters obviously must be derived).
Modify the form of the existing terms in the energy expression such that they better model experiment.

Modify the values of the parameters in the force field.

2.3.2 Additional Terms in the Energy Expression
Many force fields have evolved beyond the basic energy expression (Equation 2.1, page 52). In some cases, these modifications consist of changes or additions to the standard terms already described: for example, many employ a modified bond stretching term. Other changes include the addition of extra terms to describe other features of the structure. Typical interactions which have been added to include hydrogen bonding, out-of-plane bending and cross terms.

2.3.2.1 Bond Stretching
Although the most commonly used equation to model bond stretching is harmonic (quadratic), the function which most accurately describes the relationship between a bond’s energy and its length is the Morse potential (Equation 2.2).

\[
\text{Bond Energy} = D_0 \left[ e^{-\alpha (R - R_0)} - 1 \right]^2 - S, \quad \text{where } \alpha = \sqrt{\frac{K_b}{2D_0}}
\]  

In this equation, \(D_0\) represents the bond energy, \(R\) the interatomic separation (i.e. the bond length), \(R_0\) the equilibrium bond length, \(K_b\) the force constant and \(S\) a constant shift energy.

Whilst this equation may best describe the potential, it is not commonly used in molecular mechanics force fields, for the following reasons:

- It is harder to evaluate than some other functions used for bond stretching.
• The function requires three parameters, \( D_0, R_0 \) and \( K_b \) for each bond (whereas functions such as the harmonic only require two).

• As \( R \) tends towards infinity, the forces associated with the Morse potential tend towards zero, which makes the function inappropriate for molecules which have a poor initial geometry.

In an attempt to provide a representation of the bond energy that is more accurate than the harmonic function and is relatively easy to calculate, force field developers have looked at higher polynomials such as cubic and quartic equations.

Cubic potentials are closer to the Morse potential in the region of the equilibrium bond length, and although they require three parameters for each bond, they are relatively simple to calculate. However, cubic functions have a maximum turning point. Beyond this, as interatomic separation increases towards infinity the potential drops rapidly towards \(-\infty\). This makes them unsuitable for molecules which are far from their equilibrium structure, as the atoms are inclined to fly apart. In order to prevent this, the cubic function can be modified such that the maximum is converted to a point of inflexion. With this form, as \( R \) tends to infinity the potential energy also increases to infinity.

The harmonic (quadratic) equation and the two forms of the cubic function are compared graphically with the Morse potential in Figure 2.12.
2.3.2.2 Hydrogen Bonds

In standard force fields, hydrogen bonding is accounted for using van der Waals' or electrostatic terms. Whilst this may be appropriate for molecules where hydrogen bonding is insignificant in defining the structure, in cases where it is important the results are often unacceptable. One solution is to define the bond explicitly and use a harmonic bond stretching term. A more acceptable alternative is to use a specific function that describes the hydrogen bonds present in the molecule.

Many force fields that model hydrogen bonding use a Lennard Jones 12-10 potential. More advanced representations (YETI, Vedani, 1988; Dreiding-II, Mayo, 1990), are usually based upon this function but also take into account the geometry of the hydrogen bond, so are reliant upon the positions of hydrogen atom and the donor and acceptor atoms. Functions used to model hydrogen bonding
commonly employ a cut-off distance similar to those used for electrostatic interactions, reducing the number of calculations to be performed.

### 2.3.2.3 Out-of-Plane Bending

Out-of-plane bending terms are considered essential for force fields used for molecules containing sp\(^2\) hybridised carbon atoms. If such a molecule is strained, (e.g. in cyclic compounds), a standard force field may shift an atom out of the plane of the other three in an attempt to drive the angles towards 120°. An out-of-plane bending term takes into account the energetic benefits of maintaining the planarity of the sp\(^2\) carbon and hence its π-bonding.

The most common way of defining an out-of-plane bending term is to treat it as an ‘improper’ torsion angle (i.e. one where the atoms are not connected in sequence). The torsional term is then parameterised such that the improper torsion adopts a value of 0° or 180° (Equation 2.3).

\[
\text{Out-of-Plane Energy} = k(1 - \cos 2\theta) \tag{2.3}
\]

Two other ways of dealing with this interaction are to measure the height or angle of the deviating atom from the plane defined by the other three, as shown in Figure 2.13.

![Figure 2.13 - Two ways to describe out-of-plane bending](image)
Out-of-Plane Energy = \( \frac{k}{2} \theta^2 \) or \( \frac{k}{2} h^2 \)

The resulting value is then modelled using a simple harmonic potential, such as those described in Equation 2.4. In these equations, the out-of-plane interaction tends towards zero as the \( sp^2 \) centre approaches planarity.

### 2.3.2.4 Cross Terms

More recently, the importance of cross terms has been recognised. These represent coupling interactions within the molecule - for example, if a bond angle is reduced, the two bonds will lengthen in an attempt to maintain the atoms' separation. First employed in force fields designed for predicting vibrational spectra, they are now used to improve accuracy when modelling mechanical properties and phonon dispersion curves.

Theoretically cross terms should be included for every possible interaction - however this would cause huge problems both in terms of parameterisation and computation time. In practice, if the interactions occurring in the molecule are some distance apart the cross term approximates to zero and can be ignored. For structure prediction, only a few of the cross terms are large enough to be of significance: Maple, Dinur and Hagler (Dinur, 1991) employed quantum mechanics to discover that the most important cross terms are bond-bond, bond-angle, angle-angle, bond-torsion and angle-angle-torsion.

### 2.3.3 Specialised Force Fields

In an ideal world, one energy expression and set of parameters would be transferable between completely unrelated molecules. However, experience has shown that some classes of molecule may be better described by using a force field
that has been derived specifically for them. In fact, selecting a force field which is appropriate for the molecule under investigation is one of the most important steps in obtaining an accurate predicted structure. The following examples show some of the wide range of force fields which have been developed.

2.3.3.1 MM2 and Derivatives

MM2 (Allinger, 1977) and MM3 (Allinger, 1989) are the most famous and widely used force fields for small organic molecules. Both were developed to reproduce experimental structure and heats of formation. MM3 has also been parameterised for vibrational frequencies, heats of sublimation and crystal packing.

Other workers have extended these force fields, enabling their use for a wider range of molecules. Typical examples include the MMPEP parameterisation (Wolfe, 1988), for peptides and the development of an internal searching algorithm which automatically suggests values for parameters missing from the original force field (Schnur, 1991). This system was developed to combat the lack of parameters present for heterocyclic and poly-functionalised molecules.

MM2 and MM3 include bond stretching, angle bending, torsional, inversion, van der Waals' and electrostatic terms - however many terms are significantly different from those previously described. For example, the traditional harmonic bond and angle terms are not used since they are not suitable for describing the distortions found in highly strained organic molecules. Instead, MM2 uses a cubic polynomial for bonds and harmonic and sixth order terms for angles whilst MM3 uses a quartic function for bonds and a full expansion up to sixth order terms for angles.
2.3.3.2 Biological Force Fields - AMBER and CHARMm

Several force fields have been developed specifically to deal with biological molecules (i.e. peptides and proteins). Typical examples include AMBER - Assisted Model Building and Energy Refinement (Weiner, 1981) and CHARMm - Chemistry at Harvard Macromolecular Mechanics (Brooks, 1993). These force fields are much simpler in form than the MM2 and MM3 force fields previously described, bearing greater similarity to the generic energy expression outlined in Equation 2.1 (page 52). For example, the bond stretch and angle bend functions are harmonic, the van der Waals’ potential takes the 12:6 Lennard-Jones format and the cross terms have been omitted. Furthermore, these force fields use a united atom approach, whereby the hydrogen atoms attached to a carbon atom are not defined explicitly - instead the entire CH$_3$, CH$_2$ or CH group is treated as a single entity. These simplifications have been made because biological molecules are usually larger than those studied using MM2 or MM3. Consequently they require more computer time to reach a minimum energy configuration. The simpler terms require fewer parameters, enabling a more rapid solution to be attained.

4.2.3.3 Dreiding Force Fields

The original Dreiding force field was developed by Mayo et al (Mayo, 1990) to deal with a wider range of structures: organics, biological molecules and main group inorganics. Like AMBER and CHARMm, Dreiding force fields are based on the harmonic function for bond stretching and angle bending. Dreiding is robust, allowing reasonable predictions for a wide range of structures. This is because Dreiding’s force constants and geometry parameters are based upon the hybridisation of the atoms involved, not upon specific combinations of atoms. Dreiding is parameterised for hydrogen bonding but does not include cross terms. Atoms are represented both explicitly and as united atoms, allowing the user to decide which representation is most appropriate for the system under investigation.
2.3.3.4 Burchart

The Cerius² modelling package includes the Burchart force field (Burchart, 1992) for modelling zeolite structures: silicas and aluminophosphates. It is parameterised for four atom types, Silicon, Oxygen, Aluminium and Phosphorus, and the parameters are mainly derived from experimental data. Also included are force fields which combine Burchart with Dreiding or the Universal Force Field (UFF). These are used to model the properties of guest molecules - usually organic, within the zeolite framework. Four interactions must be considered:

- **Framework** - Interactions within the silica/aluminophosphate structure.
- **Intra-molecular** - Interactions within each guest molecule.
- **Inter-molecular** - Interactions between guest molecules.
- **Framework-molecule** - Non-bonded interactions between the framework and the guest molecules.

In each case Burchart is used for the framework and Dreiding or Universal provides parameters for the intra- and inter-molecular interactions. Parameters for the framework-molecule interactions are derived from both force fields.

2.3.3.5 Universal Force Field

The Universal Force Field, or UFF (Rappé, 1992) is a general purpose force field parameterised for 132 atom types. The parameters may be defined explicitly for a particular interaction, or calculated from 'generator parameters' defined for each atom type and equations describing how they should be combined. If an interaction is explicitly defined, these values are used in preference to the generated ones.
The Universal Force Field has been used to predict a variety of molecules ranging from main group compounds and organics (Casewit, 1992a, 1992b) through to inorganics (Rappé, 1993).

2.3.4 Energy Minimisation Techniques

The choice of minimisation method is also important in solving an optimisation problem. Whereas force field selection is based upon the type of structure being optimised, minimisation technique is related to how close the starting molecule is to an energetically stable structure and to the degree of accuracy required.

The various optimisation methods available can be subdivided into three categories: non-derivative, first-order derivative and second-order derivative.

2.3.4.1 Non-Derivative Optimisation - Simplex

Computationally, one of the easiest optimisation methods is the Simplex. This technique does not rely upon determining the gradient of the energy function, so complex derivative equations are not required.

A simplex is a geometrical 'shape' with N+1 vertices, where N is the number of variables in the energy expression. For a two-variable function, the simplex is therefore a triangle, and for three variables a tetrahedron. Each vertex corresponds to a set of coordinates for which the energy can be evaluated.

The Simplex Optimisation method allows the Simplex to roam around the energy function in a way similar to the movement of an amoeba, following rules in an attempt to achieve the minimum energy conformer. The first step is to generate the initial simplex. One vertex corresponds to the current configuration of the system - the others are commonly generated by evaluating the potential after adding a
constant increment to each coordinate in turn. The most common move the simplex can make is to reflect away from the highest energy point. If, when this vertex is evaluated, it is found to be lower in energy than all of the original vertices, the reflection will be extended further in the same direction. If an energy valley is reached, these moves will not find a lower energy conformation. In this case, the simplex will shrink away from the highest point. A fourth move is possible if this fails, whereby the simplex will contract around the lowest vertex. Simplex can be expensive computationally - however, it usually finds a better solution and is commonly used as an initial optimisation process when the initial conformation is far from the optimised structure. Once the initial stresses of the system have been reduced, Simplex is replaced with a more efficient method.

**2.3.4.2 First Derivative Optimisation - Steepest Descents**

Steepest Descents uses the first derivative of the energy expression. The first derivative gives the gradient of the expression, indicating in which direction the nearest minimum should lie. A vector is selected across the energy expression and the algorithm moves along this vector towards the minimum energy found along it. When the minimum is found, a new vector is chosen, orthogonal to the original search direction. Once again, this vector is searched and the process repeats until the nearest local minimum is achieved. The method is illustrated in Figure 2.14.
Like Simplex, Steepest Descents is good for relieving the high energy experienced by a molecule in its initial configuration. However when the molecule’s configuration places it in a long, narrow valley many small steps are needed to proceed towards the minimum. This is because each successive step the algorithm makes must be in an orthogonal direction to the last, even though this may be inappropriate for quickly reaching the minimum.

2.3.4.3 First Derivative Optimisation - Conjugate Gradients

In this technique, after the first step which is performed in the same way as Steepest Descents, the direction of the new vector is calculated from the gradient at the current point and the direction of the previous vector. This has the advantage that in long valleys the technique will not oscillate towards the minimum, instead taking a more direct route. The technique is illustrated in Figure 2.15.

*Figure 2.15 - Schematic illustration of Conjugate Gradients algorithm in operation.*

The nature of this algorithm makes it converge towards a minimum extremely rapidly - in the order of N steps where N is the number of degrees of freedom.
2.3.4.4 Second Derivative Optimisation - Newton-Raphson
The simplest method for optimising a structure using second-order derivatives is the Newton-Raphson algorithm. Second-order methods use both the first derivative - providing the gradient of the energy expression and the second derivative which gives information about the curvature of the function. This method is extremely good at quickly finding a minimum conformation - for a purely quadratic function a minimum is always found in one step. However, the method is not without drawbacks as it demands the computationally expensive calculation of the Hessian matrix of second derivatives for each step. To overcome this problem various algorithms have been developed, including:

- Use of the same Hessian for a series of steps.
- Use only first derivatives, and build up the Hessian as minimisation proceeds.
- Movement of only one atom at a time, requiring fewer elements in the matrix to be calculated for each step.

Another drawback of the method is its lack of robustness when the molecule is far from the minimum. For these reasons it is often recommended that one of the other algorithms should be employed to gain an approximate solution before the Newton-Raphson algorithm is applied.

2.3.5 Parameterisation Methods
Once the functional form has been decided, parameters must be generated for the atom types. Four approaches are: (Rappe, 1997)

- **Fitting Experimental Data**: Force field parameters are adjusted to reproduce experimental values obtained from sources such as diffraction and spectroscopic data. This technique is the commonest method - the main problem being the availability and accuracy of the data.
- **Fitting Electronic Structure Data:** Large amounts of data can be generated theoretically using ab-initio quantum mechanics programs when experimental data is unavailable. The problem with this is the poor accuracy of the models generated - in many cases the data generated must be empirically corrected.

- **Rule-Based Parameterisation:** This technique relies upon information built up from the individual atoms taking part in the interaction - i.e. the bond length component from an sp\(^3\) carbon atom is constant regardless of the atom at the other end. This method is used in Dreiding and the Universal Force Field.

- **Simple Assignment:** This method assumes that all force constants have the same value, regardless of atom types - this technique is used when there is only an extremely limited data set available.

Of course, it is unusual to attempt to derive an entirely new force field - molecular modellers are more likely to add new parameters and refine an existing force field. In this case, the usual approach is first to seek parameterised structural elements similar to the new atoms one wishes to add. If these are available the parameters may be directly transferred. If not, parameters are set which reproduce an experimentally derived structure. Only if these parameters appear successful in preliminary studies should more careful parameterisation be attempted.

### 2.4 Application of Molecular Mechanics to Coordination Compounds

#### 2.4.1 Introduction

Whilst molecular mechanics is regarded as indispensable in organic chemistry, inorganic chemists have been more reluctant to adopt the technique. This is not without justification - inorganic chemistry produces real problems which the
original molecular mechanics algorithm is simply unable to deal with, and much effort has been spent attempting to solve problems.

### 2.4.2 Problems With Conventional Force Fields

#### 2.4.2.1 Parameterisation

The problem of parameterisation for coordination compounds is not simply one of the large number of transition metals available - although this is a huge problem. An added complication is the number of oxidation states each element can exist in, and the different geometries they can adopt. Table 2.5 shows a comparison of these variables for carbon and a typical transition metal, iron.

**Table 2.5 - Comparison of structural possibilities for carbon and iron.**

<table>
<thead>
<tr>
<th>Carbon</th>
<th>sp</th>
<th>Linear</th>
<th>Bonds are generally evenly distributed around the central atom.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sp²</td>
<td>Trigonal Planar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sp³</td>
<td>Tetrahedral</td>
<td></td>
</tr>
<tr>
<td>Aromatic</td>
<td></td>
<td>Trigonal Planar</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iron</th>
<th>Fe^{II}</th>
<th>Tetrahedral</th>
<th>Huge structural variation possible - even between compounds with same oxidation state &amp; coordination number.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fe⁰⁰</td>
<td>Trigonal Bipyramidal, Octahedral</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fe⁻¹</td>
<td>Octahedral</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fe⁺⁰</td>
<td>Trigonal Bipyramidal, Square Pyramidal, Octahedral, Dodecahedral</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fe⁺¹</td>
<td>Tetrahedral, Trigonal Bipyramidal, Square Pyramidal, Octahedral, Dodecahedral</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fe⁺²</td>
<td>Trigonal, Tetrahedral, Square Pyramidal, Trigonal Bipyramidal, Octahedral, Dodecahedral</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fe⁺³</td>
<td>Tetrahedral, Octahedral</td>
<td>Distribution of bonds around the central atom may not be even.</td>
</tr>
<tr>
<td></td>
<td>Fe⁺⁴</td>
<td>Tetrahedral</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fe⁺⁵</td>
<td>Tetrahedral</td>
<td></td>
</tr>
</tbody>
</table>

#### 2.4.2.2 Geometrical Difficulties

An equally important problem is the ability of structural features in these compounds to exhibit a range of geometries, each of which must be explicitly defined. For example, if four-coordinate molecules are considered, possible
geometries are tetrahedral, square planar, and 'seesaw'. This is shown in Figure 2.16, where all coordinating atoms are assumed to be of the same type.

\[ \text{Tetrahedral} \quad \text{Square Planar} \quad \text{Seesaw} \]

**Figure 2.16 - The three possible geometries a four-coordinate atom may adopt**

If the equilibrium BÂB bond angles in these structures are considered, it is clear that for tetrahedral molecules there is one value - 109.7°. For the square planar structure there are two possible values, 90° and 180°, and the seesaw structure has three, 90°, 120° and 180°.

A standard molecular mechanics program is unable to distinguish between these different types of bond angle. This is not a problem for non-transition metal chemistry, since the most commonly adopted geometry is tetrahedral. However, if such a program was applied to a square planar or seesaw molecule, it would take the first equilibrium value encountered in the force field. The program would then attempt to drive the molecule into a conformation where all the BÂB bond angles were as close as possible to this value. The problem is not restricted to 4-coordinate systems. Ambiguity in equilibrium bond angle is also found in the 3-coordinate T-shaped, the 5-coordinate trigonal bipyramidal and square pyramidal and the 6-coordinate octahedral geometries (Figure 2.17).
Whilst this problem can be overcome through the use of multiple atom types and equilibrium positions, this requires further parameterisation and careful consideration of the starting structure.

In organic chemistry, the variations from the standard bond angles for a particular atom type are relatively small. Unfortunately, this is not the case for coordination compounds, which are much more likely to exhibit geometries which are considerably different from the standard structures. This is especially the case when large ligands (e.g. triphenylphosphines) are coordinated to the metal centre. The obvious consequence of this is a large variation in the values of the ligand-metal-ligand (L-M-L) bond angles. In these cases, the harmonic energy expression may not be an appropriate representation of the bond angles and a softer potential should be used. One way of coping with this is the addition of higher order terms, as used in the SHAPES force field (Allured, 1991) but this requires both further parameterisation and modifications to the molecular mechanics source code.

A further problem may arise due to the nature of the coordinating ligand. Entities such as cyclopentadienyl ligands are difficult to describe using conventional
molecular mechanics force fields, requiring either modification of the standard sp$^2$ carbon atom type to form a new 4-coordinate atom or the definition of a dummy atom. In either case, further parameterisation is required (Figure 2.18).

![Figure 2.18](image)

**Figure 2.18** - Cyclopentadienyl ligand coordinating to metal atom M through (a) modified sp$^2$ carbon atoms and (b) through a dummy atom, X.

### 2.4.3 Methods Developed

In spite of these difficulties, molecular mechanics is still regarded as a valuable technique for modelling inorganic molecules, and much effort has been spent producing appropriate energy expressions and force fields. Whilst the Dreiding and Universal Force Fields have been parameterised for transition metals, their energy expressions are fairly standard. An alternative approach is to modify the form of the energy expression in order to predict coordination compounds more accurately. Two examples of this type are SHAPES (Allured, 1991) and MM2MX-RIPS (Brecknell, 1985, Ferguson, 1990).

#### 2.4.3.1 The SHAPES Force Field

The SHAPES force field, implemented as an extension of CHARMm, used a modified angle bending function to describe interactions at the metal centre.

Rather than using a harmonic angle bend, Allured et al used the Angular Overlap Model (AOM), representing the interaction as a Fourier expansion. The general
form of the equation is given in Equation 2.5 where \( \phi_0 \) is the equilibrium bond angle and \( K^F \) is the Fourier force constant.

\[
\text{Bond Stretch} = K^F [1 + \cos(n \phi + \psi)]
\]

where \( n = \frac{\pi}{\pi - \phi_0} \) and \( \psi = \pi - n \phi_0 \)  

The periodic nature of the angle bending function enables it to deal with the multiple equilibrium bond angles found in coordination compounds without needing to define new atom types. In order to maintain planarity, a term analogous to the out-of-plane bending term previously described is also included.

This new treatment, initially used for square-planar rhodium complexes is equally applicable to many other geometries, including trigonal-bipyramidal, octahedral and square pyramidal. The approach, however, does have certain drawbacks:

- The expressions are based upon idealised geometries, the Fourier terms require expansion if distorted structures are to be studied;
- Angle bending functions have a minimum at \( 0^\circ \) and, like most force fields, 1-3 interactions are not considered - potentially angles can fold in on themselves.

2.4.3.2 MM2MX-RIPS

Most molecular mechanics studies on molecules containing metal centres have concentrated on at most, hexacoordinate systems which maintain a close resemblance to the standard octahedral geometry, thus avoiding the problem of distorted structures with widely varying natural bond angles.
When higher coordinated molecules are considered, the arrangement of the ligands around the metal centre becomes more complex. These compounds have several natural bond angles at the metal centre - indeed the concept of natural bond angle may be inappropriate for highly coordinated metal centres, as their geometries appear to be determined largely by ligand-ligand (1-3) interactions. This principle is illustrated by the conformational diversity and widely varying bond angles at metal centres found in heptacoordinate structures (Richardson, 1982).

Whilst it is possible to define an equilibrium bond angle for each L-M-L interaction, considerable effort is required in terms of parameterisation. For example, a seven coordinate system would require a total of 21 such angles to be specified. The resultant energy-minimised structure would be directly related to the input geometry and the definition of a standard set of angles would bias the results towards the same geometry in all cases. In any case, there is little consistent structural data available due to the wide variation between experimentally observed compounds, so the definition of a unique, transferable set of parameters for all seven-coordinate metal complexes is probably impossible.

A further problem encountered when traditional molecular mechanics codes such as MM2 are applied to highly coordinated transition metals is the presence of a large number of local minima - the result of any calculation is therefore highly dependent upon the initial input structure. In the past, an attempt to solve this problem has been to perform many calculations on different starting structures in an attempt to find the global minimum. Whilst this strategy can be beneficial, it is time consuming and still cannot guarantee the location of the ‘global’ minimum.

These problems led Ferguson and Raber (Ferguson, 1990) to develop MM2MX-RIPS, a version of MM2 extended to handle metal centres which included a
Random Incremental Pulse Search (RIPS) to automatically search the molecule's potential energy surface for the global minimum.

Allinger’s MM2 program was designed primarily to handle small organic molecules - hence the limits that each structure could contain up to 100 atoms and each atom could have up to four bonds. These limits are too small for metal complexes, so in MM2MX they were raised to 999 atoms and 20 bonds.

In order to avoid defining the various natural bond angles at the metal centre the force constants for these angles are set to zero, thereby giving a zero contribution to the overall energy of the molecule for these interactions. The terms are instead replaced with 1-3 interactions specifically for the atoms bonded to the metal - all other 1-3 interactions are treated using the standard angle bending function.

Once a structure has been minimised, there is no way of knowing whether or not it has reached a local or the global minimum. As has previously been mentioned, heptacoordinate molecules have large numbers of local minima around the global minimum, so it is highly likely that a standard ‘hill-descending’ algorithm will find, and remain in, a local minimum. The Random Incremental Pulse System (RIPS) can be used to overcome this. In the RIPS system, some or all of the atoms may have a random increment added to their coordinates. The resultant structure is then minimised once more, and the process is repeated. A termination number, n, is defined as a parameter to the procedure, and when n cycles of pulse and minimisation have returned the same structure, the procedure ends. Ferguson and Raber found in their studies of lanthanide systems that the search constraints and options needed adjusting to obtain the optimum results for each system studied - however, in general, a termination number of 200 and a maximum value of 0.5Å for the random increment was found to yield the best results.
2.4.3.3 Modified MM2
An alternative approach to the problem of multiple natural bond angles in square planar and octahedral complexes was proposed by Yates & Marsden (1994). Their modifications of MM2 allowed it to treat atoms with up to six connections, and was tested with a variety of octahedral structures. Rather than specifically parameterising for the different natural bond angles found in these complexes, the observed angle in the input structure is compared with the ideal angle (90°), and if the difference is greater than 45°, the ideal angle is doubled. The program was used to predict nickel(II), copper(II), cobalt(III) and manganese(III) complexes of N-(cyclopentyl)- and N-(cycloheptyl)dithiocarbamate. An obvious problem of this technique is the reliance upon a reasonable starting structure - a crude sketch could lead to incorrect allocation of natural bond angles.

2.4.4 Examples
The number of studies of coordination compounds using molecular mechanics is steadily increasing - fuelled by interest in these compounds both in terms of their catalytic ability and their potential structural properties.

2.4.4.1 Zirconocene Ziegler-Natta Catalysts
The polymerisation of propene to form polypropylene is commercially important. The reaction can be catalysed using zirconium metalloocene (zirconocene) Ziegler-Natta catalysts, and it has been shown that the structure of the catalyst influences whether isotactic, syndiotactic or atactic polypropylene is formed.

Castonguay and Rappé (Castonguay, 1992) used ab-initio and molecular mechanics calculations to explain the stereospecificity of existing zirconocenes and to predict a new catalyst which would produce syndiotactic polypropylene. Molecular mechanics was used to investigate the reaction mechanisms and
products. The Dreiding force field was modified to include a tetrahedral zirconium and pseudoatoms for the centroids of the coordinating entities. A variety of zirconocenes were studied to investigate the origin of the polymer’s tacticity. As a result of the molecular mechanics studies, a new syndiotactic catalyst, C₂H₄[cyclopentadienyl-l-(6,7,8,9,10,11,12,13-octahydrofluorenyl)]ZrCl₂ has been proposed.

2.4.4.2 Conducting Complexes

For a number of years there has been great interest in the solid state chemistry and particularly the magnetic behaviour of chromium(II) (Halepota, 1989a, 1989b; Jubb, 1989a). In 1989, an unusual chromium complex - [Cr(NCS)₂(thiourea)₂], (Jubb, 1989b), was synthesised. This complex crystallised in a stacked square-planar arrangement, with a Cr-Cr separation of 3.97Å (Figure 2.19).

![Figure 2.19 - Stacked square-planar [Cr(NCS)₂(thiourea)₂] complex.](image)

Other square-planar complexes and in particular macrocyclic complexes which exhibit this type of structure have proved to be good conductors - for example hemiporphyrinines (Dirk, 1984) and phthalocyanines (Petersen, 1977). In these compounds, the metal centre can form a bond perpendicular to the coordination plane and partial oxidation with a halogen (usually iodine) afforded conduction.
Of course in the chromium complex, an interplanar separation of almost 4Å is too long to be significant, but if the structure of the ligands could be modified this separation could be reduced. In the past, ligand choice has been left to chemical experience. Effective modelling methods would be of great benefit to this problem, guiding the lengthy process of synthesis, crystallisation and characterisation by x-ray crystallography.

Work in this research group by Cole (1992) resulted in the production of a new molecular mechanics program for coordination compounds, MOLMECH, which could deal with the problem of varying natural bond angles at the metal centre. The program was also equally applicable to standard small organic molecules.

X-ray structures for two chromium complexes, diethylene-triammonium tetrachlorochromate(II) and propane-1,3-diammonium tetrachlorochromate(II) are available in the Cambridge Structural Database (CSD Refcodes BAHVOV and BAHVIP, respectively). Parameters were developed from the diethylene-triammonium tetrachlorochromate(II) complex. Initial parameters for the metal-ligand vibrations were obtained from IR data, and all parameters were then modified by hand until the structure could be reproduced. These parameters were then used to successfully predict the structure of the propane-1,3-diammonium tetrachlorochromate(II) complex. Cole concludes that structure predictions can be made if the structure(s) used for parameterisation are similar to the target molecule - however these parameters cannot be transferred to other, unrelated systems.

2.4.4.3 Applications of MM2MX-RIPS

Further work in this research group (Wright, 1994), used the MM2MX-RIPS system on a variety of coordination complexes. Force field parameters were reported for the first time for uranium(IV), palladium(IV) and tungsten.
compounds. Previously reported parameters for cobalt(II) were adapted for the study of Schiff base complexes.

These studies show how the MM2MX system of 1,3 non-bonded interactions replacing ligand-metal-ligand angles can be used in some cases to control the geometry at the metal centre. It was discovered that where steric forces are strong, the use of 1,3-interactions can lead to a chemically unacceptable distortion of the geometry. However, where steric forces are less dominant, the 1,3-interactions are successful, and the definition of multiple natural bond angles is not necessary.

Each set of parameters, however, has been generated using a limited set of experimental data, and their transferability to other systems is therefore doubtful. A further problem, of course, is the need for these parameters to be hand refined for each molecular system - a laborious, time consuming task.
2.5 Automatic Parameterisation of Molecular Mechanics Force Fields

2.5.1 Introduction
Since there is never likely to be one force field which is universally applicable to all problems in chemistry, the task of generating new parameters for new or existing force fields will always be present. Traditional methods of parameterisation - the hand derivation of constants from either spectroscopic or structural data or by attempting to match a new atom type with data already present, followed by refinement of the parameters until the test structure(s) are accurately modelled can be both tedious and time consuming.

2.5.2 Examples
Since force field parameterisation has proved to be so difficult, a variety of guidelines have been produced for the production of force field parameters and in particular, force constants. Some of these techniques have been taken one stage further and computer programs have been developed which attempt to automate the parameterisation process.

Hopfinger and Pearlstein (Hopfinger, 1984) proposed a technique whereby force field constants are derived by ‘subtracting’ non-bonded molecular mechanics energies from corresponding molecular orbital energies using a compound containing the structural feature to be parameterised as a model. Once raw data has been generated for a selection of conformations (e.g. different values for the bond length if a bond stretch constant is being determined) a curve fitting algorithm is used to generate the force field parameter. Generally, seven pairs of data (energy vs., for example, bond length) are recommended for the curve fitting - comprising of three sets above and below the equilibrium value. The general strategy for
selecting model compounds is to replace all atoms connected to the structural unit with hydrogens - the parameters are therefore based upon information from only one compound. It is suggested that these procedures should be used to expand existing parameter sets and not to derive entirely new force fields.

White (1989) described a program for force constant optimisation and suggested a strategy for computing initial guesses for force constants prior to optimisation. When all force constants for bond stretching from published force fields were plotted against $1/(\text{equilibrium bond length})^2$ for all bonds which do not contain hydrogen, the result is an acceptable straight line. A similar correlation could be made for bonds containing hydrogen. This suggested that simple relationships (Equation 2.6 for non-hydrogen atoms $i$ and $j$ and Equation 2.7 for atom $i$ and hydrogen) could be used to produce values for bond stretching force constants from values for the equilibrium bond length.

\[
\frac{1}{2} k_b(ij) = \left[ \frac{c_1}{b_0^2(ij)} \right] + \left[ \frac{c_2}{(b_0^2(ij) - 1)} \right] + \left[ \frac{c_3}{b_0(ij)} \right] \tag{2.6}
\]

\[
\frac{1}{2} k_b(iH) = \frac{c_4}{b_0^3(iH)} \tag{2.7}
\]

In both equations, $b_0$ represents the equilibrium bond length and $c_1$, $c_2$, $c_3$ and $c_4$ are constants whose values are obtained by adjustment to give the optimum fit between observed and calculated structures containing a wide range of bond types.

The situation for angle bending was more complex, and the equations produced were entirely empirical. Non-bonded parameters were generated from values of $A(ii)$ and $B(ii)$ stored for each atom type $i$ in the force field. Values for $A(ii)$ and
B(ij) were calculated by combining these constants using a geometric mean. If hydrogen was involved in the interaction, the mean was multiplied by a constant value - this enabled the molecular mechanics program to calculate both structures and energies accurately. Torsion angle constants were assigned simply upon the nature of the central bond. Since out-of-plane bending force constants were rarely used, they were stored explicitly.

This force field could then be optimised using an iterative procedure taking approximately 60 cycles to converge. The program, running on a MC 68020/6881 system (equivalent to a VAX-11/780) was run overnight (completing 3-4 iterations) and at weekends (completing 16-20 iterations). Several trial runs were required per functional group to obtain the optimum weighting scheme. Although the system is time consuming (values for saturated and unsaturated hydrocarbons took 3-4 months to produce, including code development) a corresponding force field by manual trial and error methods was developed over a period of three years. Use of a transputer based system enabled the algorithm to run 3-4 times faster, and the procedure is ideal for parallelisation.

An alternative approach is to attempt to match new structural units with information already present in the force field. For example, an initial guess at the torsion angle constant for C=NOC could be provided by the value for C=NOH. These parameters must then be optimised against experimental data. The accuracy of the initial guess, however, is obviously dependent upon how much experience the user has with the force field.

Schnur, Grieshaber and Bowen, (Schnur, 1991) developed an algorithm which would search the MM2 force field for suitable analogues to undefined structural features - generating a set of parameters for any molecule whose atoms were
contained in the force field, even if the specific functional groups were not defined. The program contained a series of rules, developed to describe which substitutions were most preferable. The system worked by retrieving parameter error messages from a standard MM2 output file, searching for appropriate parameters and interrogating the user with respect to how appropriate the new parameters were. If the user felt that they were inappropriate or had experimental data, the suggested parameters could be overwritten. Finally, the program created a standard MM2 parameter deck for the molecule of interest. The authors stress that the internal searching algorithm should only be used to provide crude parameters when no other data was available. Further optimisation of these parameters would be vital before meaningful calculations could be performed.
2.6 A Genetic Algorithm for Determining Force Field Parameters

2.6.1 Program Design

2.6.1.1 Introduction
A computer program has been developed which uses a genetic algorithm to evolve new molecular mechanics force field parameters. The force fields are derived solely from crystal structure data. The parameters, therefore, are specifically aimed at the prediction of molecular structure in the crystalline state. The derived force field is used within an in-house conjugate gradients molecular mechanics program, MOLMECH (Cole, 1992).

Initial development of the program was carried out on a 50MHz 80486DX based IBM Compatible PC. The molecular mechanics minimiser and molecular fitting algorithms were translated from VAX FORTRAN-77 to standard FORTRAN-77 using the MicroWay NDP FORTRAN compiler. The genetic algorithm was developed in ANSI standard C, compiled with the MicroWay NDP C/C++ Compiler. The compilers produced language independent object files, allowing the routines to be linked together. Both compilers were for the 80486 processor, generating 32-bit code which could address up to 4Gbytes of memory in protected mode and use hard disk space as virtual memory if required.

2.6.1.2 Overview
The program operates by minimising compounds with a series of randomly generated force fields. After each minimisation, the resulting molecule is compared to that found in the compound’s crystal structure, giving a value for the root mean square (RMS) fit. These values are summed giving a score evaluating the fitness of each force field. As the score is related to how closely the crystal
structure is modelled, a high score denotes a poor match and therefore low fitness for that force field. The next generation is then created and the cycle repeats.

2.6.1.3 Outline of the Algorithm

A flow diagram of the genetic algorithm code is shown in Figures 2.20a and 2.20b (page 92). The program’s design is explained after each section of the flowchart.

![Flow diagram of the first half of the genetic algorithm.](image-url)

*Figure 2.20a - Flow diagram of the first half of the genetic algorithm.*
Reading the Structural Information
In order to run the program, a Cambridge Database FDAT file is required. This file contains the compound(s) on which the force field parameters are to be based. The path and name of this file is required as a parameter of the program. The file is opened and each molecule is decoded - the required information is stored in an array of structures.

Restarting a Run
Since initial run times were extremely long, the facility to restart an interrupted run was added. If a run is restarted, a file (Restart.fil), written at the end of each generation, is opened. This file contains the status of the program at the end of the cycle before it was interrupted. Data stored includes the number of force fields, total number of cycles and the number of the last completed cycle. Once this information has been retrieved, the random number generator can be reseeded and the program can then recommence. The program does not need to create new structure files or force fields at this stage, as they have already been created by the previously interrupted run. If the user wishes to extend a run which has been completed, the restart file can be edited for the new total number of cycles.

Starting a New Run
If a new run is being started, the first task is to seed the random number generator. If the random number generator is not seeded or is seeded with a constant, the program will always generate an identical sequence of random numbers. For this reason, the seed is generated from the system clock as the number of seconds that have elapsed since midnight. It was considered unlikely that this result would be duplicated for a particular set of data. Once the random number generator has been seeded, the following must be performed before the first generation of the GA commences:
1. Initialising Structural Data Files

The following routines are called for each compound in the FDAT file:

- **OrthData()** - orthogonalises the data stored in a ComplexData structure if necessary, overwriting the fractional coordinates.

- **hybridise()** - The Cambridge Structural Database makes no attempt to store information regarding the hybridisation of atoms in structures. This information must therefore be inferred from the connectivity information and/or the user. Each atom from the FDAT file is checked for element type and number of attached atoms, and the user is prompted to select what hybridisation state the atom should be assigned if necessary. The user interaction is vital because some CSD entries have atoms removed due to errors. If such a compound is one of only a small number of entries containing a particular structural feature it should still be included, despite the error. An automatic system for identifying the hybridisation of this type of atom is not possible, since information about deleted atoms is stored in a comments field in the FDAT file.

- **writecosmic()** - creates data files in the correct format for the MOLMECH minimiser. MOLMECH requires that the data must be orthogonalised before being written in this format. The second parameter to the routine is a Boolean variable denoting whether the data should be perturbed slightly prior to minimisation by adding a small random increment to the x, y and z coordinates for each atom. The GA requires that a perturbed file with file extension '.RND' (for minimising) and a standard file with extension '.XR' (for molecular fitting) are created. The filename in each case is derived from the structure’s REFCODE in the CSD.
2. Force Field Creation

The next section of the program generates the starting population of the genetic algorithm. The user is first asked the population size.

*genforcefield()* - creates the initial force field parameters. Currently, this routine is entirely hard-coded, requiring modification for each set of compounds. Initial parameters are generated randomly within limits. For example, the force field for alkanes allows a range of approximately 10 degrees either side of the expected equilibrium bond angles. MOLMECH's force field requires separate files for each set of interactions - bonds, angles, torsions and non-bonded. The files are linked together through their file extensions which correspond to the number of members in the population. (e.g. angle.0, bond.0 etc. ..... angle.20, bond.20 etc.). This allows up to 1000 force fields in the population (file extensions 0 to 999), when the program is run under MS-DOS with its 8.3 file format. Under other operating systems (e.g. UNIX), this restriction is removed due to the greater flexibility in file naming conventions.

3. Restart File

Finally, the initial information (number of force fields, number of cycles and number of compounds) is written to the restart file. The generation counter is set to zero in preparation for the first generation to begin.
Figure 2.20b - Flow diagram of the second half of the genetic algorithm.
A GA Generation

At the beginning of each cycle, the restart file is updated with the current program status. The evaluation section of the program is handled by the generation() routine.

1. The Evaluation Section

The evaluation routine cycles through each force field in the population, treating each molecule in the following way: the first step of the evaluation is an attempt to minimise the structure.

- **minim()** is the Conjugate Gradients molecular mechanics minimiser originally developed as MOLMECH by Cole (1992). The program was originally implemented on a MicroVax II computer, and written in the VAX/VMS dialect of FORTRAN. MOLMECH's force field is shown in Equation 2.1 (page 52).

The first stage in the GA development required a version of this program to be ported to the PC. Despite the fact that the MicroWay NDP compiler had an option to compile VMS FORTRAN, some modifications were required before the program would reproduce calculations performed on the MicroVax. The main differences involved file handling (directory structures and path names) and invalid variable names caused by differences between the VMS and DOS operating systems. Once this had been completed, the minimiser was tested by minimising small alkane molecules (methane and ethane). The PC version of the program reproduced the structures reported by Cole.

The program was then further modified to allow it to be called as a subroutine. Interaction with the user was removed by using command line parameters for the data filename and force field number as input and variables for returning whether
or not the minimiser had converged and the minimised structure’s energy as output. All screen output of the program (apart from error messages - e.g. missing parameters) was also removed. The output of the minimiser was stored in the same format as the input data, in a file named ‘OUT.XR’.

If the minimiser fails to converge, the force field parameters are obviously inappropriate. In this case, a large value (usually 15) is added to the force field’s score. If the minimiser succeeds, a molecular fitting routine is called. This program compares the output of the minimiser to the compound’s crystal structure, returning the RMS difference between them and this value is then added to the score for the force field.

- `euler_()`, the molecular fitting routine, was developed in standard FORTRAN 77 on a UNIX based system (Howlin, 1992). This routine was ported to the PC environment without any changes to the source code. The program was then modified to accept data in the same format as the minimiser. The calling procedure was changed, allowing it to be called as a subroutine, with parameters of the filenames of the two structures to be compared and a variable to return the value of the RMS fit.

After all the molecules have been minimised with a particular force field, the result is the sum of the RMS differences. A large value indicates major differences between the minimised compound(s) and their crystal structure(s). Fitness-is-evaluation is therefore inappropriate for ranking the force fields, so a fitness scaling factor was required. One of the fitness-scaling equations used is given in Equation 2.8.
Chapter 2: A GA for Generating Force Field Parameters

Fitness = \frac{10 \times \text{Number of Compounds}^2}{\left( \sum_{\text{All Compounds}} \text{RMS Difference} \right)^2} \quad 2.8

It was found that squaring the sum of the RMS differences increased the separation between good and poor force fields in a close race, improving the rate of selection of the fitter force fields. Other fitness-scaling factors raised the sum of the RMS difference to the fourth power and/or included a contribution from the sum of the energies of the minimised structures in the denominator.

The final stage in the evaluation section is the creation of the selection roulette wheel. An array containing the sum of all fitnesses up to and including the force field corresponding to each array element (i.e. array element 2 = fitness of force field 0 + fitness of force field 1 + fitness of force field 2) was created.

Once the evaluation section is completed, the results are written to the output file. If the last generation has completed, the program terminates, otherwise the force fields must be allowed to reproduce.

2. The Reproduction Section

In the reproduction section, the force fields which contribute to the next generation are selected. The new solutions are created and the old ones deleted. Although various strategies may be adopted, this genetic algorithm replaces all but the best member of its population each generation.

- selectff() - Selects the force fields which will enter the breeding pool for the next generation. Fitter force fields should be picked several times, whereas the worst are unlikely to be selected.
- **SelectCross()** - selects pairs of force fields at random from the breeding pool and calls `cross()` to perform crossover. Once a pair has been selected they are removed from the breeding pool, ensuring they won’t be selected again.

- **cross()** - sets the mutation rate and performs crossover on the selected pairs of force fields. Each set of force field files (bonds, angles, torsions and non-bonds) is crossed using separate routines - `Angleff()`, `Bondff()`, `Torsionff()` and `Nbondff()`.

The routines used for crossing each type of parameter are analogous to each other. A typical example is the angle crossover routine, `Angleff()`:

- **Angleff()** - The genetic algorithm treats the angle bending force field file as a single chromosome. Crossover consists of generating random number(s) between zero and the number of lines in the file and exchanging data between the two force fields. The lines are read from the parent force fields using `GetAngleString()`. Routines were developed for both one and two point crossover.

- **GetAngleString()** - This routine decodes each fixed format line of the angle bending force field. The method of mutation is real-number creep. If any of the mutation tests are passed, a random integer between -2 and 2 or a random float between -0.1 and 0.1 is generated and added onto the existing value. The line from the force field is then reconstructed and written to the new force field.

The final steps in the reproduction section (and the GA generation) are to copy the best force field unchanged into the next generation (elitism), and to remove the old files and rename the new force fields to allow them to be accessed from MOLMECH. These functions are carried out by `CopyBest()` and `cleanup()` respectively. The cycle then repeats until the program has completed the required number of generations.
2.6.1.4 Summary
The time taken to develop the prototype genetic algorithm was approximately ten months. Once the program was completed, initial tests were carried out using the crystal structures of straight chain alkanes. A variety of attempts were made to optimise the program, through changes to the algorithm itself and the use of alternate hardware. Finally, the program was tested on a tetrahedral nickel(II) complex. Unless otherwise stated, in all cases one point crossover is used. These investigations are discussed in the following sections.

2.6.2 Initial Investigations: Straight-Chain Alkanes
Once the program had been developed, it was necessary to decide upon a series of compounds which could be used to test the principles of the genetic algorithm. The simplest compounds commonly minimised using molecular mechanics are straight-chain alkanes, which do not require many parameters in order to be modelled. A search of the Cambridge Database was made using the QUEST shown in Figure 2.21. An explanation of each line is provided.

```
SAVE 3
T1 *CLASS 05              Save results to an FDAT file
T2 *BTEST -314 -315 -317  Miscellaneous aliphatic compounds
T3 *ELEM 6A               Triple, double or aromatic bonds
T4 *ELEM 7A               Group 6 elements
T5 *ELEM D                Group 7 elements
T6 *COORD 0               Deuterium

```

Figure 2.21 - A CSD QUEST which returns aliphatic compounds which have coordinate data but do not contain multiple bonds, elements from groups 6 or 7 or deuterium.
The QUEST returned eight structures - all of which are straight-chain alkanes. The journal file for these compounds is shown in Figure 2.22.

Figure 2.22 - CSD journal file produced by above QUEST.

Inspection of the FDAT file revealed that the entry for n-Octacosane (Refcode OCTCOS) had two hydrogen atoms removed due to suspected coordinate errors, leaving seven potential structures for use with the GA.

The first test runs with the genetic algorithm were carried out using the two simplest compounds, ethane and pentane as test molecules. This allowed an initial investigation of suitable mutation parameters and population sizes, although, due to the random nature of the program, this process was somewhat empirical.
Chapter 2: A GA for Generating Force Field Parameters

The GA was then run using six of the above compounds, allowing the resulting force field to be tested on the seventh. An FDAT file was created containing the alkanes listed in Table 2.6.

Table 2.6 - Alkanes used for GA force field optimisation.

<table>
<thead>
<tr>
<th>Alkane</th>
<th>Formula</th>
<th>CSD Refcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-Ethane</td>
<td>C₂H₆</td>
<td>ETHANE01</td>
</tr>
<tr>
<td>n-Pentane</td>
<td>C₅H₁₂</td>
<td>PENTAN01</td>
</tr>
<tr>
<td>n-Hexane</td>
<td>C₆H₁₄</td>
<td>HEXANE</td>
</tr>
<tr>
<td>n-Heptane</td>
<td>C₇H₁₆</td>
<td>HEPTAN01</td>
</tr>
<tr>
<td>n-Octane</td>
<td>C₈H₁₈</td>
<td>OCTANE10</td>
</tr>
<tr>
<td>n-Hexatricontane</td>
<td>C₃₆H₇₄</td>
<td>HXTACM</td>
</tr>
</tbody>
</table>

Two runs were performed on the 50MHz 80486 PC. The first run used the standard fitness scaling shown in Equation 2.8. The second run included a contribution from the energy in the score, as shown in Equation 2.9.

\[
\text{Fitness} = \frac{10 \times \text{Number of Compounds}^2}{\left( \sum_{\text{All Compounds}} \left[ \text{RMS Difference} + \left( \frac{\text{energy} + 150}{200} \right) \right]^2 \right)} \tag{2.9}
\]

Both algorithms used 51 force fields for 250 generations, with mutation rates of 0.01 for integers and 0.02 for floats.

The resulting graph of best and average score and RMS fits against generation for the first and second runs are shown in Figures 2.23 and 2.24 respectively.
Figure 2.23 - Graph showing variation in best & average scores and \( \Sigma \text{RMS fits} \) against generation for GA with no contribution from energy in score.

Figure 2.24 - Graph showing variation in best & average scores and \( \Sigma \text{RMS fits} \) against generation for GA with contribution from energy in score.
Both graphs show the shape typical of a genetic algorithm run. The initial stages of the graphs show a shallow gradient - this corresponds to the initial exchanges of information taking part in the reproduction section. There are no solutions in the initial population which perform dramatically better than the others. Eventually, one or more solutions combine schemata which do predict the alkanes' structures significantly better than their competitors. These reproduce at a far greater rate than the rest of the population, driving the average fitness upwards. It should however be noted that these individuals do not dominate the whole population, as shown by the differences between the best and average fitnesses. This genetic variability is maintained both by the roulette-wheel selection process and by the mutation operator, which can harm 'good' solutions in the same way that it can improve them. Both runs have converged after approximately 100 generations, although minor improvements do improve the best score marginally until the end of the program.

The results of the two runs are summarised in Table 2.7. It is immediately obvious that the times taken to complete the genetic algorithm vary widely. This is not surprising, given the randomly generated nature of both the starting structures and the force fields.

**Table 2.7 - Results of two 6-alkane genetic algorithm runs.**

<table>
<thead>
<tr>
<th></th>
<th>Run 1 (No Energy)</th>
<th>Run 2 (Energy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time of Run</td>
<td>150 hours 36 minutes</td>
<td>126 hours 58 minutes</td>
</tr>
<tr>
<td></td>
<td>(6.275 days)</td>
<td>(5.290 days)</td>
</tr>
<tr>
<td>Σ RMS Fit (Start of Run)</td>
<td>4.997</td>
<td>5.083</td>
</tr>
<tr>
<td>Σ RMS Fit (End of Run)</td>
<td>2.759</td>
<td>3.225</td>
</tr>
<tr>
<td>Improvement</td>
<td>2.238</td>
<td>1.858</td>
</tr>
<tr>
<td>Σ RMS Energy (Start of Run)</td>
<td>413.619 kcal/mol</td>
<td>106.691 kcal/mol</td>
</tr>
<tr>
<td>Σ RMS Energy (End of Run)</td>
<td>348.621 kcal/mol</td>
<td>7.527 kcal/mol</td>
</tr>
<tr>
<td>Improvement</td>
<td>64.998 kcal/mol</td>
<td>99.164 kcal/mol</td>
</tr>
</tbody>
</table>
For the purposes of this work, the accuracy of the compound’s geometry was deemed more important than its energy, as the generation of an accurate molecular structure is vital if an attempt is going to be made to extend the study to the prediction of the crystal unit. Indeed, it was seen that including a contribution from the energy later in the run could be counter-productive at times, by driving the RMS contribution upwards. On the whole, after some fluctuations in the first few generations, improvements in the energy contribution go hand in hand with improvements in the RMS fit. This is shown for both runs in Figure 2.25 (No Energy) and Figure 2.26 (Energy).

![Graph showing variation in RMS fits and energy against generation for GA without contribution from energy in score.](image)

**Figure 2.25** - Graph showing variation in RMS fits and energy against generation for GA without contribution from energy in score.
Figure 2.26 - Graph showing variation in RMS fits and energy against generation for GA with contribution from energy in score.

A tentative conclusion from these initial runs was that a "better" force field could be produced by simply monitoring the RMS fits between the minimised structures and not including a contribution from the molecules' energies. Of course, a contribution from the energies is implicitly included in the RMS fit data since a molecule's energy is directly related to its structure through the energy expression used in molecular mechanics calculations.

The parameters generated by the run without an energy contribution to the score and those originally provided by Cole (1992) are shown in Table 2.8.
Table 2.8 - Force field parameters generated by GA and provided by Cole for use with the MOLMECH minimiser.

<table>
<thead>
<tr>
<th>Bond</th>
<th>( b_0 ) (Å)</th>
<th>( k_b ) (kcal/mol/Å²)</th>
<th>( b_0 ) (Å)</th>
<th>( k_b ) (kcal/mol/Å²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-C</td>
<td>1.561</td>
<td>320.4</td>
<td>1.526</td>
<td>310.0</td>
</tr>
<tr>
<td>C-H</td>
<td>1.053</td>
<td>323.8</td>
<td>1.090</td>
<td>331.0</td>
</tr>
<tr>
<td>Angle</td>
<td>( \theta_0 ) (°)</td>
<td>( k_\theta ) (kcal/mol/rad²)</td>
<td>( \theta_0 ) (°)</td>
<td>( k_\theta ) (kcal/mol/rad²)</td>
</tr>
<tr>
<td>C-C-C</td>
<td>104.5</td>
<td>39</td>
<td>109.5</td>
<td>40</td>
</tr>
<tr>
<td>C-C-H</td>
<td>109.7</td>
<td>30</td>
<td>109.5</td>
<td>35</td>
</tr>
<tr>
<td>H-C-H</td>
<td>113.5</td>
<td>44</td>
<td>109.5</td>
<td>35</td>
</tr>
<tr>
<td>Non Bonded</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>2.06</td>
<td>0.06</td>
<td>1.80</td>
<td>0.06</td>
</tr>
<tr>
<td>H</td>
<td>1.30</td>
<td>0.08</td>
<td>1.00</td>
<td>0.02</td>
</tr>
<tr>
<td>Torsions</td>
<td>Periodicity</td>
<td>( V_n ) (kcal/mol)</td>
<td>Periodicity</td>
<td>( V_n ) (kcal/mol)</td>
</tr>
<tr>
<td>X-C-C-X</td>
<td>3</td>
<td>1.518</td>
<td>3</td>
<td>1.412</td>
</tr>
</tbody>
</table>

The force field derived by the genetic algorithm was tested using MOLMECH with two alkanes: n-pentane and n-octadecane. The results were compared with the crystal structures from the CSD to obtain an RMS fit. The process was then repeated for the original parameters provided by Cole. These parameters had previously been shown to produce results which closely match those from the MM2 package (Cole, 1992). The results are shown in Table 2.9.

Table 2.9 - Comparison of performance of GA derived force field against original MOLMECH parameters.

<table>
<thead>
<tr>
<th></th>
<th>n-Pentane</th>
<th>n-Octadecane</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GA</td>
<td>Cole</td>
</tr>
<tr>
<td>RMS from Crystal Structure</td>
<td>0.102</td>
<td>1.591</td>
</tr>
<tr>
<td>Total Energy (kcal/mole)</td>
<td>5.491</td>
<td>0.176</td>
</tr>
</tbody>
</table>

The RMS fit for both compounds is considerably closer with the GA derived force field, indicating that the technique is appropriate for deriving force fields. One source of error in the original MOLMECH parameters when applied to crystalline
alkanes is the tendency for the molecule to fold in upon itself, forming a ‘tangled’ conformation. This feature is not seen with the GA’s parameters, which reproduce the linear nature of the alkanes. This may be due in part to the higher torsion angle force constant derived by the GA.

2.6.3 Optimising the GA

One feature of the GA which is immediately apparent is the length of time it takes to perform a run. At the time this work began, the 50MHz 80486DX was Intel’s second most powerful PC processor (only being surpassed by the clock-doubled 66MHz 80486DX2). The fastest run took over five days to complete. Even if they had been terminated after 100 generations when the runs first began to converge, the fastest would still have taken over two days to complete (assuming that the time taken per generation was constant). Optimisation of the GA in terms of hardware or software was therefore of prime importance.

2.6.3.2 Hardware

Although this genetic algorithm, like all others, is ideally suited for parallelisation, this type of hardware was unavailable at the time. Attention therefore turned to running the algorithm on faster and/or more powerful systems in an attempt to improve performance. The alternative hardware available for evaluation was:

- An IBM compatible PC with a 66MHz 80486DX2 processor.
- A MicroWay NumberSmasher-860 board in the above PC.
- A Silicon Graphics Crimson.
- A four processor Silicon Graphics PowerChallengeL.
66MHz 80486DX2
The DX2 processors produced by Intel are clock doubled processors. This means that the external speed of the processor is half the quoted internal clock speed. For this reason, the improvement in run time indicated by the processor speed is only likely to be approached if the program does not need to send or retrieve information along the system bus. This situation is rarely the case.

Porting the genetic algorithm to this machine was simply a case of transferring the executable and modifying the system start-up files (config.sys and autoexec.bat) to load MicroWay’s DOS extender rather than the software provided by Microsoft.

MicroWay NumberSmasher-860
MicroWay provided a board with a 40MHz i860 processor and 32Mbytes of RAM for a four-week evaluation. Since the NumberSmasher board used i860 versions of the NDP compilers used in the initial development of the program, porting the code was relatively straight forward. The code was compiled with and without compiler optimisations, and attempts were made to vectorise the FORTRAN routines. Whilst the molecular fitting routine was successfully vectorised, vectorisation of various subroutines in the molecular mechanics program caused the genetic algorithm to crash. This was later identified as being due to problems with the source code, derived from its origins on a VMS system. Due to time constraints (the need to return the hardware), it was not possible to attempt running the i860 and the 80486 processors in parallel.

Silicon Graphics Crimson & Supercomputer
Towards the end of the project, it was possible to attempt implementing the genetic algorithm on two Silicon Graphics UNIX machines - initially a Crimson workstation and then a PowerChallenge mainframe.
Porting the genetic algorithm ANSI C code and the standard FORTRAN-77 molecular fitting routine to the Silicon Graphics was once again straightforward. However, problems were encountered with the minimiser. VMS FORTRAN allows uninitialised variables in calculations and floating point numbers to be used as array indices. The NDP FORTRAN compilers used on the PC and the NumberSmasher allow these features as long as the code is compiled with the -VMS switch. The Silicon Graphics compiler, however, is more rigid in its implementation of FORTRAN. The process of successfully compiling the minimiser on the Crimson such that it ran and reproduced results calculated on the PC was eventually completed after considerable time and effort, allowing the Crimson’s performance to be compared to the other systems. This code was then ported and compiled on the supercomputer without problems.

Results

A simple genetic algorithm was run for 100 generations with 2 compounds and 51 force fields on each machine. The code was optimised on the PCs and optimised and vectorised on the NumberSmasher. Given the variability in run time naturally occurring due to the random nature of the program, each run was repeated five times. The results are summarised in Table 2.10.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Average Run Time (Sec.)</th>
<th>Improvement over 50MHz 486</th>
</tr>
</thead>
<tbody>
<tr>
<td>50MHz 80486DX</td>
<td>56023.2</td>
<td>0%</td>
</tr>
<tr>
<td>50MHz 80486DX (Optimised)</td>
<td>54945.6</td>
<td>2%</td>
</tr>
<tr>
<td>66MHz 80486DX2</td>
<td>47198.4</td>
<td>19%</td>
</tr>
<tr>
<td>66MHz 80486DX2 (Optimised)</td>
<td>46468.6</td>
<td>21%</td>
</tr>
<tr>
<td>NumberSmasher</td>
<td>41207.2</td>
<td>36%</td>
</tr>
<tr>
<td>NumberSmasher (Optimised)</td>
<td>34733.8</td>
<td>61%</td>
</tr>
<tr>
<td>NumberSmasher (Vectorised)</td>
<td>27173.2</td>
<td>106%</td>
</tr>
<tr>
<td>Crimson</td>
<td>8153.0</td>
<td>587%</td>
</tr>
<tr>
<td>PowerChallenge</td>
<td>3579.4</td>
<td>1465%</td>
</tr>
</tbody>
</table>

Table 2.10 - Time to run genetic algorithm on different platforms.
As expected, the performance improves with increasing power (and cost) of hardware, culminating with the PowerChallenge and the Crimson running almost fifteen and six times faster than the 50MHz 486 PC respectively. It should also be noted that these machines are multitasking, so they were not able to dedicate all of their CPU time to the genetic algorithm. As previously indicated, the performance of the 66MHz 486 is not as high as the maximum possible - the genetic algorithm performs extensive disk access for reading and writing data and force field files, limiting the performance gains.

The optimisation switch provided with the NDP compilers appears to be considerably more efficient on the i860 processor. Notable performance gains are only found, however, when the FORTRAN code is vectorised, taking advantage of the pipelining capabilities of the i860 processor. It can be envisaged that the performance would be even better if the minimisation routine could also be vectorised.

2.6.3.3 Software
An analysis of the times spent in each routine showed that, not surprisingly, most time was spent in the evaluation section of the program, and in particular, in the minimiser. Optimisation of the minimisation routine would therefore give the greatest improvements in overall run time. Two strategies were attempted:

1. The conjugate gradients minimiser iterates several times before full convergence is reached. However, the first iteration usually reaches a structure which is close to the eventual minimum, so the routine could be terminated after one cycle for the first 50% of the genetic algorithm. In the following results, this is referred to as Run B.
2. The convergence criterion for the minimiser is set at $10^{-3}$ kcal/mole. The criterion could be set larger for the earlier stages of the program. A sliding scale of $0.01/(\text{generation number})$ was set. In the following results, this is referred to as Run C.

These strategies were compared to a standard (optimised) program (Run A) for 2 alkanes (ethane and pentane) and 2 force fields, optimising for 10 generations. To ensure that identical conditions pertained to each run, the same random number generator seed was used. A variety of seeds were used to reproduce the results, and runs were performed on both the 50MHz 486 and the NumberSmasher-860. The results of the runs are shown in Table 2.11 and Figure 2.27.

**Table 2.11 - Results of software optimisation attempts.**

<table>
<thead>
<tr>
<th>Seed</th>
<th>Run A Time (s)</th>
<th>Run B Time (s)</th>
<th>Run C Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>214.5</td>
<td>190</td>
<td>203.5</td>
</tr>
<tr>
<td>3</td>
<td>208.5</td>
<td>197</td>
<td>210.5</td>
</tr>
<tr>
<td>4</td>
<td>254</td>
<td>227.5</td>
<td>242.5</td>
</tr>
<tr>
<td>5</td>
<td>171</td>
<td>153</td>
<td>164</td>
</tr>
<tr>
<td>8</td>
<td>185.5</td>
<td>167.5</td>
<td>178.5</td>
</tr>
<tr>
<td>9</td>
<td>202</td>
<td>186</td>
<td>197</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Seed</th>
<th>Fitness</th>
<th>Fitness</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.975</td>
<td>3.975</td>
<td>3.975</td>
</tr>
<tr>
<td>3</td>
<td>1.765</td>
<td>1.765</td>
<td>1.765</td>
</tr>
<tr>
<td>4</td>
<td>6.531</td>
<td>6.531</td>
<td>6.531</td>
</tr>
<tr>
<td>5</td>
<td>5.099</td>
<td>5.099</td>
<td>5.099</td>
</tr>
<tr>
<td>8</td>
<td>3.584</td>
<td>3.584</td>
<td>3.584</td>
</tr>
<tr>
<td>9</td>
<td>6.408</td>
<td>6.408</td>
<td>6.408</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Seed</th>
<th>Run A Time (s)</th>
<th>Run B Time (s)</th>
<th>Run C Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>96</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>3</td>
<td>104</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>4</td>
<td>115</td>
<td>107.5</td>
<td>107</td>
</tr>
<tr>
<td>5</td>
<td>132</td>
<td>127.5</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>112</td>
<td>108</td>
<td>108.5</td>
</tr>
<tr>
<td>9</td>
<td>123</td>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Seed</th>
<th>Fitness</th>
<th>Fitness</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5.651</td>
<td>5.651</td>
<td>5.651</td>
</tr>
<tr>
<td>3</td>
<td>6.355</td>
<td>6.355</td>
<td>6.355</td>
</tr>
<tr>
<td>4</td>
<td>4.873</td>
<td>4.873</td>
<td>4.873</td>
</tr>
<tr>
<td>5</td>
<td>5.929</td>
<td>5.929</td>
<td>5.929</td>
</tr>
<tr>
<td>8</td>
<td>2.720</td>
<td>2.720</td>
<td>2.720</td>
</tr>
<tr>
<td>9</td>
<td>4.204</td>
<td>4.204</td>
<td>4.204</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Seed</th>
<th>Run A Time (s)</th>
<th>Run B Time (s)</th>
<th>Run C Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>96</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>3</td>
<td>104</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>4</td>
<td>115</td>
<td>107.5</td>
<td>107</td>
</tr>
<tr>
<td>5</td>
<td>132</td>
<td>127.5</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>112</td>
<td>108</td>
<td>108.5</td>
</tr>
<tr>
<td>9</td>
<td>123</td>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Seed</th>
<th>Fitness</th>
<th>Fitness</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5.651</td>
<td>5.651</td>
<td>5.651</td>
</tr>
<tr>
<td>3</td>
<td>6.355</td>
<td>6.355</td>
<td>6.355</td>
</tr>
<tr>
<td>4</td>
<td>4.873</td>
<td>4.873</td>
<td>4.873</td>
</tr>
<tr>
<td>5</td>
<td>5.929</td>
<td>5.929</td>
<td>5.929</td>
</tr>
<tr>
<td>8</td>
<td>2.720</td>
<td>2.720</td>
<td>2.720</td>
</tr>
<tr>
<td>9</td>
<td>4.204</td>
<td>4.204</td>
<td>4.204</td>
</tr>
</tbody>
</table>
Figure 2.27 - Graph of times taken for each run with different seeds on the 80486 and i860

The data indicates that fitness is not affected during these optimisations, although this may be due to the small population size. The use of a single iteration of the minimiser (Run B) does improve run times on both the i860 (average 6% improvement) and the 80486 (average 9% improvement). The use of a sliding scale for convergence (Run C) shows considerable variation between hardware platforms. On the PC, the times for Run C generally fall between Runs A and B (an average improvement of 3%), whereas on the NumberSmasher-860, the method is equally as good as Run B (average improvement 6%). This implies that the time spent by the 80486 processor performing the division to obtain the convergence criterion partially negates the beneficial effect of the optimisation. The i860 can handle divisions more efficiently than the 80486, allowing the benefit of the optimisation to become more apparent.
2.6.3.4 Conclusions

The optimisation of even simple force fields using a genetic algorithm can be a lengthy process. The program’s performance has been shown to be best improved by the use of more powerful hardware. However, enhancements are also possible through the use of software optimisation methods in the evaluation section.

The replacement of the minimiser program with a more robust alternative may result in enhancements in both areas - vectorisation of this routine on the NumberSmasher board should result in much faster operation.

The enhancement which the GA is inherently suited to is its implementation on a parallel machine, where significant improvements would undoubtedly be achieved.

2.6.4 Investigation of Optimum Mutation Rate

One of the most fundamental parameters of a genetic algorithm is the mutation rate - i.e. the percentage of numbers in the force fields whose values are altered during reproduction. The use of the optimum mutation rate for a particular problem makes the GA converge more quickly to its “best” score. However, if the mutation rate for the algorithm is set too low, the GA will be less likely to investigate many solutions which are not already present in the initial population. Conversely, if the mutation rate is set too high, schema which are performing well are likely to be corrupted, damaging the overall effectiveness of the algorithm. One can therefore expect a trendline overlaid on a graph of convergence generation against mutation rate to display a minimum value at the optimum mutation rate.
In order to investigate the optimum mutation rate, several series of GA runs were performed. Mutation rates of 0.0125, 0.025, 0.0375, 0.05, 0.0625, 0.075, 0.0825 and 0.1 were investigated for one alkane (octane), two alkanes (octane and ethane) and three alkanes (octane, ethane and pentane). In each case, five runs were performed, giving a total of 120 runs.

Each run was performed on a population of 51 force fields. The random number generator seed for each run was obtained from the system clock when the run was started. Since the aim of the investigation was to determine when each run had converged, it was imperative that an excess number of generations was specified to allow convergence to take place. From an empirical examination of test runs, a value of 750 cycles was chosen.

2.6.4.1 Determination of Convergence

Convergence is defined for GAs as the generation where the population consists primarily of similar individuals. For the purposes of this algorithm, this definition is unsuitable due to the computational overhead of performing $n(n+1)/2$ file comparisons per generation. An alternative approach is to propose the generation at which the algorithm has converged empirically from the shape of the best score against generation graph.

However, a more rigorous measure must be implemented if one wishes to compare the results of a series of runs using different parameters or crossover methods. This measure must also bear no direct dependence upon the score if it is to be used to compare runs performed on different sets of test molecules.

The easiest method for determining convergence in an elitist genetic algorithm is to monitor the improvement in the best solution generated. The algorithm may be
Chapter 2: A GA for Generating Force Field Parameters

Deemed to have converged when there has been no improvement for a user-specified number of cycles. This is a particularly robust method because once the convergence criterion has been established it requires no input from the user - each run of the GA must eventually converge.

One problem with this approach, however, is the arbitrary determination of the value of the convergence criterion. If the problem under investigation is simple, the output graphs typically follow the standard GA shape and a difference of 10 cycles will make little difference to the overall result. However, the problem under investigation in this case is extremely complex. Test results showed a wide variation in the shapes of the graphs produced: whilst many showed the expected shape, some settled for a "best" solution which performed only marginally better than the best force field in the initial population for long periods of time before displaying the characteristic jump to a reasonable solution (Figures 2.28a-b).

![Graph of Best Score vs. Generation for a run showing characteristic GA shape. Test compound: Octane, Mutation rate: 0.0125.](image-url)
Figure 2.28b - Graph of Best Score vs. Generation for a run showing a late improvement in best score. Test compound: Octane, Mutation rate: 0.0125.

Other results improved in series of small jumps over a relatively long period of time (Figure 2.28c, below).

Figure 2.28c - Graph of Best Score vs. Generation for a run showing a gradual improvement in best score. Test compound: Octane, Mutation rate: 0.0125.
As indicated, each of these graphs were generated from GA runs performed using identical parameters - the differences being caused entirely by the different seeds used for the random number generator. This wide variation showed that a simple convergence criterion would be inappropriate for this algorithm.

By calculating the average of the best score found in the current and all previous generations, a smoothed function is derived. Empirically, these functions show more similarities between different runs (Figures 2.29a-c), indicating that they may be more appropriate for comparing convergence.

Figure 2.29a - Graph of Average of Best Scores vs. Generation for a run showing characteristic GA shape. Test compound: Octane, Mutation rate: 0.0125.
Figure 2.29b - Graph of Average of Best Scores vs. Generation for a run showing a late improvement in best score. Test compound: Octane, Mutation rate: 0.0125.

Figure 2.29c - Graph of Average of Best Scores vs. Generation for a run showing a gradual improvement in best score. Test compound: Octane, Mutation rate: 0.0125.

This data still causes some difficulties for analysis. As can be seen, there is now no constant value which may be monitored, and the value of the Average of the
Best Scores varies widely, even for the same compound test data. This variation is obviously more marked when runs using different numbers of compounds are compared.

These problems may be overcome by calculating the percentage difference between an average score and its preceding value. A generation of convergence may then be proposed by determining the last generation in which the percentage improvement was above a particular threshold. For the purposes of this investigation, six threshold values were considered, 1.25%, 1%, 0.75%, 0.5%, 0.25% and 0.1%.

Graphs illustrating this data for the three example runs are shown in Figures 2.30a-c.

![Graph of % Improvement in Average of Best Scores vs. Generation for a run showing characteristic GA shape. Test compound: Octane, Mutation rate: 0.0125. The Y axis shows only the values less than or equal to 2%](image)
Figure 2.30b - Graph of % Improvement in Average of Best Scores vs. Generation for a run showing a late improvement in best score. Test compound: Octane, Mutation rate: 0.0125. The Y axis shows only the values less than or equal to 2%.

Figure 2.30c - Graph of % Improvement in Average of Best Scores vs. Generation for a run showing a gradual improvement in best score. Test compound: Octane, Mutation rate: 0.0125. The Y axis shows only the values less than or equal to 2%.
2.6.4.2 Results
The GA program was run on the test data sets using the parameter values outlined. The resulting data were then analysed in the manner previously described, with the “convergence generation” and the best score at “convergence” stored for each run. The data generated for each set of compounds was initially analysed separately and then combined.

Octane
The results for the 40 runs performed on octane are tabulated in Table 2.12 and selected results (at the 1%, 0.5% and 0.25% thresholds) are displayed graphically in Figures 2.31a-2.33b.
Table 2.12 - Results of GA runs on Octane analysed to various “convergence” criteria. Cycle refers to the generation at which convergence was achieved and score to the best score in the population at convergence.

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>1.25%</th>
<th>1%</th>
<th>0.75%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle</td>
<td>Score</td>
<td>Cycle</td>
</tr>
<tr>
<td>0.0125</td>
<td>63</td>
<td>45.597</td>
<td>70</td>
</tr>
<tr>
<td>0.0125</td>
<td>78</td>
<td>17.6543</td>
<td>84</td>
</tr>
<tr>
<td>0.0125</td>
<td>9</td>
<td>365.9854</td>
<td>10</td>
</tr>
<tr>
<td>0.0125</td>
<td>78</td>
<td>44.82616</td>
<td>84</td>
</tr>
<tr>
<td>0.0125</td>
<td>117</td>
<td>124.1211</td>
<td>126</td>
</tr>
<tr>
<td>0.025</td>
<td>37</td>
<td>17.63848</td>
<td>42</td>
</tr>
<tr>
<td>0.025</td>
<td>107</td>
<td>43.88589</td>
<td>115</td>
</tr>
<tr>
<td>0.025</td>
<td>167</td>
<td>349.79698</td>
<td>172</td>
</tr>
<tr>
<td>0.025</td>
<td>50</td>
<td>39.29355</td>
<td>59</td>
</tr>
<tr>
<td>0.025</td>
<td>28</td>
<td>21.48227</td>
<td>31</td>
</tr>
<tr>
<td>0.0375</td>
<td>31</td>
<td>15.02397</td>
<td>34</td>
</tr>
<tr>
<td>0.0375</td>
<td>37</td>
<td>22.81326</td>
<td>40</td>
</tr>
<tr>
<td>0.0375</td>
<td>29</td>
<td>318.108</td>
<td>32</td>
</tr>
<tr>
<td>0.0375</td>
<td>80</td>
<td>87.75476</td>
<td>86</td>
</tr>
<tr>
<td>0.0375</td>
<td>52</td>
<td>25.42765</td>
<td>56</td>
</tr>
<tr>
<td>0.05</td>
<td>25</td>
<td>16.51493</td>
<td>28</td>
</tr>
<tr>
<td>0.05</td>
<td>31</td>
<td>19.72365</td>
<td>34</td>
</tr>
<tr>
<td>0.05</td>
<td>63</td>
<td>37.41499</td>
<td>72</td>
</tr>
<tr>
<td>0.05</td>
<td>53</td>
<td>33.11635</td>
<td>58</td>
</tr>
<tr>
<td>0.05</td>
<td>85</td>
<td>33.88391</td>
<td>92</td>
</tr>
<tr>
<td>0.0625</td>
<td>92</td>
<td>96.66784</td>
<td>99</td>
</tr>
<tr>
<td>0.0625</td>
<td>35</td>
<td>517.6462</td>
<td>39</td>
</tr>
<tr>
<td>0.0625</td>
<td>6</td>
<td>7.53687</td>
<td>92</td>
</tr>
<tr>
<td>0.0625</td>
<td>54</td>
<td>28.90528</td>
<td>63</td>
</tr>
<tr>
<td>0.0625</td>
<td>30</td>
<td>186.9411</td>
<td>33</td>
</tr>
<tr>
<td>0.075</td>
<td>39</td>
<td>27.92688</td>
<td>42</td>
</tr>
<tr>
<td>0.075</td>
<td>54</td>
<td>64.44001</td>
<td>59</td>
</tr>
<tr>
<td>0.075</td>
<td>39</td>
<td>31.11288</td>
<td>42</td>
</tr>
<tr>
<td>0.075</td>
<td>46</td>
<td>61.63274</td>
<td>50</td>
</tr>
<tr>
<td>0.075</td>
<td>36</td>
<td>51.73548</td>
<td>40</td>
</tr>
<tr>
<td>0.0875</td>
<td>108</td>
<td>158.3144</td>
<td>116</td>
</tr>
<tr>
<td>0.0875</td>
<td>35</td>
<td>37.10111</td>
<td>38</td>
</tr>
<tr>
<td>0.0875</td>
<td>88</td>
<td>484.9277</td>
<td>125</td>
</tr>
<tr>
<td>0.0875</td>
<td>33</td>
<td>36.74566</td>
<td>36</td>
</tr>
<tr>
<td>0.0875</td>
<td>37</td>
<td>30.09816</td>
<td>41</td>
</tr>
<tr>
<td>0.1</td>
<td>70</td>
<td>57.35607</td>
<td>78</td>
</tr>
<tr>
<td>0.1</td>
<td>53</td>
<td>74.72286</td>
<td>58</td>
</tr>
<tr>
<td>0.1</td>
<td>105</td>
<td>72.21236</td>
<td>113</td>
</tr>
<tr>
<td>0.1</td>
<td>45</td>
<td>39.06033</td>
<td>117</td>
</tr>
<tr>
<td>0.1</td>
<td>79</td>
<td>106.3648</td>
<td>86</td>
</tr>
</tbody>
</table>
### Table 2.12 continued

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>0.5%</th>
<th>0.25%</th>
<th>0.1%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle</td>
<td>Score</td>
<td>Cycle</td>
</tr>
<tr>
<td>0.0125</td>
<td>488</td>
<td>167.3035</td>
<td>583</td>
</tr>
<tr>
<td>0.0125</td>
<td>109</td>
<td>17.75869</td>
<td>145</td>
</tr>
<tr>
<td>0.0125</td>
<td>15</td>
<td>366.5686</td>
<td>20</td>
</tr>
<tr>
<td>0.0125</td>
<td>111</td>
<td>45.62272</td>
<td>147</td>
</tr>
<tr>
<td>0.0125</td>
<td>160</td>
<td>127.7662</td>
<td>247</td>
</tr>
<tr>
<td>0.025</td>
<td>57</td>
<td>18.10954</td>
<td>309</td>
</tr>
<tr>
<td>0.025</td>
<td>145</td>
<td>43.62424</td>
<td>191</td>
</tr>
<tr>
<td>0.025</td>
<td>212</td>
<td>350.0305</td>
<td>270</td>
</tr>
<tr>
<td>0.025</td>
<td>80</td>
<td>43.37985</td>
<td>110</td>
</tr>
<tr>
<td>0.025</td>
<td>45</td>
<td>22.32772</td>
<td>149</td>
</tr>
<tr>
<td>0.0375</td>
<td>246</td>
<td>49.8164</td>
<td>314</td>
</tr>
<tr>
<td>0.0375</td>
<td>96</td>
<td>35.07828</td>
<td>128</td>
</tr>
<tr>
<td>0.0375</td>
<td>55</td>
<td>351.0615</td>
<td>123</td>
</tr>
<tr>
<td>0.0375</td>
<td>114</td>
<td>89.14901</td>
<td>151</td>
</tr>
<tr>
<td>0.0375</td>
<td>121</td>
<td>39.81459</td>
<td>160</td>
</tr>
<tr>
<td>0.05</td>
<td>229</td>
<td>54.04308</td>
<td>290</td>
</tr>
<tr>
<td>0.05</td>
<td>167</td>
<td>36.82572</td>
<td>217</td>
</tr>
<tr>
<td>0.05</td>
<td>94</td>
<td>40.03474</td>
<td>126</td>
</tr>
<tr>
<td>0.05</td>
<td>80</td>
<td>34.26884</td>
<td>125</td>
</tr>
<tr>
<td>0.05</td>
<td>118</td>
<td>33.89727</td>
<td>161</td>
</tr>
<tr>
<td>0.0625</td>
<td>126</td>
<td>96.98044</td>
<td>178</td>
</tr>
<tr>
<td>0.0625</td>
<td>53</td>
<td>328.8071</td>
<td>76</td>
</tr>
<tr>
<td>0.0625</td>
<td>134</td>
<td>24.29839</td>
<td>222</td>
</tr>
<tr>
<td>0.0625</td>
<td>84</td>
<td>31.62604</td>
<td>120</td>
</tr>
<tr>
<td>0.0625</td>
<td>46</td>
<td>189.4798</td>
<td>77</td>
</tr>
<tr>
<td>0.075</td>
<td>57</td>
<td>28.06913</td>
<td>83</td>
</tr>
<tr>
<td>0.075</td>
<td>79</td>
<td>65.1221</td>
<td>107</td>
</tr>
<tr>
<td>0.075</td>
<td>57</td>
<td>31.11288</td>
<td>89</td>
</tr>
<tr>
<td>0.075</td>
<td>71</td>
<td>65.39143</td>
<td>448</td>
</tr>
<tr>
<td>0.075</td>
<td>61</td>
<td>55.66378</td>
<td>129</td>
</tr>
<tr>
<td>0.0875</td>
<td>149</td>
<td>160.9843</td>
<td>196</td>
</tr>
<tr>
<td>0.0875</td>
<td>54</td>
<td>38.16696</td>
<td>236</td>
</tr>
<tr>
<td>0.0875</td>
<td>172</td>
<td>813.05</td>
<td>236</td>
</tr>
<tr>
<td>0.0875</td>
<td>60</td>
<td>41.72667</td>
<td>82</td>
</tr>
<tr>
<td>0.0875</td>
<td>55</td>
<td>30.09816</td>
<td>75</td>
</tr>
<tr>
<td>0.1</td>
<td>104</td>
<td>60.70556</td>
<td>141</td>
</tr>
<tr>
<td>0.1</td>
<td>77</td>
<td>74.97305</td>
<td>104</td>
</tr>
<tr>
<td>0.1</td>
<td>361</td>
<td>312.9614</td>
<td>476</td>
</tr>
<tr>
<td>0.1</td>
<td>156</td>
<td>85.39702</td>
<td>220</td>
</tr>
<tr>
<td>0.1</td>
<td>111</td>
<td>107.9239</td>
<td>165</td>
</tr>
</tbody>
</table>
Chapter 2: A GA for Generating Force Field Parameters

\[ y = 15726x^2 - 1725.7x + 101.71 \]
\[ R^2 = 0.1051 \]

**Figure 2.31a** - Graph of 1% Convergence Generation against Mutation Rate for GA runs on octane with quadratic trendline.

**Figure 2.31b** - Graph of Best Score at 1% Convergence against Mutation Rate for GA runs on octane with linear trendline.
Chapter 2: A GA for Generating Force Field Parameters

\[ y = 33120x^2 - 4093.4x + 218.52 \]
\[ R^2 = 0.0869 \]

Figure 2.32a - Graph of 0.5% Convergence Generation against Mutation Rate for GA runs on octane with quadratic trendline.

Figure 2.32b - Graph of Best Score at 0.5% Convergence against Mutation Rate for GA runs on octane with linear trendline.
Chapter 2: A GA for Generating Force Field Parameters 124

Figure 2.33a - Graph of 0.25% Convergence Generation against Mutation Rate for GA runs on octane with quadratic trendline.

\[ y = 33783x^2 - 4101.1x + 281.74 \]
\[ R^2 = 0.051 \]

Figure 2.33b - Graph of Best Score at 0.25% Convergence against Mutation Rate for GA runs on octane with linear trendline.
For each convergence threshold except 1.25% the linear trendline superimposed on the graph of best score at “convergence” vs. mutation rate increased as mutation rate increased. This general trend is expected as the increased mutation rate allows areas of the search space which crossover alone cannot reach to be investigated. The opposite behaviour at the 1.25% threshold may be because this value is too high to determine convergence in the GA - i.e. the population is too varied.

In each case, the quadratic trendline superimposed on the graph of “convergence” generation against mutation rate shows a minimum value - suggesting a mutation rate at which convergence is most quickly attained. These results are shown in Table 2.13.

**Table 2.13** - Table of quadratic trendline equations, optimum mutation rates and optimum mutation rate (to 3 decimal places) for each convergence threshold.

*Optimum mutation rate is calculated as the minimum position in the trendline.*

<table>
<thead>
<tr>
<th>Convergence Threshold</th>
<th>Equation of Quadratic Trendline</th>
<th>R-squared Value</th>
<th>Optimum Mutation Rate (3 d. p.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.25%</td>
<td>$y = 6435.1 \times - 792.93 \times + 115.24$</td>
<td>0.0016</td>
<td>0.062</td>
</tr>
<tr>
<td>1%</td>
<td>$y = 15726 \times - 1725.7 \times + 101.71$</td>
<td>0.1051</td>
<td>0.055</td>
</tr>
<tr>
<td>0.75%</td>
<td>$y = 9211.4 \times - 1027.0 \times + 103.29$</td>
<td>0.0246</td>
<td>0.056</td>
</tr>
<tr>
<td>0.5%</td>
<td>$y = 33120 \times - 4093.4 \times + 218.52$</td>
<td>0.0869</td>
<td>0.062</td>
</tr>
<tr>
<td>0.25%</td>
<td>$y = 33783 \times - 4101.1 \times + 281.74$</td>
<td>0.0510</td>
<td>0.061</td>
</tr>
<tr>
<td>0.1%</td>
<td>$y = 55398 \times - 6332.7 \times + 427.56$</td>
<td>0.0706</td>
<td>0.057</td>
</tr>
</tbody>
</table>

Although the $R^2$ values for the trendlines are low - probably due to the random nature of the GA, each data set shows a minimum at around the same mutation rate, indicating that the optimum mutation rate for this particular problem is approximately 0.059.
Octane and Ethane

The results for the 40 runs performed on octane and ethane are tabulated in Table 2.14 and selected results (at the 1%, 0.5% and 0.25% thresholds) are displayed graphically in Figures 2.34a-2.36b.

**Table 2.14 - Results of GA runs on Octane and Ethane analysed as previous.**

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>1.25%</th>
<th>1%</th>
<th>0.75%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle</td>
<td>Score</td>
<td>Cycle</td>
</tr>
<tr>
<td>0.0125</td>
<td>0</td>
<td>13.6174</td>
<td>86</td>
</tr>
<tr>
<td>0.0125</td>
<td>87</td>
<td>25.84472</td>
<td>94</td>
</tr>
<tr>
<td>0.0125</td>
<td>0</td>
<td>9.458744</td>
<td>0</td>
</tr>
<tr>
<td>0.0125</td>
<td>9</td>
<td>6.861386</td>
<td>17</td>
</tr>
<tr>
<td>0.0125</td>
<td>0</td>
<td>4.378421</td>
<td>122</td>
</tr>
<tr>
<td>0.025</td>
<td>10</td>
<td>21.53363</td>
<td>11</td>
</tr>
<tr>
<td>0.025</td>
<td>114</td>
<td>35.06065</td>
<td>123</td>
</tr>
<tr>
<td>0.025</td>
<td>59</td>
<td>86.92198</td>
<td>65</td>
</tr>
<tr>
<td>0.025</td>
<td>80</td>
<td>42.09806</td>
<td>87</td>
</tr>
<tr>
<td>0.025</td>
<td>98</td>
<td>16.08695</td>
<td>105</td>
</tr>
<tr>
<td>0.0375</td>
<td>114</td>
<td>1513.935</td>
<td>122</td>
</tr>
<tr>
<td>0.0375</td>
<td>27</td>
<td>31.35807</td>
<td>33</td>
</tr>
<tr>
<td>0.0375</td>
<td>84</td>
<td>53.05451</td>
<td>90</td>
</tr>
<tr>
<td>0.0375</td>
<td>8</td>
<td>10.46731</td>
<td>9</td>
</tr>
<tr>
<td>0.0375</td>
<td>86</td>
<td>94.57199</td>
<td>93</td>
</tr>
<tr>
<td>0.05</td>
<td>74</td>
<td>398.9815</td>
<td>80</td>
</tr>
<tr>
<td>0.05</td>
<td>56</td>
<td>523.273</td>
<td>61</td>
</tr>
<tr>
<td>0.05</td>
<td>24</td>
<td>47.41308</td>
<td>27</td>
</tr>
<tr>
<td>0.05</td>
<td>2</td>
<td>9.575574</td>
<td>58</td>
</tr>
<tr>
<td>0.05</td>
<td>51</td>
<td>100.1428</td>
<td>56</td>
</tr>
<tr>
<td>0.0625</td>
<td>60</td>
<td>63.14319</td>
<td>65</td>
</tr>
<tr>
<td>0.0625</td>
<td>63</td>
<td>96.31368</td>
<td>69</td>
</tr>
<tr>
<td>0.0625</td>
<td>23</td>
<td>33.04464</td>
<td>37</td>
</tr>
<tr>
<td>0.0625</td>
<td>5</td>
<td>49.21763</td>
<td>61</td>
</tr>
<tr>
<td>0.0625</td>
<td>8</td>
<td>11.80994</td>
<td>9</td>
</tr>
<tr>
<td>0.075</td>
<td>15</td>
<td>29.07472</td>
<td>17</td>
</tr>
<tr>
<td>0.075</td>
<td>2</td>
<td>24.39991</td>
<td>2</td>
</tr>
<tr>
<td>0.075</td>
<td>3</td>
<td>20.33392</td>
<td>3</td>
</tr>
<tr>
<td>0.075</td>
<td>29</td>
<td>72.84521</td>
<td>34</td>
</tr>
<tr>
<td>0.075</td>
<td>89</td>
<td>202.4161</td>
<td>95</td>
</tr>
<tr>
<td>0.0875</td>
<td>0</td>
<td>23.99651</td>
<td>0</td>
</tr>
<tr>
<td>0.0875</td>
<td>27</td>
<td>47.26593</td>
<td>33</td>
</tr>
<tr>
<td>0.0875</td>
<td>73</td>
<td>196.5431</td>
<td>80</td>
</tr>
<tr>
<td>0.0875</td>
<td>34</td>
<td>141.9046</td>
<td>38</td>
</tr>
<tr>
<td>0.0875</td>
<td>98</td>
<td>102.3906</td>
<td>100</td>
</tr>
<tr>
<td>0.1</td>
<td>2</td>
<td>13.29019</td>
<td>35</td>
</tr>
<tr>
<td>0.1</td>
<td>2</td>
<td>34.05116</td>
<td>2</td>
</tr>
<tr>
<td>0.1</td>
<td>44</td>
<td>31.16746</td>
<td>55</td>
</tr>
<tr>
<td>0.1</td>
<td>109</td>
<td>989.0216</td>
<td>117</td>
</tr>
<tr>
<td>0.1</td>
<td>104</td>
<td>23.94881</td>
<td>112</td>
</tr>
</tbody>
</table>
Table 2.14 - Continued

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>0.0%</th>
<th>0.25%</th>
<th>0.1%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle</td>
<td>Score</td>
<td>Cycle</td>
</tr>
<tr>
<td>0.0125</td>
<td>119</td>
<td>33.28688</td>
<td>159</td>
</tr>
<tr>
<td>0.0125</td>
<td>126</td>
<td>27.42426</td>
<td>206</td>
</tr>
<tr>
<td>0.0125</td>
<td>178</td>
<td>36.13182</td>
<td>236</td>
</tr>
<tr>
<td>0.0125</td>
<td>214</td>
<td>25.81741</td>
<td>287</td>
</tr>
<tr>
<td>0.0125</td>
<td>156</td>
<td>17.32926</td>
<td>350</td>
</tr>
<tr>
<td>0.025</td>
<td>15</td>
<td>21.53363</td>
<td>98</td>
</tr>
<tr>
<td>0.025</td>
<td>156</td>
<td>35.78792</td>
<td>234</td>
</tr>
<tr>
<td>0.025</td>
<td>90</td>
<td>92.79823</td>
<td>121</td>
</tr>
<tr>
<td>0.025</td>
<td>117</td>
<td>44.31739</td>
<td>188</td>
</tr>
<tr>
<td>0.025</td>
<td>141</td>
<td>17.35883</td>
<td>185</td>
</tr>
<tr>
<td>0.0375</td>
<td>159</td>
<td>159.114</td>
<td>215</td>
</tr>
<tr>
<td>0.0375</td>
<td>60</td>
<td>39.68408</td>
<td>118</td>
</tr>
<tr>
<td>0.0375</td>
<td>117</td>
<td>53.15725</td>
<td>155</td>
</tr>
<tr>
<td>0.0375</td>
<td>50</td>
<td>15.0974</td>
<td>129</td>
</tr>
<tr>
<td>0.0375</td>
<td>123</td>
<td>98.26824</td>
<td>162</td>
</tr>
<tr>
<td>0.05</td>
<td>104</td>
<td>399.1041</td>
<td>139</td>
</tr>
<tr>
<td>0.05</td>
<td>80</td>
<td>523.273</td>
<td>194</td>
</tr>
<tr>
<td>0.05</td>
<td>38</td>
<td>48.18684</td>
<td>56</td>
</tr>
<tr>
<td>0.05</td>
<td>77</td>
<td>17.38126</td>
<td>103</td>
</tr>
<tr>
<td>0.05</td>
<td>77</td>
<td>102.4178</td>
<td>108</td>
</tr>
<tr>
<td>0.0625</td>
<td>85</td>
<td>63.51319</td>
<td>115</td>
</tr>
<tr>
<td>0.0625</td>
<td>91</td>
<td>97.77677</td>
<td>122</td>
</tr>
<tr>
<td>0.0625</td>
<td>50</td>
<td>41.50575</td>
<td>155</td>
</tr>
<tr>
<td>0.0625</td>
<td>86</td>
<td>100.2478</td>
<td>122</td>
</tr>
<tr>
<td>0.0625</td>
<td>12</td>
<td>11.80994</td>
<td>17</td>
</tr>
<tr>
<td>0.075</td>
<td>25</td>
<td>29.53932</td>
<td>39</td>
</tr>
<tr>
<td>0.075</td>
<td>3</td>
<td>24.39991</td>
<td>88</td>
</tr>
<tr>
<td>0.075</td>
<td>8</td>
<td>20.87496</td>
<td>22</td>
</tr>
<tr>
<td>0.075</td>
<td>49</td>
<td>78.38957</td>
<td>67</td>
</tr>
<tr>
<td>0.075</td>
<td>122</td>
<td>202.4161</td>
<td>161</td>
</tr>
<tr>
<td>0.0875</td>
<td>6</td>
<td>25.16195</td>
<td>42</td>
</tr>
<tr>
<td>0.0875</td>
<td>196</td>
<td>135.782</td>
<td>251</td>
</tr>
<tr>
<td>0.0875</td>
<td>103</td>
<td>199.0291</td>
<td>138</td>
</tr>
<tr>
<td>0.0875</td>
<td>51</td>
<td>141.9046</td>
<td>70</td>
</tr>
<tr>
<td>0.0875</td>
<td>135</td>
<td>102.3906</td>
<td>177</td>
</tr>
<tr>
<td>0.1</td>
<td>53</td>
<td>21.74164</td>
<td>88</td>
</tr>
<tr>
<td>0.1</td>
<td>97</td>
<td>62.81697</td>
<td>129</td>
</tr>
<tr>
<td>0.1</td>
<td>73</td>
<td>34.69383</td>
<td>99</td>
</tr>
<tr>
<td>0.1</td>
<td>162</td>
<td>114.351</td>
<td>248</td>
</tr>
<tr>
<td>0.1</td>
<td>153</td>
<td>26.68856</td>
<td>214</td>
</tr>
</tbody>
</table>
Figure 2.34a - Graph of 1% Convergence Generation against Mutation Rate for GA runs on octane and ethane with quadratic trendline.

\[ y = 7718.1x^2 - 1118.8x + 89.754 \]
\[ R^2 = 0.0537 \]

Figure 2.34b - Graph of Best Score at 1% Convergence against Mutation Rate for GA runs on octane and ethane with linear trendline.
Chapter 2: A GA for Generating Force Field Parameters

\[ y = 35650x^2 - 4559.7x + 208.37 \]

\[ R^2 = 0.3085 \]

**Figure 2.35a** - Graph of 0.5% Convergence Generation against Mutation Rate for GA runs on octane and ethane with quadratic trendline.

**Figure 2.35b** - Graph of Best Score at 0.5% Convergence against Mutation Rate for GA runs on octane and ethane with linear trendline.
Figure 2.36a - Graph of 0.25% Convergence Generation against Mutation Rate for GA runs on octane and ethane with quadratic trendline.

Figure 2.36b - Graph of Best Score at 0.25% Convergence against Mutation Rate for GA runs on octane and ethane with linear trendline.
For the two-alkane runs, at each convergence threshold the linear trendline superimposed on the graph of best score at “convergence” vs. mutation rate increased as mutation rate increased, as expected.

In each case except for the 1.25% threshold, the quadratic trendline superimposed on the graph of “convergence” generation against mutation rate shows a minimum value - suggesting a mutation rate at which convergence is most quickly attained. This is further evidence that the 1.25% threshold is too high to determine convergence in the GA’s population. These results are shown in Table 2.15.

Table 2.15 - Table of quadratic trendline equations, optimum mutation rates and optimum mutation rate (to 3 decimal places) for each convergence threshold. Optimum mutation rate is calculated as the minimum position in the trendline.

<table>
<thead>
<tr>
<th>Convergence Threshold</th>
<th>Equation of Quadratic Trendline</th>
<th>R-squared Value</th>
<th>Optimum Mutation Rate (3 d.p.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.25%</td>
<td>$y = -30832 x^2 + 4208.1 x + 17.411$</td>
<td>0.0117</td>
<td>N/A</td>
</tr>
<tr>
<td>1%</td>
<td>$y = 7718.1 x^2 - 1118.8 x + 89.754$</td>
<td>0.0537</td>
<td>0.072</td>
</tr>
<tr>
<td>0.75%</td>
<td>$y = 16274 x^2 - 2315.8 x + 134.82$</td>
<td>0.1726</td>
<td>0.071</td>
</tr>
<tr>
<td>0.5%</td>
<td>$y = 35650 x^2 - 4559.7 x + 208.37$</td>
<td>0.3085</td>
<td>0.064</td>
</tr>
<tr>
<td>0.25%</td>
<td>$y = 49470 x^2 - 6562.6 x + 317.21$</td>
<td>0.3902</td>
<td>0.066</td>
</tr>
<tr>
<td>0.1%</td>
<td>$y = 71528x^2 - 9630.9x + 535.99$</td>
<td>0.2526</td>
<td>0.067</td>
</tr>
</tbody>
</table>

The R² values for the trendlines are again low. The lowest value corresponds to the 1.25% threshold which does not show a minimum value, which is further evidence to suggest that this value is inappropriate. All other data have minimum values that indicate that the optimum mutation rate for the two-alkane problem is around 0.068.
Octane, Ethane and Pentane

The results for the 40 runs performed on octane ethane and pentane are tabulated in Table 2.16 and selected results (at the 1%, 0.5% and 0.25% thresholds) are displayed graphically in Figures 2.37a-2.39b.

Table 2.16 - Results of GA runs on Octane, Ethane and Pentane.

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>1.25% Cycle Score</th>
<th>1.0% Cycle Score</th>
<th>0.75% Cycle Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0125</td>
<td>101 18.68981</td>
<td>109 18.96411</td>
<td>120 18.96411</td>
</tr>
<tr>
<td>0.0125</td>
<td>65 34.82974</td>
<td>99 50.4069</td>
<td>110 50.43573</td>
</tr>
<tr>
<td>0.0125</td>
<td>155 22.93373</td>
<td>184 30.14993</td>
<td>224 38.84164</td>
</tr>
<tr>
<td>0.0125</td>
<td>18 7.685286</td>
<td>203 32.66445</td>
<td>218 32.66445</td>
</tr>
<tr>
<td>0.0125</td>
<td>127 48.4554</td>
<td>135 48.4554</td>
<td>148 48.4554</td>
</tr>
<tr>
<td>0.025</td>
<td>113 66.1958</td>
<td>139 92.21465</td>
<td>151 92.21465</td>
</tr>
<tr>
<td>0.025</td>
<td>126 75.38400</td>
<td>134 75.63156</td>
<td>148 77.083</td>
</tr>
<tr>
<td>0.025</td>
<td>24 13.91838</td>
<td>37 16.63496</td>
<td>74 23.39999</td>
</tr>
<tr>
<td>0.025</td>
<td>7 20.77316</td>
<td>11 22.25546</td>
<td>13 22.25546</td>
</tr>
<tr>
<td>0.025</td>
<td>37 58.82974</td>
<td>40 58.29797</td>
<td>46 59.04782</td>
</tr>
<tr>
<td>0.0375</td>
<td>14 51.20028</td>
<td>15 51.50264</td>
<td>18 51.50264</td>
</tr>
<tr>
<td>0.0375</td>
<td>46 40.44379</td>
<td>61 48.32457</td>
<td>68 48.32457</td>
</tr>
<tr>
<td>0.0375</td>
<td>19 28.15077</td>
<td>21 28.15077</td>
<td>30 30.4788</td>
</tr>
<tr>
<td>0.0375</td>
<td>16 39.7842</td>
<td>17 39.7842</td>
<td>20 40.0569</td>
</tr>
<tr>
<td>0.0375</td>
<td>125 171.3484</td>
<td>133 171.3484</td>
<td>145 171.3484</td>
</tr>
<tr>
<td>0.05</td>
<td>138 75.29092</td>
<td>149 77.46055</td>
<td>164 78.49946</td>
</tr>
<tr>
<td>0.05</td>
<td>21 69.3794</td>
<td>23 69.3794</td>
<td>27 69.3794</td>
</tr>
<tr>
<td>0.05</td>
<td>17 33.45546</td>
<td>19 33.45546</td>
<td>22 33.84153</td>
</tr>
<tr>
<td>0.05</td>
<td>64 406.2626</td>
<td>70 410.8072</td>
<td>79 414.0018</td>
</tr>
<tr>
<td>0.05</td>
<td>36 30.08786</td>
<td>44 32.76686</td>
<td>60 37.35406</td>
</tr>
<tr>
<td>0.0625</td>
<td>12 23.99113</td>
<td>13 23.99113</td>
<td>29 28.14168</td>
</tr>
<tr>
<td>0.0625</td>
<td>53 23.63598</td>
<td>62 25.3109</td>
<td>75 26.97744</td>
</tr>
<tr>
<td>0.0625</td>
<td>19 33.83691</td>
<td>21 33.83691</td>
<td>24 33.83691</td>
</tr>
<tr>
<td>0.0625</td>
<td>44 126.1568</td>
<td>48 126.1568</td>
<td>54 126.1568</td>
</tr>
<tr>
<td>0.0625</td>
<td>61 44.68452</td>
<td>67 44.68452</td>
<td>77 46.36554</td>
</tr>
<tr>
<td>0.075</td>
<td>23 25.31337</td>
<td>28 26.7897</td>
<td>44 31.52239</td>
</tr>
<tr>
<td>0.075</td>
<td>10 28.44202</td>
<td>11 28.44202</td>
<td>13 28.44202</td>
</tr>
<tr>
<td>0.075</td>
<td>95 85.05506</td>
<td>75 85.05506</td>
<td>83 85.05506</td>
</tr>
<tr>
<td>0.075</td>
<td>11 22.38085</td>
<td>73 40.13717</td>
<td>120 61.30057</td>
</tr>
<tr>
<td>0.075</td>
<td>0 11.92645</td>
<td>2 38.33918</td>
<td>31 50.95307</td>
</tr>
<tr>
<td>0.0875</td>
<td>25 22.72221</td>
<td>27 22.72221</td>
<td>31 22.72221</td>
</tr>
<tr>
<td>0.0875</td>
<td>42 37.34424</td>
<td>57 44.87291</td>
<td>64 44.87291</td>
</tr>
<tr>
<td>0.0875</td>
<td>29 34.11686</td>
<td>32 34.11686</td>
<td>36 34.11686</td>
</tr>
<tr>
<td>0.0875</td>
<td>36 31.72368</td>
<td>40 31.72368</td>
<td>45 31.72368</td>
</tr>
<tr>
<td>0.0875</td>
<td>21 68.64221</td>
<td>30 78.24537</td>
<td>44 88.06015</td>
</tr>
<tr>
<td>0.1</td>
<td>7 21.29777</td>
<td>8 21.29777</td>
<td>9 21.29777</td>
</tr>
<tr>
<td>0.1</td>
<td>6 13.77758</td>
<td>105 40.46511</td>
<td>116 40.46511</td>
</tr>
<tr>
<td>0.1</td>
<td>12 53.33439</td>
<td>13 53.33439</td>
<td>15 53.464</td>
</tr>
<tr>
<td>0.1</td>
<td>43 124.3722</td>
<td>54 143.754</td>
<td>68 158.6812</td>
</tr>
<tr>
<td>0.1</td>
<td>37 316.1649</td>
<td>41 316.1649</td>
<td>46 316.1649</td>
</tr>
</tbody>
</table>
Table 2.16 continued

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>0.5% Cycle</th>
<th>Score</th>
<th>0.25% Cycle</th>
<th>Score</th>
<th>0.1% Cycle</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00125</td>
<td>171</td>
<td>23.76259</td>
<td>221</td>
<td>23.76259</td>
<td>322</td>
<td>23.76259</td>
</tr>
<tr>
<td>0.00125</td>
<td>127</td>
<td>50.43573</td>
<td>168</td>
<td>50.44119</td>
<td>270</td>
<td>53.51319</td>
</tr>
<tr>
<td>0.00125</td>
<td>252</td>
<td>38.84164</td>
<td>317</td>
<td>38.85732</td>
<td>451</td>
<td>38.85732</td>
</tr>
<tr>
<td>0.00125</td>
<td>249</td>
<td>33.50964</td>
<td>317</td>
<td>34.03373</td>
<td>462</td>
<td>35.03377</td>
</tr>
<tr>
<td>0.00125</td>
<td>199</td>
<td>61.47622</td>
<td>267</td>
<td>64.70365</td>
<td>392</td>
<td>65.95444</td>
</tr>
<tr>
<td>0.025</td>
<td>173</td>
<td>92.21464</td>
<td>271</td>
<td>114.211</td>
<td>390</td>
<td>114.211</td>
</tr>
<tr>
<td>0.025</td>
<td>170</td>
<td>77.083</td>
<td>220</td>
<td>77.083</td>
<td>321</td>
<td>77.083</td>
</tr>
<tr>
<td>0.025</td>
<td>88</td>
<td>23.40458</td>
<td>118</td>
<td>23.41782</td>
<td>177</td>
<td>23.41782</td>
</tr>
<tr>
<td>0.025</td>
<td>16</td>
<td>22.25546</td>
<td>108</td>
<td>30.02592</td>
<td>163</td>
<td>30.02592</td>
</tr>
<tr>
<td>0.025</td>
<td>55</td>
<td>59.04782</td>
<td>76</td>
<td>59.36146</td>
<td>121</td>
<td>60.05019</td>
</tr>
<tr>
<td>0.0375</td>
<td>24</td>
<td>52.96293</td>
<td>33</td>
<td>52.96293</td>
<td>76</td>
<td>55.81628</td>
</tr>
<tr>
<td>0.0375</td>
<td>83</td>
<td>49.48821</td>
<td>114</td>
<td>50.47471</td>
<td>261</td>
<td>59.96184</td>
</tr>
<tr>
<td>0.0375</td>
<td>36</td>
<td>30.4788</td>
<td>50</td>
<td>30.4788</td>
<td>77</td>
<td>30.4788</td>
</tr>
<tr>
<td>0.0375</td>
<td>46</td>
<td>47.91008</td>
<td>63</td>
<td>47.91008</td>
<td>162</td>
<td>55.96815</td>
</tr>
<tr>
<td>0.0375</td>
<td>167</td>
<td>171.3484</td>
<td>216</td>
<td>171.3484</td>
<td>316</td>
<td>171.3484</td>
</tr>
<tr>
<td>0.05</td>
<td>187</td>
<td>79.08908</td>
<td>244</td>
<td>80.0724</td>
<td>353</td>
<td>80.0724</td>
</tr>
<tr>
<td>0.05</td>
<td>120</td>
<td>132.7112</td>
<td>164</td>
<td>136.4965</td>
<td>276</td>
<td>149.2467</td>
</tr>
<tr>
<td>0.05</td>
<td>27</td>
<td>34.09466</td>
<td>55</td>
<td>37.57667</td>
<td>156</td>
<td>43.21709</td>
</tr>
<tr>
<td>0.05</td>
<td>93</td>
<td>414.9409</td>
<td>127</td>
<td>420.4329</td>
<td>191</td>
<td>420.4329</td>
</tr>
<tr>
<td>0.05</td>
<td>71</td>
<td>37.35406</td>
<td>103</td>
<td>39.35791</td>
<td>695</td>
<td>125.0268</td>
</tr>
<tr>
<td>0.0625</td>
<td>46</td>
<td>31.35115</td>
<td>74</td>
<td>33.35624</td>
<td>122</td>
<td>33.92252</td>
</tr>
<tr>
<td>0.0625</td>
<td>88</td>
<td>26.97744</td>
<td>132</td>
<td>28.9967</td>
<td>271</td>
<td>34.56631</td>
</tr>
<tr>
<td>0.0625</td>
<td>32</td>
<td>34.91053</td>
<td>48</td>
<td>35.62788</td>
<td>74</td>
<td>35.6366</td>
</tr>
<tr>
<td>0.0625</td>
<td>64</td>
<td>126.1568</td>
<td>87</td>
<td>126.576</td>
<td>340</td>
<td>189.3596</td>
</tr>
<tr>
<td>0.0625</td>
<td>91</td>
<td>46.36554</td>
<td>122</td>
<td>46.64051</td>
<td>203</td>
<td>48.56519</td>
</tr>
<tr>
<td>0.075</td>
<td>53</td>
<td>31.52239</td>
<td>76</td>
<td>32.15768</td>
<td>180</td>
<td>37.53865</td>
</tr>
<tr>
<td>0.075</td>
<td>27</td>
<td>31.84034</td>
<td>162</td>
<td>55.02555</td>
<td>260</td>
<td>57.46037</td>
</tr>
<tr>
<td>0.075</td>
<td>97</td>
<td>85.05506</td>
<td>130</td>
<td>85.05506</td>
<td>423</td>
<td>130.1814</td>
</tr>
<tr>
<td>0.075</td>
<td>160</td>
<td>72.53423</td>
<td>213</td>
<td>74.74373</td>
<td>449</td>
<td>102.3536</td>
</tr>
<tr>
<td>0.075</td>
<td>72</td>
<td>70.06765</td>
<td>106</td>
<td>74.1925</td>
<td>161</td>
<td>74.1925</td>
</tr>
<tr>
<td>0.0875</td>
<td>37</td>
<td>22.72221</td>
<td>56</td>
<td>23.27877</td>
<td>120</td>
<td>25.24847</td>
</tr>
<tr>
<td>0.0875</td>
<td>82</td>
<td>47.73227</td>
<td>137</td>
<td>54.47945</td>
<td>244</td>
<td>58.36922</td>
</tr>
<tr>
<td>0.0875</td>
<td>43</td>
<td>34.11686</td>
<td>110</td>
<td>41.9054</td>
<td>167</td>
<td>41.9054</td>
</tr>
<tr>
<td>0.0875</td>
<td>54</td>
<td>31.902</td>
<td>76</td>
<td>32.72675</td>
<td>749</td>
<td>83.64557</td>
</tr>
<tr>
<td>0.0875</td>
<td>52</td>
<td>88.06015</td>
<td>148</td>
<td>136.0787</td>
<td>255</td>
<td>148.9867</td>
</tr>
<tr>
<td>0.1</td>
<td>116</td>
<td>41.27219</td>
<td>154</td>
<td>41.27219</td>
<td>250</td>
<td>43.4019</td>
</tr>
<tr>
<td>0.1</td>
<td>134</td>
<td>40.46511</td>
<td>177</td>
<td>40.71283</td>
<td>267</td>
<td>41.26156</td>
</tr>
<tr>
<td>0.1</td>
<td>27</td>
<td>58.46249</td>
<td>41</td>
<td>59.84769</td>
<td>749</td>
<td>139.7878</td>
</tr>
<tr>
<td>0.1</td>
<td>80</td>
<td>158.6812</td>
<td>140</td>
<td>185.3772</td>
<td>220</td>
<td>190.5894</td>
</tr>
<tr>
<td>0.1</td>
<td>56</td>
<td>322.3287</td>
<td>96</td>
<td>352.511</td>
<td>184</td>
<td>381.6203</td>
</tr>
</tbody>
</table>
Chapter 2: A GA for Generating Force Field Parameters

\[ y = 25265x^2 - 3738.7x + 170.89 \]
\[ R^2 = 0.3792 \]

**Figure 2.37a** - Graph of 1% Convergence Generation against Mutation Rate for GA runs on octane, ethane and pentane with quadratic trendline.

**Figure 2.37b** - Graph of Best Score at 1% Convergence against Mutation Rate for GA runs on octane, ethane and pentane with linear trendline.
Figure 2.38a - Graph of 0.5% Convergence Generation against Mutation Rate for GA runs on octane, ethane and pentane with quadratic trendline.

Figure 2.38b - Graph of Best Score at 0.5% Convergence against Mutation Rate for GA runs on octane, ethane and pentane with linear trendline.
**Chapter 2: A GA for Generating Force Field Parameters**

The equation for the quadratic trendline is:

\[ y = 40663x^2 - 5660.5x + 294.81 \]

\[ R^2 = 0.3302 \]

**Figure 2.39a** - Graph of 0.25% Convergence Generation against Mutation Rate for GA runs on octane, ethane and pentane with quadratic trendline.

**Figure 2.39b** - Graph of Best Score at 0.25% Convergence against Mutation Rate for GA runs on octane, ethane and pentane with linear trendline.
As for the two-alkane data, the three-alkane runs all indicate that the best score at “convergence” increases with increase in mutation rate.

Each set of data shows a trend towards a minimum value, indicating that an optimum mutation rate exists in each case. The results are shown in Table 2.17.

Table 2.17 - Table of quadratic trendline equations, optimum mutation rates and optimum mutation rate (to 3 decimal places) for each convergence threshold. Optimum mutation rate is calculated as the minimum position in the trendline.

<table>
<thead>
<tr>
<th>Convergence Threshold</th>
<th>Equation of Quadratic Trendline</th>
<th>R-squared Value</th>
<th>Optimum Mutation Rate (3 d. p.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.25%</td>
<td>$y = 8403.8x^2 - 1645.4x + 105.05$</td>
<td>0.258</td>
<td>0.098</td>
</tr>
<tr>
<td>1%</td>
<td>$y = 25265x^2 - 3738.7x + 170.89$</td>
<td>0.3792</td>
<td>0.074</td>
</tr>
<tr>
<td>0.75%</td>
<td>$y = 25897x^2 - 3882x + 187.9$</td>
<td>0.3612</td>
<td>0.075</td>
</tr>
<tr>
<td>0.5%</td>
<td>$y = 32434x^2 - 4655.1x + 226.75$</td>
<td>0.3588</td>
<td>0.072</td>
</tr>
<tr>
<td>0.25%</td>
<td>$y = 40663x^2 - 5660.5x + 294.81$</td>
<td>0.3302</td>
<td>0.070</td>
</tr>
<tr>
<td>0.1%</td>
<td>$y = 54674x^2 - 5901.7x + 397.13$</td>
<td>0.0584</td>
<td>0.054</td>
</tr>
</tbody>
</table>

All data sets display minimum values that indicate that the optimum mutation rate for the three-alkane problem is 0.074 (3 decimal places).

2.6.4.3 Conclusions

The method chosen to predict convergence appears to be suitable for threshold values of 1% or less - a higher value has been shown to be unpredictable at times both in terms of proposing an optimum mutation rate and in the expected increase in best score at convergence as mutation rate increases.

The data generated in this investigation suggests two conclusions:

(a) The optimum mutation rate for fastest convergence for alkanes is between 0.05 and 0.075, and
(b) as the complexity of the problem increases (i.e. generating a force field which is suitable for a larger number of compounds), the optimum mutation rate appears to increase.

2.6.5 Determination of Optimum Convergence Threshold

The data generated in the investigation of optimum mutation rates was also analysed in an attempt to propose the most suitable threshold value for predicting convergence. For each set of data, the regression analysis ($R^2$ value) for the quadratic trendlines was plotted against the threshold value to see if the convergence generations at a particular threshold value were more consistent.

2.6.5.1 Results

The data are tabulated in Table 2.18 and plotted in Figures 2.40a-c. The results are tabulated in Table 2.19.

Table 2.18 - Table of regression values calculated for the quadratic trendline graphs at each convergence threshold.

<table>
<thead>
<tr>
<th>Convergence Threshold</th>
<th>R-squared Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 alkane</td>
</tr>
<tr>
<td>1.25</td>
<td>0.0016</td>
</tr>
<tr>
<td>1</td>
<td>0.1051</td>
</tr>
<tr>
<td>0.75</td>
<td>0.0246</td>
</tr>
<tr>
<td>0.5</td>
<td>0.0869</td>
</tr>
<tr>
<td>0.25</td>
<td>0.051</td>
</tr>
<tr>
<td>0.1</td>
<td>0.0706</td>
</tr>
</tbody>
</table>
Figure 2.40a - Graph of R-squared regression value against convergence threshold for GA runs on octane with quadratic trendline.

Figure 2.40b - Graph of R-squared regression value against convergence threshold for GA runs on octane and ethane with quadratic trendline. The regression value at 1.25% has not been included because the GAs at this threshold were deemed not to have converged.
Figure 2.40c - Graph of R-squared regression value against convergence threshold for GA runs on octane, pentane and ethane with quadratic trendline.

Table 2.19 - Table of quadratic trendline equations, R-squared values and optimum convergence thresholds (to 3 decimal places) for each set of alkanes. Optimum convergence threshold is calculated as the maximum position in the trendline.

<table>
<thead>
<tr>
<th>Alkane Type</th>
<th>Equation of Quadratic Trendline</th>
<th>R-squared Value</th>
<th>Optimum Convergence Threshold (3 d.p.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 alkane</td>
<td>( y = -0.09x^2 + 0.0902x + 0.0504 )</td>
<td>0.2251</td>
<td>0.501</td>
</tr>
<tr>
<td>2 alkanes</td>
<td>( y = -0.6671x^2 + 0.4446x + 0.2558 )</td>
<td>0.8793</td>
<td>0.333</td>
</tr>
<tr>
<td>3 alkanes</td>
<td>( y = -0.6589x^2 + 1.0027x + 0.0262 )</td>
<td>0.797</td>
<td>0.761</td>
</tr>
</tbody>
</table>

2.6.5.2 Conclusions

The results suggest that using a convergence threshold of 0.25% will ensure that a particular GA run has converged.
2.6.6 The Effect of Including Energy in the Score

A series of runs of the GA were performed using Octane as the test compound. The investigation involved four runs at each of the eight mutation rates previously outlined. For each mutation rate, two runs included an energy contribution in the score (as previously described) and two did not. In order to allow a direct comparison between the results, two constant seed values (1 and 2) were chosen to initialise the computer's random number generator. This resulted in two identical populations being initialised at each mutation rate (one for each seed).

In order to gain an early comparison between the runs with and without energy in the score, the results were initially analysed simply in terms of RMS fit at the end of 750 cycles. These results are shown in Table 2.20.

Table 2.20 - RMS fit for Octane after 750 generations for runs at constant seed with and without an energy contribution to the score.

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>RMS Fit, Seed = 1</th>
<th>RMS Fit, Seed = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Energy</td>
<td>Energy</td>
</tr>
<tr>
<td>0.0125</td>
<td>0.488451</td>
<td>0.455985</td>
</tr>
<tr>
<td>0.0250</td>
<td>0.379419</td>
<td>0.398476</td>
</tr>
<tr>
<td>0.0375</td>
<td>0.364556</td>
<td>0.432926</td>
</tr>
<tr>
<td>0.0500</td>
<td>0.373533</td>
<td>0.424203</td>
</tr>
<tr>
<td>0.0625</td>
<td>0.377879</td>
<td>0.395884</td>
</tr>
<tr>
<td>0.0750</td>
<td>0.382866</td>
<td>0.42726</td>
</tr>
<tr>
<td>0.0825</td>
<td>0.381095</td>
<td>0.419318</td>
</tr>
<tr>
<td>0.1000</td>
<td>0.363195</td>
<td>0.425998</td>
</tr>
</tbody>
</table>

The initial indication from this raw data was that, in the majority of cases (15 out of 16), including the energy in the score degraded the performance of the genetic algorithm in terms of finding a force field which accurately predicts the geometry of the test compound. However, in order to provide a more accurate comparison, the data was analysed similarly to the previous alkane data.
2.6.6.1 Results

When the data from these GA runs is compared, it becomes immediately apparent that including the energy in the score decreases the scores generated for the force fields considerably (in some cases by an order of magnitude for the best score after 750 generations). Since the program does not practice any form of fitness scaling, this difference does not affect the roulette wheel selection process, allowing comparisons to be drawn between the runs including and excluding energy in the score. In order to make these comparisons, the raw data was treated to provide the percentage improvement in average best score. Given the results of the investigation into convergence thresholds, convergence criteria of 0.75%, 0.5%, 0.25% and 0.1% were chosen. The convergence generations at each of these thresholds is shown in Tables 2.21 (seed = 1) and 2.22 (seed = 2).

Table 2.21 - Comparison of convergence generation at various thresholds for runs including and excluding energy from the score with seed = 1.

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>Threshold = 0.75%</th>
<th></th>
<th></th>
<th>Threshold = 0.50%</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Energy</td>
<td>Energy</td>
<td>No Energy</td>
<td>Energy</td>
<td>No Energy</td>
</tr>
<tr>
<td>0.0125</td>
<td>120</td>
<td>11</td>
<td>139</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>0.025</td>
<td>165</td>
<td>77</td>
<td>186</td>
<td>91</td>
<td>91</td>
</tr>
<tr>
<td>0.0375</td>
<td>105</td>
<td>72</td>
<td>131</td>
<td>106</td>
<td>106</td>
</tr>
<tr>
<td>0.05</td>
<td>94</td>
<td>59</td>
<td>109</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>0.0625</td>
<td>72</td>
<td>51</td>
<td>88</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>0.075</td>
<td>70</td>
<td>47</td>
<td>83</td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td>0.0875</td>
<td>88</td>
<td>59</td>
<td>103</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>0.1</td>
<td>75</td>
<td>51</td>
<td>91</td>
<td>55</td>
<td>55</td>
</tr>
</tbody>
</table>
Table 2.21 - Continued

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>Threshold = 0.25%</th>
<th>Threshold = 0.10%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Energy</td>
<td>Energy</td>
</tr>
<tr>
<td>0.0125</td>
<td>182</td>
<td>131</td>
</tr>
<tr>
<td>0.025</td>
<td>240</td>
<td>129</td>
</tr>
<tr>
<td>0.0375</td>
<td>172</td>
<td>158</td>
</tr>
<tr>
<td>0.05</td>
<td>146</td>
<td>99</td>
</tr>
<tr>
<td>0.0625</td>
<td>141</td>
<td>142</td>
</tr>
<tr>
<td>0.075</td>
<td>144</td>
<td>103</td>
</tr>
<tr>
<td>0.0875</td>
<td>144</td>
<td>98</td>
</tr>
<tr>
<td>0.1</td>
<td>128</td>
<td>75</td>
</tr>
</tbody>
</table>

Table 2.22 - Comparison of convergence generation at various thresholds for runs including and excluding energy from the score with seed = 2.

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>Threshold = 0.75%</th>
<th>Threshold = 0.50%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Energy</td>
<td>Energy</td>
</tr>
<tr>
<td>0.0125</td>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td>0.025</td>
<td>30</td>
<td>34</td>
</tr>
<tr>
<td>0.0375</td>
<td>36</td>
<td>34</td>
</tr>
<tr>
<td>0.05</td>
<td>13</td>
<td>20</td>
</tr>
<tr>
<td>0.0625</td>
<td>33</td>
<td>23</td>
</tr>
<tr>
<td>0.075</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>0.0875</td>
<td>21</td>
<td>29</td>
</tr>
<tr>
<td>0.1</td>
<td>25</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 2.22 - Continued

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>Threshold = 0.25%</th>
<th>Threshold = 0.10%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Energy</td>
<td>Energy</td>
</tr>
<tr>
<td>0.0125</td>
<td>46</td>
<td>48</td>
</tr>
<tr>
<td>0.025</td>
<td>96</td>
<td>58</td>
</tr>
<tr>
<td>0.0375</td>
<td>73</td>
<td>60</td>
</tr>
<tr>
<td>0.05</td>
<td>48</td>
<td>33</td>
</tr>
<tr>
<td>0.0625</td>
<td>71</td>
<td>47</td>
</tr>
<tr>
<td>0.075</td>
<td>378</td>
<td>31</td>
</tr>
<tr>
<td>0.0875</td>
<td>69</td>
<td>60</td>
</tr>
<tr>
<td>0.1</td>
<td>56</td>
<td>55</td>
</tr>
</tbody>
</table>
The results of these GA runs in terms of best score and RMS fit at convergence are tabulated in Tables 2.23 (seed = 1) and 2.24 (seed = 2). In each table, comparisons where the GA run including an energy contribution produced a better RMS fit are shown in bold italics.

**Table 2.23 - Comparison of best score and rms fit at convergence from runs including and excluding energy from the score, with seed = 1.**

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>Threshold = 0.75% Score</th>
<th>RMS</th>
<th>Energy Score</th>
<th>RMS</th>
<th>Threshold = 0.50% Score</th>
<th>RMS</th>
<th>Energy Score</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0125</td>
<td>40.66527</td>
<td>0.495893</td>
<td>2.645139</td>
<td>1.189285</td>
<td>40.69287</td>
<td>0.495725</td>
<td>5.307169</td>
<td>0.618009</td>
</tr>
<tr>
<td>0.0250</td>
<td>64.86227</td>
<td>0.392648</td>
<td>6.792589</td>
<td>0.459556</td>
<td>65.03362</td>
<td>0.392131</td>
<td>6.836903</td>
<td>0.455442</td>
</tr>
<tr>
<td>0.0375</td>
<td>62.19669</td>
<td>0.400974</td>
<td>5.156497</td>
<td>0.616305</td>
<td>67.10118</td>
<td>0.386042</td>
<td>6.168236</td>
<td>0.43711</td>
</tr>
<tr>
<td>0.0500</td>
<td>64.66353</td>
<td>0.393246</td>
<td>5.059411</td>
<td>0.61776</td>
<td>64.66351</td>
<td>0.393246</td>
<td>5.121912</td>
<td>0.615169</td>
</tr>
<tr>
<td>0.0625</td>
<td>54.44948</td>
<td>0.428552</td>
<td>5.281603</td>
<td>0.616045</td>
<td>55.85615</td>
<td>0.423121</td>
<td>5.29861</td>
<td>0.61391</td>
</tr>
<tr>
<td>0.0750</td>
<td>58.55805</td>
<td>0.413244</td>
<td>5.245192</td>
<td>0.618864</td>
<td>59.00862</td>
<td>0.411663</td>
<td>6.154659</td>
<td>0.517564</td>
</tr>
<tr>
<td>0.0825</td>
<td>59.4667</td>
<td>0.410075</td>
<td>6.757302</td>
<td>0.440926</td>
<td>59.91618</td>
<td>0.408534</td>
<td>6.757302</td>
<td>0.440926</td>
</tr>
<tr>
<td>0.100</td>
<td>60.28962</td>
<td>0.407267</td>
<td>6.362492</td>
<td>0.479536</td>
<td>61.86429</td>
<td>0.40205</td>
<td>6.579102</td>
<td>0.458429</td>
</tr>
</tbody>
</table>

**Table 2.23 - Continued**

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>Threshold = 0.25% Score</th>
<th>RMS</th>
<th>Energy Score</th>
<th>RMS</th>
<th>Threshold = 0.1% Score</th>
<th>RMS</th>
<th>Energy Score</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0125</td>
<td>40.69287</td>
<td>0.495725</td>
<td>5.311687</td>
<td>0.617929</td>
<td>40.72664</td>
<td>0.49519</td>
<td>6.479384</td>
<td>0.489912</td>
</tr>
<tr>
<td>0.0250</td>
<td>65.23392</td>
<td>0.391528</td>
<td>7.075958</td>
<td>0.435867</td>
<td>70.36071</td>
<td>0.376994</td>
<td>7.210092</td>
<td>0.425741</td>
</tr>
<tr>
<td>0.0375</td>
<td>67.19716</td>
<td>0.385767</td>
<td>6.786835</td>
<td>0.437897</td>
<td>67.48562</td>
<td>0.384941</td>
<td>6.820668</td>
<td>0.437635</td>
</tr>
<tr>
<td>0.0500</td>
<td>64.66353</td>
<td>0.393246</td>
<td>5.18903</td>
<td>0.616321</td>
<td>64.85518</td>
<td>0.39267</td>
<td>7.079014</td>
<td>0.453652</td>
</tr>
<tr>
<td>0.0625</td>
<td>64.57577</td>
<td>0.393519</td>
<td>7.12018</td>
<td>0.42778</td>
<td>66.29547</td>
<td>0.388381</td>
<td>7.135407</td>
<td>0.427582</td>
</tr>
<tr>
<td>0.0750</td>
<td>67.14437</td>
<td>0.385918</td>
<td>6.33624</td>
<td>0.490962</td>
<td>67.69892</td>
<td>0.384334</td>
<td>6.646334</td>
<td>0.478644</td>
</tr>
<tr>
<td>0.0825</td>
<td>62.08387</td>
<td>0.401338</td>
<td>6.875962</td>
<td>0.439871</td>
<td>68.84455</td>
<td>0.381123</td>
<td>6.981762</td>
<td>0.431376</td>
</tr>
<tr>
<td>0.100</td>
<td>63.71608</td>
<td>0.396164</td>
<td>6.579102</td>
<td>0.458429</td>
<td>66.88494</td>
<td>0.386666</td>
<td>6.59188</td>
<td>0.458626</td>
</tr>
</tbody>
</table>
Table 2.24 - Comparison of best score and rms fit at convergence from runs including and excluding energy from the score, with seed = 2.

| Mutation Rate | Threshold = 0.75% | | | Threshold = 0.50% | | |
|---|---|---|---|---|---|
| | No Energy | Energy | No Energy | Energy |
| | Score | RMS | Score | RMS | Score | RMS |
| 0.0125 | 234.5283 | 0.206492 | 10.27269 | 0.22191 | 234.5283 | 0.206492 | 10.2851 | 0.221937 |
| 0.0250 | 359.6949 | 0.166737 | 10.85357 | 0.195777 | 369.8156 | 0.16444 | 10.86453 | 0.196936 |
| 0.0375 | 277.9041 | 0.189694 | 10.55329 | 0.21586 | 278.8847 | 0.18936 | 10.56725 | 0.215102 |
| 0.0500 | 236.311 | 0.205711 | 10.71366 | 0.200882 | 263.4127 | 0.194842 | 10.71366 | 0.200882 |
| 0.0625 | 253.9052 | 0.198456 | 10.44173 | 0.222955 | 254.2776 | 0.198311 | 10.50879 | 0.219852 |
| 0.0750 | 243.9759 | 0.202454 | 10.81671 | 0.206751 | 395.4073 | 0.159029 | 10.90544 | 0.204329 |
| 0.0825 | 333.3523 | 0.1732 | 10.10039 | 0.190792 | 494.6161 | 0.142189 | 10.82062 | 0.20305 |
| 0.100 | 494.6161 | 0.142189 | 10.54967 | 0.209109 | 494.6161 | 0.142189 | 10.75219 | 0.200762 |

Table 2.24 - Continued.

| Mutation Rate | Threshold = 0.25% | | | Threshold = 0.1% | | |
|---|---|---|---|---|---|
| | No Energy | Energy | No Energy | Energy |
| | Score | RMS | Score | RMS | Score | RMS |
| 0.0125 | 234.5283 | 0.206492 | 10.34246 | 0.219765 | 234.5283 | 0.206492 | 10.64919 | 0.208125 |
| 0.0250 | 434.9424 | 0.15163 | 10.95924 | 0.196178 | 462.4607 | 0.147049 | 10.97774 | 0.195801 |
| 0.0375 | 278.8847 | 0.18936 | 10.81671 | 0.206751 | 395.4073 | 0.159029 | 10.90544 | 0.204329 |
| 0.0500 | 267.8174 | 0.193233 | 10.71366 | 0.200882 | 267.8174 | 0.193233 | 10.71912 | 0.201041 |
| 0.0625 | 258.6662 | 0.196621 | 10.92915 | 0.201648 | 694.6778 | 0.11998 | 11.08858 | 0.193738 |
| 0.0750 | 830.9677 | 0.1097 | 10.47199 | 0.183023 | 845.8656 | 0.10873 | 10.49457 | 0.176308 |
| 0.0825 | 376.1788 | 0.163043 | 10.91762 | 0.200132 | 387.637 | 0.160615 | 11.1681 | 0.193248 |
| 0.100 | 509.2385 | 0.140133 | 10.90974 | 0.205556 | 528.1747 | 0.137598 | 10.90974 | 0.205556 |

2.6.6.2 Conclusions

The tables of convergence generation for each set of data show a tendency for the runs with an energy contribution in the score to converge more rapidly than those which do not include energy.
The variability in a population may be analysed by comparing the mean average and mean best score at each generation throughout a run. (The use of these values rather than the average and best scores provides a smoother graph, allowing for easier comparison.) When these ratios are compared for the four runs at each mutation rate it becomes clear that, as a run proceeds, they are higher for the runs including energy (both seeds). (Two examples of these graphs are shown in Figures 2.41 and 2.42 for mutation rates of 0.0125 and 0.0825).

*Figure 2.41 - Graph of mean average / mean best score for four GA runs at 0.0125 mutation rate.*
This indicates that the populations which do not include energy contain a wider spread of schema, allowing a more thorough exploration of the search space and therefore later convergence.

The RMS results shown above suggest that including a contribution from the molecule’s energy in the GA’s score in the manner investigated is not beneficial to generating a force field which accurately models the molecular structure of octane in the crystalline state. Of the 64 comparisons drawn at “convergence”, only four show an improvement in RMS fit by including energy.

This conclusion is obviously based partly upon the assumption that it is appropriate to treat both sets of data in the same way in order to determine
convergence. However, the data originally presented after 750 generations also suggests that, in a raw comparison of performance, the GAs which do not include energy are likely to give a “better” force field.

Due to the limited data generated in this investigation, it was not appropriate to test the runs using energy in the score for optimum mutation rates and convergence criteria.

2.6.7 Two Point Crossover

In order to investigate the effect of two point crossover on the two methods of scoring the force fields (i.e. excluding and including an energy contribution) a new reproduction routine was developed. Runs were performed on octane using this module with 51 force fields for 750 generations. A mutation rate of 0.1 and constant random number seeds of 1 and 2 were used, enabling direct comparisons with runs generated in the previous investigation.

2.6.7.1 Results

Once more the results were initially analysed simply in terms of RMS fit at the end of 750 cycles enabling an early comparison between the runs with and without energy in the score for one and two point crossover. These results are shown in Table 2.25.

Table 2.25 - RMS fit for Octane after 750 generations for runs at constant seed with one and two point crossover, with and without an energy contribution to the score.

<table>
<thead>
<tr>
<th>Crossover Type</th>
<th>RMS Fit, Seed = 1</th>
<th>RMS Fit, Seed = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Energy</td>
<td>Energy</td>
</tr>
<tr>
<td>one point</td>
<td>0.363195</td>
<td>0.425998</td>
</tr>
<tr>
<td>two point</td>
<td>0.389391</td>
<td>0.430899</td>
</tr>
</tbody>
</table>
Analysis of these raw results indicates that including energy in the score is detrimental to the program for two point crossover as well as for one point. (Indeed, two point crossover excluding energy performs better than one point including energy in both cases.) For both seeds, the one point crossover without energy performed the best of all four algorithms. The raw data does not allow a conclusion to be drawn when one and two point crossovers including energy are compared.

The eight runs were analysed as previously with convergence thresholds of 0.75%, 0.5%, 0.25% and 0.1 %. The results (convergence generation and RMS fit at convergence) are shown in Tables 2.26 - 2.27 and displayed graphically in Figures 2.43 - 2.46.

**Table 2.26 - Comparison of one point and two point crossover runs with and without energy included in the score - populations initialised with seed = 1.**

<table>
<thead>
<tr>
<th>Convergence Threshold</th>
<th>One Point Crossover</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Energy</td>
<td>Energy</td>
</tr>
<tr>
<td></td>
<td>Generation</td>
<td>RMS Fit</td>
</tr>
<tr>
<td>0.75%</td>
<td>75</td>
<td>0.407267</td>
</tr>
<tr>
<td>0.5%</td>
<td>91</td>
<td>0.40205</td>
</tr>
<tr>
<td>0.25%</td>
<td>128</td>
<td>0.396164</td>
</tr>
<tr>
<td>0.1%</td>
<td>230</td>
<td>0.38196</td>
</tr>
</tbody>
</table>
Table 2.26 - Continued

<table>
<thead>
<tr>
<th>Convergence Threshold</th>
<th>No Energy</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generation</td>
<td>RMS Fit</td>
</tr>
<tr>
<td>0.75%</td>
<td>81</td>
<td>0.427763</td>
</tr>
<tr>
<td>0.5%</td>
<td>95</td>
<td>0.426951</td>
</tr>
<tr>
<td>0.25%</td>
<td>130</td>
<td>0.423182</td>
</tr>
<tr>
<td>0.1%</td>
<td>196</td>
<td>0.423176</td>
</tr>
</tbody>
</table>

Table 2.27 - Comparison of one point and two point crossover runs with and without energy included in the score - populations initialised with seed = 2.

<table>
<thead>
<tr>
<th>Convergence Threshold</th>
<th>No Energy</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generation</td>
<td>RMS Fit</td>
</tr>
<tr>
<td>0.75%</td>
<td>31</td>
<td>0.142189</td>
</tr>
<tr>
<td>0.5%</td>
<td>37</td>
<td>0.142189</td>
</tr>
<tr>
<td>0.25%</td>
<td>56</td>
<td>0.140133</td>
</tr>
<tr>
<td>0.1%</td>
<td>106</td>
<td>0.137598</td>
</tr>
</tbody>
</table>

Table 2.27 - Continued

<table>
<thead>
<tr>
<th>Convergence Threshold</th>
<th>No Energy</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generation</td>
<td>RMS Fit</td>
</tr>
<tr>
<td>0.75%</td>
<td>66</td>
<td>0.138243</td>
</tr>
<tr>
<td>0.5%</td>
<td>78</td>
<td>0.138243</td>
</tr>
<tr>
<td>0.25%</td>
<td>157</td>
<td>0.122095</td>
</tr>
<tr>
<td>0.1%</td>
<td>238</td>
<td>0.121437</td>
</tr>
</tbody>
</table>
Figure 2.43 - Graph of RMS Fit achieved at each convergence threshold for GA runs initialised with a seed of 1.

Figure 2.44 - Graph of generation of convergence at each threshold for GA runs initialised with a seed of 1.
Figure 2.45 - Graph of RMS Fit achieved at each convergence threshold for GA runs initialised with a seed of 2.

Figure 2.46 - Graph of generation of convergence at each threshold for GA runs initialised with a seed of 2.
In all cases (both one and two point crossover), the runs which included energy in the score did not find a force field which reproduced the structure of Octane as well as that found by not including energy. These runs also converged more rapidly.

When the runs which used two point crossover are compared with the corresponding ones using one point, it is more difficult to draw any conclusions. With a seed of one, the general trend is for one point crossover to produce a better force field at convergence than two point. However, this situation is completely reversed when two is used as the random number generator seed.

2.6.7.2 Conclusions

Whilst the results after 750 generations seem to indicate that one point is the preferred method of crossover, the data compared at “convergence” is inconclusive in confirming this.

A possible reason for this discrepancy may be that it is inappropriate to treat data generated from two point crossover runs in the same way as one point and attempt to draw direct comparisons.

A more important limitation of two point crossover in this problem is the nature of the force fields. Of the four parameter sets optimised (bond, angle, torsion and non-bond), two have only two entries (bond and non-bond). These can experience no benefit from two-point crossover, since at least one of the cut points will always be at the beginning or the end of the parameter set. In order to provide a more rigorous test of two point crossover, a more complex molecule (and hence force field) must be studied.
2.6.8 Application of the GA to a Coordination Complex

2.6.8.1 Initial Investigation

Since the GA seemed to generate appropriate parameters for alkanes, it was decided that it should be tested with a coordination compound. To avoid the added difficulties of parameterising for multiple equilibrium angles, a tetrahedral nickel complex, (dichloro-((-)-2,2-dimethyl-4,5-bis(diphenylphosphinomethyl)-1,3-dioxolane-P,P')-nickel, refcode CIPBNI10) was selected from the CSD. Its structure is shown in Figure 2.47.

![Figure 2.47 - The structure of the tetrahedral nickel complex with refcode CIPBNI10. Hydrogen atoms have been removed for clarity.](image)

The genetic algorithm was run with 51 force fields for a greatly extended run of 3500 generations. This run took approximately 25 days in total to complete, running in the background on the Silicon Graphics Crimson whilst other calculations were being performed. A graph of the average and best fitness for this run is shown in Figure 2.48. Fitness was calculated as being equal to \(200/[(\Sigma \text{RMS})^3]\).
The run converged at around 1000 generations, with only minor improvements after this point. The best score gave an RMS difference between the minimised and crystal structures of 0.524. The angles around the metal centre were reproduced relatively well, but the Ni-P and Ni-Cl bond lengths were too short. The bond lengths and angles found in the crystal structure and in that generated by the derived force field are compared in Table 2.28. The force field parameters giving rise to these values are shown in Table 2.29.

**Table 2.28 - Comparison of crystal and modelled structures for the NiP_2Cl_2 unit.**

<table>
<thead>
<tr>
<th></th>
<th>Crystal Structure (Å, 3 d.p.)</th>
<th>Modelled Structure (Å, 3 d.p.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bonds</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ni-Cl</td>
<td>2.204, 2.204</td>
<td>1.309, 1.306</td>
</tr>
<tr>
<td>Ni-P</td>
<td>2.305, 2.285</td>
<td>2.044, 2.059</td>
</tr>
<tr>
<td><strong>Angles</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cl-Ni-Cl</td>
<td>129.7</td>
<td>137.1</td>
</tr>
<tr>
<td>P-Ni-Cl</td>
<td>112.5, 109.0, 100.3, 99.1</td>
<td>105.7, 105.3, 102.5, 101.6</td>
</tr>
<tr>
<td>P-Ni-P</td>
<td>102.7</td>
<td>98.7</td>
</tr>
</tbody>
</table>
Table 2.29 - Selected parameters derived by the GA for the NiP₂Cl₂ unit.

<table>
<thead>
<tr>
<th>Bonds</th>
<th>$k_b$ (kcal/mol Å²)</th>
<th>$b_0$ (Å)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ni-Cl</td>
<td>306.8</td>
<td>1.31</td>
</tr>
<tr>
<td>Ni-P</td>
<td>313.2</td>
<td>2.065</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Angles</th>
<th>$k_0$ (kcal/mol rad²)</th>
<th>$\theta_0$ (°)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cl-Ni-Cl</td>
<td>44</td>
<td>141.7</td>
</tr>
<tr>
<td>P -Ni-Cl</td>
<td>74</td>
<td>105.8</td>
</tr>
<tr>
<td>P -Ni-P</td>
<td>44</td>
<td>106.3</td>
</tr>
</tbody>
</table>

Much of the difference between the crystal and minimised structures lies in the organic part of the molecule, and the inclusion of a series of suitable organic templates (e.g. benzene for the aromatic carbons), may have improved this section of the force field, although obviously the run time of the program would have increased correspondingly.

The convergence of the GA at these values appears to indicate that there was not enough genetic variability in the population. The force fields needed to model this compound are far larger than the ones needed for alkanes, and the spread of information in the force fields created by the genetic algorithm may not be wide enough to find a better solution.

Strategies which would increase genetic variability without significantly increasing the run time are to optimise a greater number of force fields for fewer generations or to increase the mutation rate towards the end of the run.

2.6.8.1 Further Investigations

The former of these two strategies was chosen for GA runs using one and two point crossover both including and excluding energy from the score. The runs were performed on 501 force fields over 1500 generations. A constant mutation
rate of 0.1 was used for each run. The random number generator was seeded from the system clock.

Results

Once again, the results were initially analysed in terms of RMS fit at the end of the run. These results are shown in Table 2.30.

Table 2.30 - RMS fit for Nickel complex after 1500 generations with one and two point crossover, with and without an energy contribution to the score.

<table>
<thead>
<tr>
<th>Crossover Type</th>
<th>RMS Fit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Energy</td>
</tr>
<tr>
<td>one point</td>
<td>0.219044</td>
</tr>
<tr>
<td>two point</td>
<td>0.38564</td>
</tr>
</tbody>
</table>

Three of the four runs produced force fields which modelled the compound more accurately than the force field produced in the initial test run, showing that the optimisation of a larger number of force fields for fewer generations is an appropriate way of improving the genetic algorithm’s performance.

Once again, using one point crossover without energy produced the best result - however trends could not be proposed from the raw data either in terms of crossover type or scoring method.

The four GA runs were therefore analysed in the same way as previously described with convergence thresholds of 0.75%, 0.5%, 0.25% and 0.1%. The results (convergence generation and RMS fit at convergence) are shown in Table 2.31 and displayed graphically in Figures 2.49 and 2.50.
Table 2.31 - Comparison of one point and two point crossover runs with and without energy included in the score for Nickel complex.

<table>
<thead>
<tr>
<th>Convergence Threshold</th>
<th>One Point Crossover</th>
<th></th>
<th>Two Point Crossover</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Energy</td>
<td>Energy</td>
<td>No Energy</td>
<td>Energy</td>
</tr>
<tr>
<td></td>
<td>Generation</td>
<td>RMS Fit</td>
<td>Generation</td>
<td>RMS Fit</td>
</tr>
<tr>
<td>0.75%</td>
<td>66</td>
<td>0.435176</td>
<td>30</td>
<td>0.885346</td>
</tr>
<tr>
<td>0.5%</td>
<td>78</td>
<td>0.435176</td>
<td>36</td>
<td>0.885346</td>
</tr>
<tr>
<td>0.25%</td>
<td>203</td>
<td>0.343544</td>
<td>92</td>
<td>0.799462</td>
</tr>
<tr>
<td>0.1%</td>
<td>719</td>
<td>0.239339</td>
<td>158</td>
<td>0.790332</td>
</tr>
</tbody>
</table>

Table 2.31 - Continued

<table>
<thead>
<tr>
<th>Convergence Threshold</th>
<th>Two Point Crossover</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Energy</td>
<td>Energy</td>
<td>No Energy</td>
</tr>
<tr>
<td></td>
<td>Generation</td>
<td>RMS Fit</td>
<td>Generation</td>
</tr>
<tr>
<td>0.75%</td>
<td>74</td>
<td>0.646675</td>
<td>27</td>
</tr>
<tr>
<td>0.5%</td>
<td>134</td>
<td>0.548678</td>
<td>62</td>
</tr>
<tr>
<td>0.25%</td>
<td>189</td>
<td>0.529815</td>
<td>194</td>
</tr>
<tr>
<td>0.1%</td>
<td>278</td>
<td>0.529815</td>
<td>396</td>
</tr>
</tbody>
</table>

Figure 2.49 - Graph of RMS Fit achieved at each convergence threshold for Nickel complex.
When the convergence data is analysed it is seen that over the majority of the generations in the GA runs, including energy in the score degrades the performance of the genetic algorithm - this is true for both one and two point crossover. When the relative performances of one and two point crossover are compared, it is more difficult to draw conclusions. When one point crossover is considered, excluding energy from the score causes the resultant force field to be more accurate in modelling the complex's structure. The converse is true with two point crossover.

When convergence data is analysed, it is again apparent that the GAs which include an energy contribution in the score converge more rapidly than those which do not.
The values of the bond lengths and angles for the NiP₂Cl₂ unit from the crystal structure and the most accurate force field (one point crossover, no energy) are shown in Table 2.32. Whilst the generated structure is still some way from that found in the crystal, all lengths and angles are modelled more accurately than in the previous investigation, indicating that investigating a larger population for fewer generations is likely to provide a more accurate solution. The parameters derived for the force field are given in Table 2.33.

**Table 2.32 - Comparison of crystal and modelled structures for the NiP₂Cl₂ unit.**

<table>
<thead>
<tr>
<th>Bonds</th>
<th>Crystal Structure (Å, 3 d.p.)</th>
<th>Modelled Structure (Å, 3 d.p.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ni-Cl</td>
<td>2.204, 2.204</td>
<td>1.676, 1.676</td>
</tr>
<tr>
<td>Ni-P</td>
<td>2.305, 2.285</td>
<td>2.365, 2.364</td>
</tr>
<tr>
<td>Angles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cl-Ni-Cl</td>
<td>129.7</td>
<td>133.2</td>
</tr>
<tr>
<td>P-Ni-Cl</td>
<td>112.5, 109.0, 100.3, 99.1</td>
<td>106.4, 105.5, 103.7, 103.2</td>
</tr>
<tr>
<td>P-Ni-P</td>
<td>102.7</td>
<td>100.6</td>
</tr>
</tbody>
</table>

**Table 2.33 - Selected parameters derived by the GA for the NiP₂Cl₂ unit.**

<table>
<thead>
<tr>
<th>Bonds</th>
<th>k₀ (kcal/mol/Å²)</th>
<th>b₀ (Å)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ni-Cl</td>
<td>323.9</td>
<td>1.677</td>
</tr>
<tr>
<td>Ni-P</td>
<td>310.4</td>
<td>2.372</td>
</tr>
<tr>
<td>Angles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cl-Ni-Cl</td>
<td>51</td>
<td>139.7</td>
</tr>
<tr>
<td>P-Ni-Cl</td>
<td>62</td>
<td>106.6</td>
</tr>
<tr>
<td>P-Ni-P</td>
<td>57</td>
<td>107.5</td>
</tr>
</tbody>
</table>
Conclusions
As expected, optimising a larger population for fewer generations is a more effective strategy for running the GA. The resulting force field is better able to model the crystal structure. Alternative methods which may further improve this performance are discussed in the next section.

Once again, the limited data indicates that better results may be obtained on the whole by excluding the energy contribution from the score, although further investigations are needed to confirm this.

Unfortunately, it is impossible to draw conclusions on the effectiveness of two point crossover when compared with one point - more data is required.

2.6.9 Future Developments
The eventual aim of a fully automated system requires direct access to crystal data through an on-line link to the CSD coupled with a fast optimisation process. The GA produced is very much a prototype system, and a variety of different techniques could be tried in order to improve its efficiency. The following proposals may provide the desired enhancements:

2.6.9.1 Force Field Generation
There is no necessity to repeatedly regenerate existing force field parameters, especially not at the beginning of a run, when existing parameters would be more than adequate. Any standard parameters used in the initial force fields could then begin optimising later in the procedure, to take account of any unusual structural features in the molecules under investigation. This strategy would focus the GA’s
attention upon the new parameters being generated, effectively increasing the
selection pressure in favour of the better performing force fields.

Currently, the program must be recompiled for each set of compounds studied. A
major improvement would be the automatic generation of the force field files
without human intervention. This should be possible - the minimiser reports any
parameters that are missing from the force field. These messages could be
intercepted and used to provide a list of new parameters required.

2.6.9.2 Force Field Optimisation Strategy
Rather than modifying all four sets of parameters (bonds, angles, torsions and
non-bonds), simultaneously, it may be more efficient to spend time optimising
each type in turn. Once this has been carried out for all four categories, the best
solutions of each type can be optimised together.

2.6.9.3 Minimiser
The use of an alternative minimiser may improve the program’s efficiency,
especially if the genetic algorithm is to be run on a hardware platform which
supports vectorised code. Minimisers which have been modified to deal with
coordination compounds are now more readily available, and the use of such a
program would allow the incorporation of a wider range of existing parameter
sets.

2.6.9.4 The Evaluation Module
The equation selected to evaluate the score in these calculations may not be the
best way to separate the force fields, and other methods could be investigated.
It has been shown that including a contribution from the molecule’s energy in the score in the manner investigated has a detrimental effect on the GA in terms of finding a force field which accurately predicts the molecular structure of alkanes in the crystalline state. The populations suffered from a lower variability and therefore converged earlier.

Other methods of evaluating the force field which include an energy contribution should be investigated to prove whether these conclusions are simply a result of this technique, or if the explicit inclusion of an energy term always leads to this result.

2.6.9.5 The Reproduction Module
A variety of alternative techniques are available for controlling how the next generation is formed. This GA has been extensively investigated with one point crossover with preliminary work using two point. More complicated forms of crossover may be more efficient at exchanging information, especially for larger force fields. It may also be more efficient to allow more of the better force fields to survive into the next generation unaltered.

Further investigations of optimum mutation rate are also required to verify the transferability of the results obtained to other crossovers and sets of compounds. The variation of crossover and mutation rates as the run proceeds should also be investigated, as, towards the end of the run, an increase in the mutation rate is the only way of introducing genetic variability into a ‘stagnant’ population.

2.6.9.6 A Distributed Genetic Algorithm
The implementation of a distributed GA, where a series of separate populations are optimised in isolation to each other should result in fewer good force fields
being lost. This is because copies of the best individual found in each separate population are periodically copied around to the other sets, ensuring that genetic variability is maintained throughout the whole population.

### 2.6.9.7 A Parallel Genetic Algorithm

As previously mentioned, the best way to improve the performance is to parallelise the system. The most time consuming section of the GA is the evaluation section. If a series of processors could be used, each evaluating potential force fields with the test molecules, the run time for the overall program could be dramatically reduced.

### 2.6.10 Conclusions

The production of molecular mechanics parameters can be extremely time consuming and tedious. The procedure requires not only the initial gathering and analysis of data from sources such as the Cambridge Structural Database but also the iterative optimisation of these parameters until an acceptable result is achieved.

A prototype genetic algorithm has been developed for the generation of force field parameters for predicting the structures of molecular units within the crystalline environment. The starting population is initialised by generating random values for bond lengths, angles etc., centred around typical values obtained by searching the CSD. Currently, initial values for force constants are centred around values from existing parameters, which seem to provide an adequate starting position.

The performance of the GA has been investigated in a variety of ways. A method has been proposed for determining when the algorithm has converged, and an optimum mutation rate has been suggested for one-point crossover on simple alkanes.
Results from all investigations (one and two point crossover on alkanes and the Nickel complex) suggest that a more accurate force field for structure prediction is produced if the genetic algorithm does not contain a contribution from the energy in the score. Including energy in the way investigated also appears to cause earlier convergence. Further work is suggested with alternative ways of incorporating an energy contribution to verify these conclusions.

Early investigations of two point crossover have proved inconclusive and further runs are required using more complex force fields to ascertain if this or other more complex methods of crossover are more effective than the one point crossover initially used.

The program has been shown to generate parameters which are more effective at reproducing crystalline structures of simple alkanes than those provided by Cole. Parameters have also been determined for the tetrahedral nickel complex (dichloro-(((-)-2,2-dimethyl-4,5-bis(diphenylphosphinomethyl)-1,3-dioxolane-P,P'))-nickel which give an RMS fit with the crystal structure of 0.219.

However, the operation of the program is extremely time consuming and requires considerable human intervention at the beginning of the run. Modifications have been suggested which address both these points.
Hardware Considerations

3.1 Introduction 167
3.2 Enhancing A PC's Processing Power 167
   3.2.1 Available Processors 168
       3.2.1.1 The Intel i860 168
       3.2.1.2 The Texas Instruments TMS34020 171
   3.2.2 Enhancing Computational Power 172
       3.2.2.1 Coprocessors 172
       3.2.2.2 Multi- and Parallel Processing Systems 173
       3.2.2.3 Transputer Boards 178
       3.2.2.4 MicroWay i860 Boards 180
   3.2.3 Enhancing Graphics Capability 181
       3.2.3.1 High Performance Graphics Cards 182
3.3 PC Hardware Incompatibilities 184
3.4 Recent Developments 186
3.5 Conclusions 187
3.1 Introduction

The majority of the features of the molecular modelling package outlined in Chapter One are intended for the graphical analysis of results and do not require vast amounts of conventional computational power. However, graphical operations such as the rotation of a rendered image in real time can be computationally extremely intensive. More importantly, many of the algorithms which have been developed for structure and property prediction, including the Genetic Algorithm described in Chapter Two, require powerful processors if results are to be obtained in a reasonable amount of time.

Various alternatives may be considered to address the need for improved computational power. Two possibilities are the use of a RISC processor-based UNIX workstation, such as the IBM RS/6000 series or a Silicon Graphics Indigo or the modification of a standard IBM-compatible personal computer, both in terms of computational and graphics capabilities. Cards obtainable at the time from a variety of manufacturers claimed to boost the power of a PC to near mini-supercomputer performance at a fraction of the cost (approximately £3K for each board).

3.2 Enhancing A PC’s Processing Power

In order to gain real improvements for complicated computational techniques, two areas have been addressed by PC hardware manufacturers: computational power and graphics capability.
3.2.1 Available Processors

When designing add-on hardware for the PC environment, manufacturers are not limited to processors from the Intel 80x86 family (used in standard PCs) or their clones. Instead, they can choose the processor most suited to the task to be performed - i.e. one which has specialised features built into the processor’s design and firmware (internal instruction set). A variety of chips are available which address the features previously mentioned including the Intel i860, the Texas TMS34010 and its successor the TMS34020.

3.2.1.1 The Intel i860

Intel (1989) claimed that the release of their 64-bit i860 (80860) microprocessor in June, 1989, brought supercomputing to the desktop. This CPU was Intel’s first RISC (Reduced Instruction Set Chip) processor - a major departure from their previous designs. This prevented it from being completely compatible with the 80x86 family. It was the first processor to contain over 1 million transistors and could run at up to 50MHz.

The major difference between the i860 and other RISC processors available at the time was that all the processor’s functionality was embedded onto one piece of silicon. The SPARC chip set, found in the Sun SPARCStation, used five chips and Motorola’s 88000 chip set used three. An obvious benefit of this is the reduced space and power requirements of the i860.

The i860 included a RISC integer core, a pipelined floating point unit with parallel adder and multiplier units, 3D-graphics, memory management and separate instruction and data caches. With the pipelined architecture running at optimum speed, the three maths units can produce one new result each per clock cycle - a theoretical 150 million results per second for a 50MHz processor. Another
advantage is the speed of data transfer between the various components of the processor. When the system's memory and processor are physically separated, the internal clock speed of the CPU is invariably faster than the speed with which data can be transferred across the data bus, causing the CPU to wait for information.

Fried (1991) produced a series of comparisons between the i860 and 80x86 processors with coprocessor chips produced by Intel (80x87) and Weitek (x167) where appropriate. Even though the varying clock speeds of the compared processors make direct comparisons difficult, it is clear that in scalar mode the i860 is considerably faster than the 80x86 processors. The improvement in performance against the Weitek 4167 coprocessor is not considerable, especially for single-precision calculations, because scalar operations are not repetitive and therefore cannot be pipelined. A selection of the results are summarised in Table 3.1.

**Table 3.1 - Comparison of performance for Intel (80x86/80x87, i860) & Weitek (x167) processors performing scalar calculations. Results are given as 'Whtescales' in MFLOPS.**

<table>
<thead>
<tr>
<th></th>
<th>80387DX 33MHz</th>
<th>3167 33MHz</th>
<th>80486 25MHz</th>
<th>4167 25MHz</th>
<th>i860 33MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Precision</td>
<td>1.61</td>
<td>4.05</td>
<td>3.31</td>
<td>9.95</td>
<td>12.36</td>
</tr>
<tr>
<td>Double-Precision</td>
<td>1.43</td>
<td>3.57</td>
<td>2.94</td>
<td>7.71</td>
<td>12.36</td>
</tr>
</tbody>
</table>

However, when vector operations such as matrix multiplications are considered, the i860's relative performance is increased, especially for double-precision calculations, even though the processor is still operating in scalar mode. The results are summarised in Table 3.2.
Table 3.2 - Comparison of performance for Intel (80x86/80x87, i860) & Weitek (x167) processors performing vector calculations in scalar mode. Results are given as 'Whetmats' in MFLOPS.

<table>
<thead>
<tr>
<th></th>
<th>80387DX 33MHz</th>
<th>3167 33MHz</th>
<th>80486 25MHz</th>
<th>4167 25MHz</th>
<th>i860 33MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Precision</td>
<td>0.866</td>
<td>2.56</td>
<td>1.87</td>
<td>4.55</td>
<td>5.88</td>
</tr>
<tr>
<td>Double-Precision</td>
<td>0.672</td>
<td>1.12</td>
<td>1.70</td>
<td>1.93</td>
<td>4.91</td>
</tr>
</tbody>
</table>

The true potential of the i860 is revealed with a calculation such as a hand-coded matrix multiplication, incorporating pipelining and dual-instruction mode (whereby pipelined loads and pipelined computations are carried out simultaneously). When this type of routine is benchmarked, the results can approach the theoretical maximum of 66 MFLOPS (at 33MHz). Of course, it is unrealistic to hand-code large routines unless large performance gains are to be expected. The results obtained from a vectorising compiler are unlikely to result in such high gains in efficiency.

Since Intel designed the i860 to act as a “Cray-on-a-Chip” (Hayes, 1989) incorporating all of the functionality of a graphics workstation or a mini-supercomputer, it is obviously of great importance that a high standard of 3D-graphics capability is included in the processor. Although this functionality only takes up around 3% of the chip’s surface area, it does incorporate algorithms for Phong and Gouraud shading and z-buffering. These features can be considered essential if the user is trying to visualise a 3-dimensional object on a 2-dimensional screen.

The design of the i860 made it a multi-purpose processor, suitable for a wide variety of applications. Whilst one intention was its use in single processor
workstations, it could work in parallel with other i860s, making it ideal for use in multi-processor systems such as hypercubes. The ability to communicate with 80386 processors (and higher), meant that it was also ideally suited for use as a coprocessor card for PCs, or as a graphics accelerator card.

3.2.1.2 The Texas Instruments TMS34020
The 32-bit TMS34020 microprocessor from Texas Instruments was released in 1990. This chip was different from the Intel i860 in that, although it could be configured as a stand-alone processor, it was really developed to act as a graphics engine which would operate alongside a host CPU. The processor was the second generation of the TMS340 series of chips, being up to 20 times faster than its predecessor, the TMS34010 at some operations.

The TMS34020 incorporated hardware support for curve drawing algorithms and raster graphics operations such as PixBlts, along with the general purpose instructions found on most processors. The chip could also be extended through its coprocessor interface, which gave access to the optional TMS34082 floating point unit, allowing faster calculations.

One important feature of both the TMS34020 and its predecessor was their compliance to the TIGA (Texas Instruments Graphics Architecture) programming interface. This standard graphics interface had many advantages to offer:

- TIGA enabled graphics tasks in an application to run on the graphics processor in parallel with instructions running on the host CPU, enhancing speed.

- The interface provided a set of graphics primitives compatible with Microsoft’s implementation of the C programming language, making application development easier for software engineers.
- Use of TIGA allowed hardware independence since the program could determine the architecture of the graphics subsystem and adapt itself accordingly.

- Older applications written to the TIGA standard automatically operate more efficiently on newer TIGA compliant hardware providing a clear upgrade path.

Of course, if the functionality required by a particular application was not already embodied within the TIGA libraries, the developer was still able to develop their own custom graphics routines. The TMS340 family of processors has been incorporated into a variety of PC graphics boards.

### 3.2.2 Enhancing Computational Power

#### 3.2.2.1 Coprocessors

One of the easiest methods of upgrading mathematical capabilities was the addition of a coprocessor. Intel and its competitors (e.g. Cyrix, IIT and Weitek) all produced maths coprocessor chips for PCs. Whilst many of the clone chips provided little improvement over the Intel variants, a exception was seen with the Weitek xl67 family, albeit at a considerably higher price. Performance figures for these processors compared to Intel’s maths coprocessors can be seen in Tables 3.1 and 3.2 (pages 169 and 170).

Upgrade processors have also been released which take, for example, a 386 to a 486 equivalent, or a 486 to a Pentium equivalent. The chips produced by Intel are known as Overdrive processors.
3.2.2.2 Multi- and Parallel Processing Systems

One obvious solution to a lack of computational power is to use multiple processors. For the purposes of this discussion, systems which utilise multiple instances of the 80x86 processor family will be considered. Solutions which use one or more alternative processors (namely transputers or i860 chips), as well as an 80x86, will be addressed in the following sections.

It is important to make the distinction between multiprocessing and parallel processing. Whilst both systems have multiple processors, a multiprocessing machine runs several discrete processes at the same time. A parallel processing system divides the task into many parts - each running on a separate processor. The individual results are then combined. In general terms, more significant speed increases can be obtained by using a parallel processing machine.

Systems which contain multiple processors are obviously more complex than single processor systems. Memory can be allocated to processors in two ways - shared or distributed. In general, multiprocessing machines commonly use shared memory, and parallel processing machines distributed memory. Inter-processor communications were addressed by Corollary, who developed C-bus, a bus and cache architecture for linking multiple 386 and 486 processors.

Scalability is also important: an ideal system is linearly scaleable - 50% more processors give 50% more performance. Shared memory systems are scaleable for small numbers of processors (10-20). Beyond this, the shared bus between the processors and the single block of memory becomes overloaded and performance/processor decreases. Because distributed memory systems do not rely so heavily upon the bus they can theoretically include hundreds or even thousands of processors.
An example of a commercial multiprocessing system is Advanced Logic Research (ALR)'s MultiAccess Series 3000 UNIX machine, reviewed by Unger (1991). This machine could accommodate up to six Intel 486 processors (25 or 33MHz), each with a 256Kb cache. A 33MHz 386 processor sits on the SCSI controller, dealing both with the applications running on the system and with file I/O requests. Separate boards, each holding up to 16Mb of Ram make up the shared memory. Communication is carried out via the 16MHz C-Bus. The top of the range MultiAccess system could serve up to 160 users at any one time.

In a parallel processing system, the connections between the CPUs are of paramount performance since the processors must communicate with each other in order to complete the task. Ideally, each processor should be connected to every other processor - no information would have to pass through an intermediate node before reaching its destination. Unfortunately, this is difficult for all but the smallest of distributed memory systems.

Proposed connection architectures include rings and 2-dimensional arrays of processors, where each processor is connected to all of its neighbours. The drawback of these systems is that as processors are added the number of nodes a message passes through before reaching its destination can increase dramatically.

Probably the most successful distributed-memory systems are based upon the hypercube topology, first demonstrated in the California Institute of Technology's Cosmic Cube. In a hypercube, $2^n$ processors and their associated memory are arranged in an n-dimensional cube. Examples of a 3-dimensional hypercube, which has 8 processors arranged as a simple cube, and a 4-dimensional hypercube with 16 processors connected as a tesseract are shown in Figure 3.1.
Each processor in a hypercube is connected to $n$ other processors and the maximum path length (or communications diameter) is $n$ nodes. One of the major advantages of the hypercube system is its scalability - an 8-dimensional hypercube consists of 256 processors with a maximum path length of only 8 nodes.

In the field of chemical research, many programs have been developed or modified to run on parallel machines. In the area of molecular mechanics, two research groups have developed parallel implementations of the AMBER modelling package (Vincent, 1995; Swanson, 1995). The reason why this package has been chosen is because AMBER is parameterised primarily for biological molecules such as proteins and calculations on these large structures can be computationally intensive. The programs have been developed with portability in mind. Both groups show impressive gains in computation speed for parallel supercomputers (e.g. CRAY T3D) and for clusters of networked UNIX workstations.

White (1996) decided, given the performance-price ratio of current mass-produced hardware, that proprietary nodes in a parallel computer were no longer cost effective. Instead, his Parallel PC consisted of PCs based upon processors from...
Intel's 80x86 family, (486, Pentium or Pentium Pro). The reported architecture consisted of a host 486 PC connected to eight node 486 PCs by a dedicated 20-Mbits/second Inmos serial link. Each of the nodes was also connected to every other node, using the same link. This is shown schematically in Figure 3.2.

The design could be implemented either using dedicated PCs or in a general computing laboratory with the addition of the Inmos Serial cabling. White believes that this type of architecture is scaleable up to 32-nodes; beyond this the complete interconnectivity of the nodes would be too difficult. However, clusters of 32 fully interconnected nodes could be joined to each other through sparser links.

The molecular mechanics algorithm implemented upon this system was an in-house Newton-Raphson minimiser, (VULCAN), which evolved from previous programs developed for a variety of systems including an OCCAM version developed for parallel processing using transputers. The program was compiled using Microsoft Fortran Powerstation. Molecules of any size can be treated, subject to the limitations of available memory.
Communication between the processors on the parallel machine was handled through a message passing library (COMFORT) developed originally in-house for transputer-based systems. The VULCAN program was parallelised by allocating 'slices' of atoms to the host and nodes - each processor could effectively then run the same code during the calculation. COMFORT provided a means for allocating appropriate slices of atoms, dependent upon the host or node CPU’s power.

Table 3.3 illustrates performance figures for a Parallel PC using a 100MHz AMD 486 processor with 16Mb RAM as the host and 80MHz AMD 486 processors with 4Mb RAM each as the nodes.

Table 3.3 - Energy minimisation run times and parallel efficiencies for 250-iteration calculations for various Parallel PC configurations.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th><strong>Crambin (327 atoms)</strong></th>
<th><strong>Ubiquitin (602 atoms)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (sec)</td>
<td>Efficiency (%)</td>
</tr>
<tr>
<td>Host</td>
<td>788.73</td>
<td>100.0</td>
</tr>
<tr>
<td>Host + 1 Node</td>
<td>447.26</td>
<td>97.97</td>
</tr>
<tr>
<td>Host + 2 Nodes</td>
<td>316.70</td>
<td>95.78</td>
</tr>
<tr>
<td>Host + 4 Nodes</td>
<td>204.38</td>
<td>91.88</td>
</tr>
<tr>
<td>Host + 6 Nodes</td>
<td>155.00</td>
<td>87.73</td>
</tr>
<tr>
<td>Host + 8 Nodes</td>
<td>123.51</td>
<td>86.30</td>
</tr>
</tbody>
</table>

Parallel efficiency figures are used in preference because the host and nodes differ in computational power. The decreased efficiency as more nodes are added is due to the increased communication time as data is transferred between the nodes and the host. The Parallel PC is more efficient for larger molecules, since the communication times are smaller relative to the time the overall calculation takes. For this reason, parallel efficiency falls to 50% for a 27-atom molecule and 8 processors. Parallel efficiency is calculated as shown in Equation 3.1:
Effective No of Processors = 1 + \[
\left[ \frac{\text{No of Nodes} \times \text{Node CPU Speed}}{\text{Host CPU Speed}} \right]
\]

Parallel Efficiency = \[
\left( \frac{\text{Time of Run on Host}}{\text{Time of Run} \times \text{Effective No of Processors}} \right) \times 100
\]

3.1

The parallel efficiency figures quoted are higher than those for similar calculations on other machines. White attributes this to a variety of reasons concerning the design and operation of the Parallel PC (e.g. single tasking, so no task swapping overhead; broadcasting is handled by hardware rather than software, etc.).

3.2.2.3 Transputer Boards

Transputers were designed to be connected together as the heart of a parallel processing system. Communications between the different types of transputer was designed to operate transparently to the user. A typical Inmos transputer processor was the T800 - a 32-bit RISC processor running at 20MHz with 4Kb of on chip RAM and a 64-bit floating point unit. At the time of its introduction, this processor was easily the fastest readily available CPU, having a Whetstone benchmark almost 30% higher than a 16MHz Intel 80386 with a Weitek coprocessor and 340% higher than a 16MHz 80386 with an Intel (80387) coprocessor.

A commercial company with a history of high performance computation engines developed for PCs is MicroWay. Towards the end of the 1980s, their main products in this area were based upon transputer technology. Two boards were available: the MonoPuter with one Inmos transputer, or the QuadPuter with either two or four. A third product, the VideoPuter, was a transputer based graphics card, which enabled the production of high resolution graphics at high speeds in parallel systems.
The MonoPuter could be fitted with between one and sixteen megabytes of dedicated memory. The transputer operated at 10 MIPS, and the built-in floating point unit at 1.5 MFLOPS - the board provided additional computing power equivalent to an 80386 processor with a Weitek coprocessor. By using a network of MonoPuter boards, multiple tasks could be run in parallel.

The QuadPuter evolved from the MonoPuter's design. When launched, this board allowed up to 4Mb of memory per chip, and increased processing speed by an additional two to three times over the MonoPuter. Once again, the boards could be run in parallel, and up to eight QuadPuters (32 transputers) could be linked using MicroWay's LinkPuter board.

Programming languages and software tools available for these boards included Parallel C, Parallel Fortran and Parallel Pascal, a window-based debugger and a ROM configurer. A variety of tools were supplied with the compilers which, for example, loaded the executable files onto the transputer(s) and allowed the allocation of software tasks to specific transputers. The Flood-Filling configurer allowed even more flexibility by automatically routing work packets generated by a 'master' task to any free processor, and carrying back result packets.

White and co-workers, (White 1989, 1996), have performed extensive research on the use of relatively low cost parallelised systems for molecular modelling and, in particular, molecular mechanics calculations. Initial work concentrated upon various transputer based systems used IBM or compatible PCs containing an 8088 or 80386 processor as the host system.

Versions of their molecular mechanics program were written in OCCAM and Fortran 77. Even running on a single transputer, the calculations were around 45%
faster than corresponding programs run on typical minicomputers available at the
time, such as DEC's VAX-11/785/FPA. The program attained a parallel efficiency
of between 80 and 90% for up to 25 transputers, and could handle molecules of up
to 12,000 atoms with 4Mb of memory per transputer. Beyond 25 transputers,
however, parallel efficiency decreases because of the relative increase in
communication time between the nodes, in the same way as was experienced with
the Parallel PC.

White also describes techniques for developing and automatically optimising force
field parameters using a transputer based system. As a result of migrating the
program from a minicomputer to a single 20MHz T800 transputer, the program
ran 3-4 times faster, reducing force field development time to approximately one
month. The technique has been described in more detail in Chapter Two.

3.2.2.4 MicroWay i860 Boards
A natural progression of MicroWay's transputer boards was the development of
equivalent cards which used more powerful processors, namely Intel i860s of
varying speeds (33, 40 or 50MHz, depending upon board chosen). The two
variants available were the Number Smasher-860, with one CPU, and the
QuadPuter-860, with four; both cards used the EISA bus architecture.

The single i860 board came with 8Mb of dedicated RAM, upgradable to 32Mb -
programs were executed by downloading the code from the PC. The card was not
intended for use as a parallel processing system, so there was no need to allocate
tasks between the i860 and the 80x86 host processor. The results of using a
40MHz i860 Number Smasher with 32Mb RAM to run a genetic algorithm have
been previously described in Chapter Two.
The QuadPuter-860 differed from its transputer equivalent in that, in addition to 2Mb of local RAM for each processor, the board also had 32Mb of shared memory. The board had approximately the same throughput as a Cray 1, with only a fraction of the power requirements (15 Watts compared to 150 Kilowatts). It was also possible to have several QuadPuters running in the same system - five boards operating together could operate at one GFLOP, whilst consuming just 75 Watts.

Both boards could be programmed using either native i860 compilers or cross compilers from the 80x86. Programming languages available include Fortran 77, C/C++ and Pascal. The Fortran compiler's performance could be increased through the use of the VAST-2 vectoriser from Pacific Sierra Research Corporation and Intel's i860 Vector Primitive Library. The vectoriser converts Fortran loops into a series of calls to the i860 vector primitive library, which is coded to use the i860's pipelined floating point unit. Use of this tool could improve the performance of vector code by 50-100%, since vector operations run much faster on the i860 than scalar operations. To aid the porting of mainframe applications, many of which have historically been written in Fortran, the Fortran compiler was also 98% VAX/VMS compatible. The C/C++ and Pascal compilers both complied with the ANSI standards for these languages.

3.2.3 Enhancing Graphics Capability
The standard graphics capabilities of early PCs was low. As time has progressed, users' expectations have increased and all PCs are now shipped with at least 256 colours at 640 x 480 pixels (VGA) resolution. For improved general performance, a number of replacement cards from a variety of manufacturers are available at relatively low cost.
3.2.3.1 High Performance Graphics Cards

If graphics performance is of great importance, highly sophisticated graphics hardware can be installed. The important difference between these and standard cards is their ‘intelligence’ - these boards are built around processors developed specifically for graphics processing. The chips have special instruction sets enabling, for example, fast vector operations and pixel manipulations. When this work commenced, a variety of cards of this type were available from a range of manufacturers, with varying degrees of sophistication. The simplest of these high performance cards were based upon a single graphics processor, such as those in the TMS340 series from Texas Instruments.

Several hardware manufacturers realised that even higher graphics performance could be obtained by combining the graphics abilities of the TMS340 family with the number-crunching capabilities of Intel’s i860 processor. One example of this type was Spea’s FGA 860GX - a two-card set comprising the TIGA compliant Future Graphic Adapter (FGA) and a full sized coprocessor board - the 860GX.

The FGA card, based upon a TMS34020 processor, was marketed as a stand-alone graphics card, offering 2Mb of video memory and up to 4Mb of programmable RAM. VGA support was included on the card with a chipset from Headland Technologies. Two video connectors were provided on the card allowing VGA output to be displayed either on a separate monitor, or on the main monitor, either full screen or in a window. The maximum possible resolution from the system was 1280x1024 pixels, displaying up to 256 colours from 16.7 million.

To provide additional power, two options were available. The simplest was the addition of a TMS34082 coprocessor board to handle 3D TIGA calculations. This daughter board plugged directly into a connector on the FGA card.
For even more power, a full-size board, the Spea 860GX, could be connected instead of the coprocessor system. The 860GX board incorporated an i860 running at 33 or 40MHz and up to 32Mb of programmable memory. The board was designed as a general purpose ‘supercomputing’ board, able to perform computationally intensive tasks such as ray tracing along with graphics processing. This was possible through the board’s support for Intel’s Applications Processor Executive (APX) standard. The complete Spea FGA 860GX system is shown in Figure 3.3.

**Figure 3.3 - Spea’s FGA 860GX 3D Graphics Subsystem.**

Software support for the FGA series came in terms of standard TIGA compatibility, Windows 3.x drivers and drivers for CAD/Rendering packages. Since the TIGA library was unable to utilise the i860’s capabilities, Spea also supplied libraries of 2D and 3D graphics functions specifically for the FGA 860GX two card set.
3.3 PC Hardware Incompatibilities

One important area which must be considered when upgrading a computer is the compatibility of the new hardware with the existing system. The computing industry has addressed this by formulating a variety of standards for PC architecture and as time has passed incompatibilities have become fewer. One area which can still cause difficulties is the physical dimensions of many of the components one may wish to install.

This is especially prevalent when one is installing specialist hardware. Initial difficulties were experienced when installing the Spea FGA 860GX two card set into an Elonex 486DX 50MHz computer with a large desktop case. The first problem concerned the physical size of the cards - their height meant that it was impossible to replace the PC’s case, despite its larger size.

A more fundamental problem arose due to the direct connection between the two cards. The drive to reduce the footprint of PCs meant that the expansion slots on the motherboard were closer together than usual. In most cases this would not cause a problem, since most cards are inserted as a single entity. However the two cards in the FGA 860GX used a rigid connector and the decreased separation at their base prevented contact from being made between the two boards. This is illustrated in Figure 3.4.
Figure 3.4 - Diagram illustrating (i) The Spea FGA 860GX card set and (ii) the separation caused by installing the system in the Elonex PC.

The symptom of this problem was a complete lack of response from the 860GX (i860 based computational engine) board. The cause of the fault was extremely difficult to diagnose - once diagnosed the cards were installed in a machine with a wide tower case.

This lack of standardisation in motherboard design causes other problems. It was impossible to install a new motherboard into Elonex desktop machines because of their reduced size. The problem can still arise today: some motherboard are impossible to install in some cases as the height of components on the motherboard (especially processors with heatsinks and fans mounted on them) can be too great to fit under drive bays, etc.
3.4 Recent Developments

It would have been difficult to predict the massive advances in computational power which have occurred for the PC platform since this project commenced. The major advances have been in terms of processor power - Intel have continued to enhance the 80x86 family of processors - increasing computational power, graphics capability (MMX) and clock speed.

The latest advances in graphics hardware and software - mainly introduced for the computer games market, have enabled the production of workstation quality images on a desktop PC (James, 1997a).

Other areas of PC architecture which have seen dramatic improvements include motherboard design (improved bus speeds and data path widths), memory and hard disks (vastly increased capacities at reduced prices) and the widespread uptake of powerful 32 bit operating systems (Windows 95 and NT).

On the UNIX workstation front, the cost of entry level machines such as the Silicon Graphics Indy has fallen to prices similar to those of high specification PCs although upgrade costs are still considerably higher. Such a machine, however, will typically have a higher system bus speed and a more efficient RISC processor than the PC.
3.5 Conclusions

When this work commenced, a key aim of the project was to investigate the enhancements that could be made to an IBM compatible PC in order to approach the power of a UNIX graphics workstation. The problems encountered due to the incompatibilities between the personal computer and the chosen graphics hardware meant that it was difficult to progress work in this area and the direction of the project therefore changed towards the Genetic Algorithm investigations described in Chapter 2. However, given the changes which have occurred since this work commenced, the question can still be posed, is it worth upgrading a PC in an attempt to emulate a UNIX workstation? In terms of graphics performance, the answer is probably no, unless extremely high quality rendered images must be manipulated in real time. Applications which take advantage of MMX technology and even a medium specification standard graphics card are probably perfectly adequate for the visualisation required in chemistry.

Computational ability, however, may still be a different matter. There are many predictive codes currently available for chemists, and more are always being developed. Many of these codes require high performance computers (i.e. remote supercomputers) in order to return results in a realistic time scale, even though these machines are often performing calculations on a variety of problems at any one time. There is also considerable interest in using simpler techniques such as molecular mechanics on more complicated systems such as proteins - again requiring an increase in computational power. Unfortunately, supercomputing facilities are still prohibitively expensive to many researchers, so there should be an advantage in trying to implement these codes on local machines. However, the use of add-on cards such as the transputer and i860 cards described previously seems to be declining - indeed Intel have ceased manufacture of i860 chips,
concentrating instead on the 80x86 processors. The way forward, therefore, would seem to be to use the multi-processing abilities built into the latest members of this family - currently running at speeds in excess of 400MHz, either as a set of processors in the same machine (on the same motherboard or each processor on a separate card), or by linking a series of discrete machines together as described by White (1996). By splitting the task between a series of high performance PC processors, with the whole unit dedicated to performing one overall task, it should be possible to approach supercomputing abilities at a fraction of the cost.
DESIRABLE FEATURES IN A CRYSTAL MODELLING PACKAGE

4.1 Introduction

4.1.1 Ligand Browser 190
4.1.2 CSD On-line Link 190
4.1.3 Symmetry Operators Overlay 203
4.1.4 Manipulation of Symmetry Related Fragments 206
4.1.5 Interactive Modification of Unit Cell Parameters 206
4.1.6 Symmetry Operator Replacement 206
4.1.7 Molecular Volume Calculations 207
4.1.8 Lattice Slicing Algorithm 210
4.1.9 Single Crystal X-ray Diffraction Pattern 210
4.1.10 Ligand MO/Electrostatic Calculations 210
4.1.11 Structure & Property Prediction Algorithms 210
4.1.12 Partial Display of Structure 211
4.1.13 Data File Input/Output 211

4.2 Conclusions 212
4.1 Introduction

The Genetic Algorithm described in Chapter Two is only one of many tools a crystal engineer would require to facilitate the prediction of the structure and properties of new materials. Initially, a reasonable starting structure must be assembled and optimised. Once the molecular unit has been generated, various visualisation and monitoring utilities are required to propose how it will pack into a crystal structure. Whilst the following list is by no means exhaustive, some of these features are discussed below.

4.1.1 Ligand Browser

A ligand browser would act as a library of structural information, containing a summary of the ligand information contained in the CSD, indexed according to the ligand’s donor characteristics. At the top level, screens of 2-dimensional ligand structures would be searched according to the classifications already outlined (i.e. denticity, donor atom(s) etc.) or on other information such as preferred metal centres. Once selected, a rotatable 3-dimensional representation of the ligand would be displayed, along with summary information such as preferred metals, conformations (cis or trans) etc. The user would then select their ligand(s) of choice, and use them to build up their initial structure for further refinement by the predictive algorithms. The program would act solely as a catalogue - without requirements for intensive computational power, allowing implementation on a lower powered machine, such as an IBM compatible PC.

In order to implement such a program, two important issues are raised;

i) How is the structural information about the ligands obtained and stored?

ii) How can this information be best displayed on the screen?
Ideally one would be able to add new ligands to the package without having to rebuild the software or, better still, without the need for any human intervention.

**i) Obtaining and Storing the Structural Information**

Wright, (1993, 1994) has developed software which transforms the CSD’s structural information into a new data format designed specifically for coordination compounds. The new data format treats the metal centre of the complex as the focal point, with the remainder of the connectivity table separated into ligand subsets. It also contains a geometrical description of the spatial arrangement of the ligands around the metal centre and the CSD reference code (refcode) providing a means to cross-reference the information back to the main database should the need arise.

The program only deals with four to six coordinate mononuclear complexes of the transition, lanthanide and actinide block metals with mono, bi or tridentate ligands - however this accounts for the majority of structures encountered in the database.

It is written in FORTRAN-77, running on a MicroVAX-QII computer under the VMS V5.4 operating system. The initial development work on the program occurred prior to the release of version 5 of the CSD system, which includes full matching of chemical and crystallographic connectivity representations in addition to 3D search screening. The program’s input is in the CSD’s FDAT format. This is the output of a QUEST search, allowing the user to decide which subset of the CSD will be converted. Once data files of complexes have been extracted to the new data format, the specific ligand data can be extracted for use in the browser program.
An alternative approach would be to shell out from the application to the CSD search software. The search would be carried out asynchronously to the main program - the results being left in a buffer for collection later.

**ii) Representing the Information on Screen**
There are three possibilities for representing the structure as a 2-dimensional image on the screen.

1) **Dynamic Structure Generation**
One possible method is to implement an algorithm which dynamically creates a 2-dimensional representation of the ligand from the 3-dimensional coordinates and connectivity information obtained from the CSD. This problem is more complex than one may at first think. Four characteristics have been identified which are of vital importance in the generation of good chemical structures from a connection table, (Shelley, 1983), namely;

   a) Ring systems, if present, must be perceived and assigned coordinates that depict the ring system in an easily recognisable format.

   b) Coordinates should be assigned to the acyclic atoms of the structure that minimise atom crowding and the number of overlapping bonds.

   c) Complete structures should be correctly oriented. For example, steroids with the following ring system are always drawn with the A ring on the left and below the D ring (Figure 4.1).
d) Similar structures should be oriented in the same way so that the chemist can quickly perceive differences and similarities in a list of structures.

Whilst a simple projection onto a plane may resolve some structures, more complicated structures stand little chance of being resolved satisfactorily. The display of even simple structures is highly dependent upon the orientation of the molecule to the plane of projection (Figure 4.2). Computer algorithms exist which will orient a molecule such that it is viewed from its most planar side, but even then ambiguity can exist when this structure is projected into 2 dimensions.

Various attempts have been made to address this problem. Some attempts (Zimmerman, 1971, Dittmar, 1977) use template structures for basic rings. This type of system produces good results for most compounds - the templates aiding the correct orientation of structures containing common ring systems.

Figure 4.1 - The standard display orientation for steroids.

Figure 4.2 - Projection of a molecule’s coordinates onto a plane may lead to complete resolution or total ambiguity in the 2-dimensional representation.
Unfortunately, templates were not developed for more unusual systems, so the programs’ outputs cannot be relied upon for more unusual molecules.

Other work (Carhart, 1975, 1976) focused on a non-template-based system for providing the user-interface of CONGEN, the constrained structure generator developed by the DENDRAL project at Stanford. CONGEN generates such a wide range of unusual cyclic structures that a small set of templates and rules could not be expected to cover them all. The program was developed for teletype output, and can lead to more confusing structures than a good projection may generate (Figure 4.3).

![Diagram of chemical structures]

**Figure 4.3 - Teletype output from Carhart's program, when non-bonded atoms can occupy adjacent grid squares.**

Carhart concludes “...the program is somewhat larger and more time-consuming than might be desired”. The program also ignores structure orientation and often assigns poor coordinates to ring systems. However, it has been important in the use and development of many programs in chemistry.

Two other procedures for displaying chemical structures without the use of a ring system dictionary have been developed (Cox, 1973, Zippel, 1982). Neither uses the same model-building technique as Carhart, but, as for Carhart, they cannot maintain similar orientations for similar structures or orient ring systems correctly.
Shelley’s system (1983) comes closest to fulfilling the four characteristics defined earlier. His program is flexible because it does not rely on templates, but it can still resolve complex structures such as the morphine fused ring system, which appears as in Figure 4.4.

![Figure 4.4 - The morphine fused ring system.](image)

However, the program does have some limitations:

a) The orientation of structures is independent of heteroatoms and multiple bonds. For example, pyridine is displayed with the nitrogen at any of the six vertices - the position dependent upon the connection table numbering scheme.

b) More seriously, the program is unable to generate good coordinates for some complex ring systems, so ring system A appears as B (Figure 4.5).
Figure 4.5 - A ring system and its 2D-representation generated by Shelley's algorithm

c) Much of the program’s I/O depends upon the host computer (Data General MV/8000), and the raster graphics terminal (Hewlett-Packard 2647A).

Dynamic structure generation in this context is therefore not feasible for the following reasons:

- If one performs a simple projection of the coordinates onto a 2-dimensional plane the resulting structure is often unrecognisable.

- Due to their graphical nature, existing algorithms have been developed for specific hardware systems. These programs would be difficult to convert to an IBM compatible PC, probably demanding a complete rewrite. All systems exhibit problems displaying some structures in a readily recognisable format and many of the algorithms are not freely available.
• The algorithms are extremely complicated - at least two PhD theses have been written on this topic alone.

• The algorithms are complex and computationally intensive.

An alternative approach is therefore needed - both alternatives require some form of manual data preparation before the ligand can be added to the browser.

2) Coordinate Data Files
Data files could be created which contain the coordinates of a 2-dimensional representation of each ligand. These files could be read into the browser at run-time, and the structures could be generated by the program. There is still the problem of the time taken to generate the structures before they can be displayed.

3) Store the Structures as Pictures
A more practical alternative is to manually draw a 2-dimensional representation of each ligand using a standard drawing package or one for chemical structure drawing. The datafile for each ligand would contain a reference to its picture. By using this approach, the program would not require rebuilding every time a new structure was added to the program - the only human being the generation of the data file and chemical structure. Of course, this approach would still require a chemist to identify which ligands were present in a particular compound.

4.1.2 CSD On-line Link
An on-line link to the Cambridge Structural Database (CSD) from a modelling package would allow direct searching and comparison of potentially useful structures. The ability to identify and extract ligands as fragments from the
database into a ligand library or browser is also a desirable feature as this could provide information about commonly occurring ligand-metal combinations and would aid in the construction of new molecules. This link would also be of benefit to a fully automated version of the GA application.

**The Cambridge Structural Database**

The CSD forms an invaluable store of structural information for over 160,000 compounds (as of April 1997), determined by X-ray and neutron diffraction (Allen 1979, 1991). Along with 3-dimensional coordinates, the CSD contains bibliographic and connectivity information for crystal structures containing up to ~500 atoms. The data is accessed through the provision of powerful search, analysis and display software (QUEST-3D and VISTA), available for both VMS and UNIX based systems.

Since 1965 there has been a dramatic increase in the number of compounds contained in the database - currently increasing at a rate of more than 10,000 new entries per year. Data quality is also excellent - over 89% of the data in the October 1992 data file were reported as having acceptable or better precision (R-factor < 10%) and, given the size of the data file, the statistics are unlikely to have changed appreciably. The CSD has therefore been adopted as a major structural resource in materials research, often with little or no requirement to refer to the original literature. Typical published examples of studies include:

- Tables of standard bond lengths for structural studies and in the assignment of equilibrium values in molecular mechanics force field parameterisation, updated recently for organics (Allen, 1987), and organometallics and coordination complexes (Orpen, 1989).

- Average molecular geometry (Sheldrick, 1980; Taylor, 1982)
- Conformational studies (Auf der Heyde, 1989a, 1989b, 1989c; Nøskov-Lauritsen, 1985)

It is therefore vital that there should be free exchange of information between the CSD and any crystal modelling package.

Limitations for Modelling Coordination Compounds

In coordination chemistry, a structure is considered as a number of components, namely metal centre(s) surrounded by coordinating ligands. The ligands can be classified in a variety of different ways (Cotton, 1980):

a) According to the way they donate electrons to the metal centre. For example, ligands may be classical electron pair donors, forming complexes with all types of Lewis acids, metal ions or molecules, or \( \pi \)-bonding ligands with both electron donor and acceptor orbitals, which form compounds largely with transition metal atoms.

b) Electronically, according to the number of electrons they when the ligands are regarded as neutral species: One-electron donors form a single covalent bond, (e.g. Cl or \( \text{CH}_3 \)). Two-electron donors have an electron pair, (e.g. NH\(_3\) or H\(_2\)O\(^-\)). The ethanoate ion can be a one- or three-electron donor (Figure 4.6).

![Figure 4.6 - The ethanoate ion: A one- or three-electron donor.](image-url)
This method is useful as it is an aid in electron counting - particularly in transition metal complexes which follow the eighteen-electron rule.

c) Structurally, according to the *denticity* of the ligand. Bidentate ligands bound by both donor atoms to one metal centre are termed *chelates*. Another important feature of ligands is as *bridging groups* - where one ligand bonds to two or more different metal atoms. Examples of this phenomenon are shown in Figure 4.7.

![Figure 4.7 - Examples of bridging ligands, taken from Cotton (1980), page 64](image)

d) According to the nature of the donor atom, e.g. oxygen, nitrogen, sulphur etc. In this classification *ambidentate* ligands should also be considered (Balghura, 1976; Norbury, 1970 and 1975; Burmeister, 1966). These are unidentate ligands with more than one donor site, giving rise to the possibility of *linkage isomerism*. Typical examples of these ligands are:

```
{ M-NO₂  Nitro
  M-ONO  Nitrito
  M-SCN  S-Thiocyanato
  M-NCS  N-Thiocyanato
  M-CN   Cyano
  M-NC   Isocyno
```

However, it is difficult to search the CSD on criteria such as these. Indeed, the very concept of a ‘ligand’ is only present in the CSD in a very limited number of chemical classes (e.g. chemical class 76 contains complexes of ethylenediamine).
Very few structures of uncoordinated ligands are available and those which are present often display geometries which are far removed from those found in a metal complex. Whilst this difference is not necessarily large for monodentate ligands, it is readily displayed for ligands of bi- or greater denticity. For example, if the crystal structure of ethylenediamine is extracted (refcode ETDIAM10), from the CSD it can be seen that the unbound ligand is linear (i.e. it has an N-C-C-N torsion angle of 180°, Figure 4.8).

![Figure 4.8 - Free ethylenediamine](image)

![Figure 4.9 - Coordinated ethylenediamine](image)

On searching the CSD for the same ligand when it is coordinated to chromium (Figure 4.9), 154 occurrences are found in 54 structures. Plotting frequency of occurrence against the same torsion angle for these complexes, gives the distribution shown in Figure 4.10.

![Figure 4.10 - Distribution of N-C-C-N torsion angles in chromium complexes containing ethylenediamine](image)
As can be seen, the torsion angles for these complexes lie in the range -62° to +63°, with the majority of occurrences centred around -52° and +52°. These all relate to the ligand when it is donating both nitrogens to the same chromium atom - obviously this structure is far removed from that of the free ligand. Two occurrences of the fragment have torsion angles of 177° and 179°, which approximates to the unbound structure. In both of these cases, the ligand is only bound to the chromium in a monodentate fashion.

A further problem with the CSD for chemists who are interested in metal complexes is that the geometric nature of the metal centre and the stereochemistry of the ligands around it are not explicitly stored; this information only being accessible from the atomic coordinates. A search query which can differentiate between cis- and trans-isomerism in metal complexes has only recently become available with the introduction of 3D search tests in QUEST-3D.

More general searches inevitably result in the generation of much unwanted information, leading to the need for manual intervention to extract the information of specific interest.

To summarise, whilst the CSD contains most structural information a crystal engineer could require, it often requires a great deal of effort to extract the relevant information into a form which is useful. There is currently no other easily accessible repository of information about the ligands encountered in organometallic/complex chemistry. This is holding back the development of new computational tools for studying this type of molecule. What is needed to allow the CSD to be used to predict molecular structure is a reordering of the data, making it more accessible to the molecular modeller.
4.1.3 Symmetry Operators Overlay

A common difficulty experienced by chemists studying crystal structures is the visualisation of the symmetry present in the molecule or unit cell. Computer graphics is a great aid to the study of objects in 3 dimensions - however even this tool may not make a molecule’s inherent symmetry immediately obvious.

In the International Tables for Crystallography (Hahn, 1983) an attempt is made to show the symmetry elements present in a space group by showing projections of the unit cell down each of the cell axes. Figure 4.11 shows the entry for the monoclinic space group P2₁/c (International Tables No. 14), with unique axis b, cell choice 1. Whilst this gives some idea of the symmetry present in the space group it is still difficult to visualise how these elements interact and how they act upon the asymmetric unit.

![Symmetry Operators Overlay](image)

*Figure 4.11 - Entry in International Tables for Crystallography for the monoclinic space group P2₁/c*
A solution is to overlay a semi-transparent representation of some or all of the symmetry operators present in the crystal over the unit cell. This, coupled with the use of colour, would enable the modeller to visualise the symmetry relationships within the structure.

For example if 2-Bromobenzo(b)indeno(1,2-e)pyran, C_{16}H_{9}BrO (Figure 4.12), which crystallises in the P2_{1}/c space group is considered:

![Figure 4.12 - 2-Bromobenzo(b)indeno(1,2-e)pyran, C_{16}H_{9}BrO. Hydrogen atoms have been removed for clarity.](image)

It is known that this space group has the following symmetry associated with it:

**Symmetry Elements (within Unit Cell)**

- Centre of Symmetry at 0, 0, 0
- Screw Axis at 0, y, \(\frac{1}{4}\)
- Glide Plane at x, \(\frac{1}{4}\), z

These give rise to the four general equivalent positions:

- \(x, y, z\)
- \(-x, -y, -z\)
- \(-x, \frac{1}{4}+y, \frac{1}{2}-z\)
- \(x, \frac{1}{2}-y, \frac{1}{2}+z\)

When the compound’s entry in the CSD is retrieved (Refcode = BBINPY) it is found that it has unit cell parameters \(a = 7.508\text{Å},\ b = 5.959\text{Å},\ c = 26.172\text{Å},\ \alpha = \gamma = 90^\circ,\ \beta = 92.55^\circ\). The asymmetric unit contains 26 atoms and there are four formula units in the unit cell (\(Z = 4\)). The coordinates of the asymmetric unit lie outside the unit cell (Figure 4.13).
Figure 4.13 - Graphical representation of symmetry elements associated with space group $P2_1/c$. 
4.1.4 Manipulation of Symmetry Related Fragments

Once there is an understanding of the symmetry present in a crystal, small modifications to the asymmetric unit - which may have important ramifications on the internal structure of the unit cell, can then be considered to see how they affect the overall structure.

Whilst engineering a crystal one may make seemingly minor changes to the chemical structure of the crystal - minor modifications to a ligand or changing the metal centre or counter ion. This may introduce unfavourable contacts in the structure, which could be relieved by making small rotations or translations to fragments in the asymmetric unit, whilst still retaining the structure's original symmetry. The same principle, of course, also works in the reverse case, where one may be searching for favourable contacts.

4.1.5 Interactive Modification of Unit Cell Parameters

It should be possible to modify the unit cell dimensions and interactively monitor the resultant changes in the lattice energy. This is of particular significance when it is necessary to assess the changes in packing which can arise as a result of raising or, more importantly, lowering the temperature.

4.1.6 Symmetry Operator Replacement

The interactive replacement of symmetry elements (e.g. a 2-fold rotation axis with a 2₁ screw axis) would enable the investigation of further modifications to the structure. It has often been found that small modifications to a structure can result...
in the relaxation of the symmetry to another space group which is a sub-group of the former structure. The interactive visualisation of the changes which occur during this process would allow an assessment of whether a new mode of packing is possible.

These features are invaluable in the study of the different packing found in crystal structures. One such application is in the study of polymorphism - an area of crystal chemistry which is of great industrial importance. Different polymorphs of the same compound exhibit identical properties once they are in solution, liquid or gaseous form, but they may have extremely different properties in the solid state.

4.1.7 Molecular Volume Calculations

At the most fundamental level a molecule’s properties are related to its size and shape - for example, a drug molecule must be able to fit into a receptive site. Even though these interactions most commonly occur in solution or in the gaseous phase where the dynamic motion of the entities leads to constantly changing shapes, much useful information may be obtained by looking at a ‘snapshot’ of the molecule - either in solvated form, modelled in vacuum or even as the crystal structure.

The calculation of the molecular volume of all or part of a unit cell has been the subject of studies by a variety of workers. The molecular volume gives information on the packing coefficient for a compound and allows one to quantitatively assess the possibility of replacing structural units (metal atoms, cations etc.) with other units of a similar size in an attempt to modify the crystal’s structure and properties.
Non-polar solids, especially hydrocarbons, adopt packing arrangements based solely upon van der Waals' forces. These structures can be explained using Kitaigorskii's close-packing principle (Kitaigorskii, 1973). This packing coefficient corresponds to the efficiency of packing in the unit cell, providing a means of quantitatively comparing the packing displayed in different structures. This has been estimated by calculating the volume of the molecular unit, and multiplying this value by the number of molecules present in the unit cell (Z).

Kitaigorskii's method for evaluating molecular volume is relatively crude, due to the lack of computational power available at the time of development. Atoms are treated as solid spheres (volume \((4/3)r^3\) where \(r\) is the van der Waals' radius of the atom). To improve efficiency, the values of commonly occurring atoms and groups are pre-calculated. Account is taken of an atom's departure from spherical nature due to the presence of valence bonded atoms by the removal of 'caps'.

This method is obviously a crude approximation - disregarding steric interactions between atoms that are not formally valence bonded. Efforts were made to address this problem by introducing terms to account for those interactions which are known. Whilst the method is satisfactory for simple organic molecules, more complex entities are difficult or impossible to handle. This method was also used by Bondi (1964), and extended by Pavani & Ranghino (1982) in an attempt to address the problem of the volume occupied by three overlapping spheres.

Gavezzotti's method (1983a, 1983b) places the molecule in a box divided into a number of sampling points. Each point lies within one (or more) of the atomic 'spheres' or outside the molecule. By sampling all of the points in the 3-dimensional mesh, and calculating which lie within the van der Waals' radius of each atomic centre, the number of free and occupied points can be determined.
Chapter 4: Desirable Features in a Crystal Modelling Package

The molecular volume may then be derived by Equation 4.1:

\[ V_{\text{occ}} = V_c \left[ \frac{N_{\text{occ}}}{N_{\text{occ}} + N_{\text{free}}} \right] \]  

Where \( V_{\text{occ}} \) is the molecular volume, \( V_c \) is the volume of the box, \( N_{\text{occ}} \) is the number of points within the molecule and \( N_{\text{free}} \) is the number of points outside.

Simple rearrangement of this equation gives the packing coefficient, \( P \) of a crystallographic unit cell as follows (Equation 4.2):

\[ P = \frac{V_{\text{occ}}}{V_c} = \frac{N_{\text{occ}}}{N_{\text{occ}} + N_{\text{free}}} \]  

This method is obviously computationally intensive, and only became possible due to the rapid advances in affordable computational power.

Gavezzotti used this approach to calculate the size of channels and intermolecular cavities within crystal structures. Accuracy is obviously dependent upon the van der Waals’ radii used in the calculation to generate the molecular model, and also upon the size of the mesh chosen (i.e. the number of points sampled). The author recommends a mesh size of 1000 points/Å³ to be suitable for most cases.

Further work and refinement of the technique has been carried out by Beringhelli (1983), Marsili (1988) and Mingos (1991), who applied the program to the volumes and packing coefficients of some inorganic ions and salts. More recently, work has been carried out in this research group by Wright (1994). In his program, certain atoms or structural units such as a solvent molecule or metal centre may be excluded, enabling comparisons between analogous metal complexes. Wright applied his algorithm to a variety of transition metal complexes. His results
suggest that the packing coefficients for such compounds is in the same range as those found for organic molecules, implying that the close packing model is equally applicable to these compounds as well as to organics.

4.1.8 Lattice Slicing Algorithm
The implementation of a slicing routine for void volumes, unit cells or lattices would enable the investigation of the possibility of replacing cations and/or ligands. This could be extended to calculate the electrostatic environment of the void, as a means of obtaining a suitably matched replacement ion or ligand, both in terms of volume and charge.

4.1.9 Single Crystal X-ray Diffraction Pattern
The ability to calculate part of the single crystal X-ray diffraction pattern of a proposed structure may be useful in assessing the packing in an unknown material. If only a 2-d slice is shown, the effect on the diffraction pattern of interactively modifying the structure can be instantaneously visualised.

4.1.10 Ligand MO/Electrostatic Calculations
The calculation and display of molecular orbitals and electrostatic potential maps of the ligands in the ligand library would give an insight into their interchangability.

4.1.11 Structure & Property Prediction Algorithms
Interfaces to algorithms developed to predict the structure and properties of materials - especially those in the crystalline state. Typical examples of this type of program include MOPAC (molecular orbital calculations), and EHMACC (prediction of band structure in the solid state) - all readily available from the Quantum Chemistry Program Exchange (QCPE).
4.1.12 Partial Display of Structure
A useful facility for clarifying the structure is the ability to switch on or off the
display of part(s) of the compound - for example, counter ions, solvent molecules,
symmetry generated atoms etc.

4.1.13 Data File Input/Output
Given the wide variety of software tools which may be of relevance to chemists
for certain aspects of crystal modelling it is imperative that a modelling package
can read and/or write data in as many of the common formats as possible.

For example, Cerius$^2$ (BIOSYM/Molecular Simulations Inc.) can export files in
the following formats: MSI (Cerius$^2$'s native file format), BGF (MSI's
BIOGRAF/POLYGRAF/NMRgraf format), MSF (MSI's QUANTA format),
CSSR (SERC Daresbury Laboratory's Cambridge Structure Search and Retrieval
Format), CZ (Cerius$^2$ Z-matrix file format), MACCS (MDL's maccs file format),
MOPAC (MOPAC / AMPAC file format), MolEN (Enraf-Nonius' MolEN file
format), PDB (Brookhaven Protein Data Bank format) and SHELX (SHELX
structure file format).

It can input data from files in the above formats and in the following: FDAT
(Cambridge Structural Database FDAT format) and ICSD (Inorganic Crystal
Structure Database format).

Most data file formats are comprehensively explained in the relevant software's
documentation - it is usually a trivial task to import these files. It is often more
difficult to generate the files in the correct format for exporting from the package.
In the case of Cerius$^2$, MSI have written the data file to be easy to read but
difficult to edit or create.
4.2 Conclusions

An initial aim of this project was to develop a molecular modelling package designed around a PC specifically for the study of crystals - in particular, those of metal complexes.

Features associated with a package of this type have been discussed. Many of these features rely heavily on visualisation. Because of the difficulties with the graphics hardware outlined in Chapter 3, it was not possible to develop them further.

Whilst attempts were made to solve the hardware incompatibilities, work concentrated on those features that did not require high quality graphics, namely the ligand browser, the implementation of PC versions of the QCPE programs MOPAC and EHMACC and the data file handling routines.
5.1 Conclusions

The original aims of this thesis were as follows:

- To develop a technique which automates the derivation of force field parameters for use in molecular mechanics calculations.

- To evaluate whether an IBM compatible personal computer may be enhanced in terms of hardware capabilities to such an extent that it rivals the performance of graphics workstations.

- To investigate the software tools which may be developed or used for the molecular modelling of metal coordination complexes.

The application of molecular mechanics to transition metal complexes has been proved to be a valid technique, despite the difficulties associated with defining their often distorted geometries. Once appropriate methodologies have been developed to handle these structural features, the main stumbling block to their application across a wide range of compounds is the problem of parameterisation.

The development of a robust automated system for this process is therefore of great importance. The genetic algorithm described has been shown to be able to produce from data stored in the CSD force field parameters which predict crystal structures with greater accuracy than the standard parameters provided by the
minimiser’s author. However, initial runs of the program proved to be extremely (possibly prohibitively) time consuming.

A variety of solutions were investigated to reduce the run time of the program. Greatest benefits were achieved by porting the program to a more powerful platform. However, a faster run time could also be achieved without degrading performance by modifying the algorithm.

A major investigation of the GA’s performance involved determining the optimum mutation rate for rapid convergence. For simple straight chain alkanes, this is proposed to be between 0.05 and 0.075. Further investigations are required to determine if this holds true for more complicated problems. During this investigation, a method was developed for determining when the GA had converged.

Runs were performed under a variety of conditions with and without an explicit contribution from the molecules’ energies in the score. These results suggest that including the energy degrades the GA’s performance - both in terms of earlier convergence and poorer force fields. Further work should investigate alternative methods of including energy which would maintain a larger spread between the scores of the best and worst force fields, possibly preventing the early convergence and allowing better solutions to be found.

Initial investigations of an alternative crossover method (two point) have proved inconclusive and further work is needed to determine whether this or other
crossover methods are more appropriate than the one point crossover currently used.

The GA was finally tested with a variety of different parameters on the tetrahedral nickel complex (dichloro-((−)-2,2-dimethyl-4,5-bis(diphenylphosphinomethyl)-1,3-dioxolane-P,P') - nickel. The best force field produced gave an RMS fit with the crystal structure of 0.219.

However, the program requires further development in terms of both software and hardware optimisation in order to produce accurate force fields for more complex molecules on an appropriate time-scale. A variety of strategies for this investigation have been proposed including new methods for generating the population through different forms of crossover and the use of a standard minimiser. Perhaps more important could be changes to the fundamental design of the algorithm itself. Possible areas for investigation include the optimisation of one set of parameters at a time (e.g. bonds, followed by angles, etc.) and the use of a distributed genetic algorithm strategy.

The implementation of the program on a parallel architecture would give the most significant improvements in terms of run time. A parallel system based upon mass produced processors would appear to be an affordable solution.

The eventual aim of a fully automated system for generating force fields incorporating an on-line link to the Cambridge Structural Database is a practical proposition which could prove an invaluable tool for modelling molecular crystals.
The field of computational hardware has changed dramatically since this work commenced, with ever increasing performance coupled with plummeting costs across the entire hardware range. For most molecular visualisation, standard PC graphics is likely to be acceptable. A ‘high specification’ graphics card will offer performance approaching that of a graphics workstation at a fraction of the cost, with the added benefit of standardised software support through the Windows interface.

Hardware improvements in the design of PCs, both in terms of processors and peripherals, have advanced at an astonishing rate, and the power of a current high performance machine approaches that of graphics workstations. When performance/price is considered, despite the falling costs of UNIX based machines, the PC becomes even more attractive.

However, some calculations will happily utilise as much processor power as is available. The commercial market for the PC enhancements described has never reached mass appeal and interest in this area is waning. This leaves the high power PC user little alternative but to seek their own solutions in terms of ‘home-made’ boards and parallelisation.

Ab-initio prediction of the structure of a molecular crystal is still, in most cases, impossible, because an understanding of the chemical processes behind crystallisation still seems to elude chemists. Until this knowledge is attained, molecular crystal prediction can only be progressed by utilising existing data from
known compounds in an attempt to predict new molecular structures - a process known as ‘Crystal Engineering’. The largest repository of such data is the Cambridge Structural Database.

A suite of tools has been proposed to facilitate crystal engineering, and some have been investigated in detail. Many of these modules rely heavily upon molecular graphics, providing a means to visualise the symmetry inherent within crystal chemistry. Due to the changing nature of PC technology, the focus of this project has evolved somewhat over the period of study, and it was deemed inappropriate to attempt developing many of these tools specifically for hardware that did not meet expectations and was rapidly becoming redundant. The increasing standardisation in terms of software and hardware on the PC platform in more recent years coupled with the ever increasing power of PCs now makes the development of these tools more practical.
References


Bader, R.F.W, 1985, Accounts of Chemistry Research, 18, 9


References


Burkert, U. and Allinger, N.L., 1982, Molecular Mechanics, ACS Monograph 177, American Chemical Society, Washington D.C., USA


Cartwright, H.M. and Harris, S.P., 1992, Private Communication


Cerius², BIOSYM/Molecular Simulations Inc., Burlington, Massachusetts, USA


Cox, B.D., 1973, PhD Thesis, Arizona State University, USA


DeKock, R. L. and Gray, H. B., 1980, Chemical Structures and Bonding The Benjamin/Cummings Publishing Company, California, USA


Fried, S. S., January 1991, Byte, 347

Gasteiger, J. and Marsili, M., 1980, Tetrahedron, 36, 3219
References


Hayes, F., May 1989, *Byte*, 113


Holland, J.H., 1975, *Adaptation in Natural and Artificial Systems* the University of Michigan Press, USA


Howlin, B.J., 1992, Private Communication

Intel Corporation, 1989, *i860™ Microprocessor Family Product Briefs*


Kitaigorskii, A.I., 1973, Molecular Crystals and Molecules, Academic Press, New York, USA


Sarle, W., 1st September 1993, *Kangaroos*, article posted on comp.ai.neural-nets


References


Structure Determination Package, V3.0, Enraf Nonius, Delft, Netherlands


# Appendix: Genetic Algorithm Source Code : GA.C

```
#include <ctype.h> /* Character Testing/Mapping Functions */
#include "ga.h" /* GA Specific Definitions */

/** Main program - command line includes name of data file */
void main( int argc, char *argv[] )
{
    char index[20][16]; /* Index of reference codes from FDAT */
    double score[MAXFF]; /* Array containing scores for each */
    double energy[MAXFF]; /* Force field */
    double rms[MAXFF];
    double select[MAXFF]; /* Array of sums of scores up to this */
    struct ComplexData * Complex;
    int i, j; /* Pointer to array of complexes */
    int compno; /* Number of compounds in FDAT file */
    int ffno; /* Number of force fields for this run */
    int cycles; /* Number of cycles (generations) for */
    /* this run */
    int totals[MAXFF]; /* Number of each force field going into */
    /* the next generation */
    time_t time1, time2; /* Used to calculate total run time */
    FILE * fptr; /* File pointer for I/O */
    char buffer[84]; /* Character buffer for I/O */
    int best; /* Index of best force field in current */
    /* generation */
    double bestscore; /* Best force field's score */
    double energy;
    double fit;
    int current; /* Current cycle number */
    int cint; /* Character for console I/O */
    double elapsetime; /* Elapsed time program has been running */
    int seed; /* Seed for random number generator */
    BOOL worked;

    /* If command line arguments <> 1, then flag error */
    /* N.B. program name counts as an argument too */
    if ( argc != 2 )
    {
        printf( "FDAT filename required on command line\n" );
        exit(0);
    }

```
/* Allocate memory for the ComplexData structures - if error return -1 */
Complex = (struct ComplexData *)calloc( 1, sizeof( struct ComplexData ) * 20 );
if ( Complex == NULL )
{
    printf( "Error in memory allocation\n" );
    exit(0);
}

/* Read compounds from FDAT file, returns number of compounds read or -1 on error */
compno = readfile( Argv[1], Complex );
if ( compno == -1 )
    exit(0);

/* Output number of compounds found in FDAT file */
printf( "Read %d compounds\n", compno );

/* Find out if restarting an interrupted run */
printf( "Restart a run ? (y or n ) \n" );
gets( buffer );
cint = toupper( buffer[0] );
while ( cint != 'N' && cint != 'Y' )
{
    cint = getchar();
    cint = toupper( cint );
}

/* If restarting a run */
if ( cint == 'Y' )
{
/* Get no of force fields, generations, current generation, elapsed time */
/* and seed from restart file */
fptr = fopen( "Restart.fil", "r" );
scanf( fptr, "%d\n%d\n%d\n%f\n%d\n", &ffno, &cycles, &current,
&elapsedtime, &seed );
fclose( fptr );
printf( "%d force fields, %d cycles, starting at %d, time so far =%f,
seed = %d\n", ffno, cycles, current, elapsedtime, seed ) ;

/* For each compound */
for ( i=0; i < compno; i++)
{
    /* Copy the refcode into the index array */
    strcpy( index[i], Complex[i].RefCode );
    printf( "No %d, %s\n", i+1, index[i] );
}

/* Seed the random number generator with the seed from the restart file */
srand( seed );
else
{
/* Seed the random number generator with the time from the system clock */
    seed = (int)time(NULL);
    srand( seed );

/* For each compound */
for ( i=0; i < compno; i++)
{
    /* Copy the refcode into the index array */
    strcpy( index[i], Complex[i].RefCode );
    printf( "No %d, %s\n", i+1, index[i] );

    /* Orthogonalise the coordinates */
    OrthData( Complex+i );

    /* Routine for Alkanes to assign atom types for compounds with */
    /* some coordinate data missing */
    hybridise( Complex+i );
}
Appendix: Genetic Algorithm Source Code : GA.C

/* Display whether or not there is hydrogen data for the compound */
if ( Complex[i].HPresent )
    printf("Hydrogen data present\n");
else
    printf("No hydrogen data\n");

/* Write Complex in cosmic format, Random = NO */
writecosmic( Complex+i, NO );
/* Write Complex in cosmic format, Random = YES */
writecosmic( Complex+i, YES );

/* Generate random number force fields */
/* First find out how many we need */
ffno = 0;
printf("How many force fields do you wish to generate ? ");
while ( ffno == 0 )
{
    gets( buffer );
    ffno = atoi( buffer );
}
/* Then generate them */
for ( i = 0; i < ffno; i ++ )
{
    genforcefield( i );
    totals[i] = 0;
}
/* Get number of generations required */
cycles = 0;
printf("How many generations do you require ? ");
while ( cycles == 0 )
{
    gets( buffer );
    cycles = atoi( buffer );
}
/* Write initial information to data file */
fp = fopen( "Results.fil", "w" );
fprintf( fp, "GA run ; no of force fields = %d, cycles = %d, compounds = %d\n", 
    ffno, cycles, compno );
fclose( fp );
current = 0;

/* Get start time of run */
time( &t1 );

/* For each generation */
for ( j = current; j < cycles; j++ )
{
    /* Write information on current status of program to restart file */
    time( &t2 );
    printf(" Elapsed time : Cycle start = %f\n", difftime( t2, t1 ) );
    fp = fopen( "Restart.fil", "w" );
    fprintf( fp, "%d\n%d\n%f\n%d", ffno, cycles, j, 
    (elapsedtime + difftime( t2, t1 )), seed );
fclose( fp );
    /* Open results file */
    fp = fopen( "Results.fil", "a" );
    /* Perform one cycle of the GA */
generation( score, select, index, ffno, compno, rms, energy );
    time( &t2 );
    printf(" Elapsed time : generation end = %f\n", difftime( t2, t1 ) );
    /* Output the average fitness of the population */
    printf("Cycle %d, Average = %f\n", j, select[ffno-1]/ffno );
    fprintf( fp, "Cycle %d, Average = %f, \n", j, select[ffno-1]/ffno );
/* If not the last generation, we must select ffno-1 force fields to contribute */
/* to the next generation. Other force field is the best one from this cycle */
if ( j != cycles-1 )
  selectff( select, totals, ffno );
time(&time2);
printf(" Elapsed time : selectff end = %f\n", difftime(time2, time1));

/* Reset bestscore to zero */
bestscore = 0;
#else
#endif
#ifdef
/* Reset bestscore to zero */
bestscore = 0;
#else
#endif
/* Output summary of information on contributions to next generation */
for ( i=0; i<ffno; i++ )
  /* Find best force field in generation */
  if ( score[i] > bestscore )
    bestscore = score[i];
    fit = rms[i];
    energ = energy[i];
    best = i;
  printf("Force field %d, score = %f, contribution = %d\n", i, score[i], totals[i]);
#endif
/* Best force field is :- */
printf("Best is %d, score = %f, rms = %f, energy = %f\n", best, bestscore, fit, energ);
/* If the last cycle, exit before performing crossover */
if ( j == cycles-1 )
  printf("Exit from program \n");
  #ifdef
  printf("Exit from program \n");
  #endif
  break;

/* Select a pair of force fields from the breeding pool and cross them */
SelectCross( ffno-1, totals );
time(&time2);
printf(" Elapsed time : SelectCross end = %f\n", difftime(time2, time1));
/* Make a copy of the best force field to the next generation */
CopyBest( best );
time(&time2);
printf(" Elapsed time : CopyBest end = %f\n", difftime(time2, time1));
/* Remove old force fields and rename temporary files to correct names */
cleanup( ffno );
time(&time2);
printf(" Elapsed time : Cleanup end = %f\n", difftime(time2, time1));
/* Close log file (if program crashes, data is then kept) */
close( fptr );
/* Get time at end of run and calculate run time */
time(&time2);
printf( "Program running for %f seconds\n", elapsedtime + difftime(time2, time1) );
fprintf( fptr, "Program running for %f seconds\n", elapsedtime + difftime(time2, time1) );
fclose( fptr );
exit(1);
}

/***********************************************************/
/* Filename: ga.c */
/* Function: SelectCross */
/* Parameters: int ffno - Number of force fields */
/* int totals[] - Number of each force field going into the */
/* the next generation */
/* Returns: Nothing */
/* Description: Select force fields for crossover and cross them */
/* Calls: cross - Cross and mutate the force fields */
/* Author: CRBaldwin */
/***********************************************************/

void SelectCross( int ffno, int totals[] )
{
    int i; /* Loop counter */
    int ffl, ff2; /* Force fields to be crossed */
    /* For each pair of force fields */
    for ( i=0; i < ffno/2; i++ )
    {
        /* Get first force field - ensure that it is in the breeding pool */
        ffl = getrandom( 0, ffno );
        while( totals[ffl] <= 0 )
        {
            ffl = getrandom( 0, ffno );
        }
        /* Decrement totals array for this force field */
        totals[ffl]--;
        /* Get second force field - ensure that it is in the breeding pool */
        ff2 = getrandom( 0, ffno );
        while( totals[ff2] <= 0 )
        {
            ff2 = getrandom( 0, ffno );
        }
        /* Decrement totals array for this force field */
        totals[ff2]--;
        /* Cross the force fields */
        cross( i, ffl, ff2 );
    }
}

/***********************************************************/
/* Filename: ga.c */
/* Function: cleanup */
/* Parameters: int ffno - Number of force fields */
/* Returns: Nothing */
/* Description: Clean up temporary files for next generation */
/* Calls: Nothing */
/* Author: CRBaldwin */
/***********************************************************/

void cleanup( int ffno )
{
    int i; /* Loop counter */
    char filel[14], file2[14]; /* Old and new filename for each force */
    /* field file */
    /* For each force field */
    for ( i = 0; i < ffno; i++ )
    {
        /* Remove old bond force field */
        sprintf( filel, "bond.%d", i );
        remove( filel );
        /* Rename new bond force field */
```c
void CopyBest(  int best )
{
    char filel[14]; /* Old filename for each force field */

    /* Copy bond force field */
    sprintf( filel, "bond.%d", best );
    remove( "btemp.0" );
    rename( filel, "btemp.0" );

    /* Copy non bond force field */
    sprintf( filel, "nbond.%d", best );
    remove( "ntemp.0" );
    rename( filel, "ntemp.0" );

    /* Copy angle force field */
    sprintf( filel, "angle.%d", best );
    remove( "atemp.0" );
    rename( filel, "atemp.0" );

    /* Copy torsion angle force field */
    sprintf( filel, "torang.%d", best );
    remove( "ttemp.0" );
    rename( filel, "ttemp.0" );
}
```
Appendix: Genetic Algorithm Source Code: README.C

/* Filename: readfile.c */
/* Function: readfile */
/* Parameters:  char * filename - FDAT file name */
/* ComplexData * Complex - Pointer to ComplexData structure */
/* Returns: int - Number of compounds in file */
/* Returns -1 if error occurs */
/* Description: Read compounds in FDAT file 'filename', and store the */
/* information we require in the ComplexData structures */
/* Calls:  readfdat() - Read a complex from the FDAT */
/* Author: CR Baldwin */
/* */
/* 25-Feb-93 Created */
/* 04-Mar-93 Completed */
/* */
/* *********************************************** */
#include "ga.h" /* GA specific definitions */
int readfile( char * filename, struct ComplexData * Complex )
{
    FILE * fp; /* Pointer to FDAT file */
    int count = 0; /* Number of complexes in FDAT */

    /* Open the FDAT file in read-only mode */
    fp = fopen( filename, "r" );
    /* If file doesn't exist, return error [-1] */
    if ( fp == NULL )
    {
        printf("FDAT file " filename " does not exist\n");
        return -1;
    }
    else
        printf("nSuccess\n");

    /* While reading compounds from the FDAT file, increment count */
    while( readfdat( fp, Complex+count ) == YES )
        count++;

    /* Close the FDAT file */
    fclose( fp );

    /* Return the number of complexes read */
    return count;
}
Appendix: Genetic Algorithm Source Code: READFDAT.C

/* Filename: readfdat.c */
/* Function: readfdat */
/* Parameters: FILE * Fptr - Pointer to FDAT file */
/* ComplexData * Complex - Pointer to ComplexData Structure */
/* Returns: BOOL - YES if compound is successfully */
/* read */
/* NO if not or EOF */
/* Description: Read FDAT compound into structure ComplexData. */
/* For more information, read Cambridge Crystallographic */
/* Database User Manual Part II, FDAT File Formats Appendix */
/* Calls: Nothing */
/* Author: CR Baldwin */
/* */
/* 01-Mar-93 Created/Modified from Crystal work. */
/* 04-Mar-93 Completed */
/* */

#include <ctype.h> /* Character Testing/Mapping Functions */
#include "ga.h" /* GA Specific Definitions */

BOOL readfdat(  FILE * Fptr, struct ComplexData * Complex )
{
    /* General Variables */
    int CardsRead = 0; /* Number of cards read */
    int i; /* Loop counter */
    int count; /* Number of vectors to be read in */
    int temp; /* Temporary int used for connectivity */
    int Zero; /* Number of zeros in connectivity list */
    int NVects; /* Number of connectivity vectors */
    char Dummy[10]; /* Dummy character variable */
    double Divide; /* Number to divide coordinates by - */
    double Rad = 3.141592 / 180.0; /* Conversion factor from degrees to */
    /* radians */
    char cAtLabel[6]; /* Atom label from FDAT */
    char buffer[500]; /* Buffer for file I/O */
    /* Card Type 1 */
    char Unused[15];
    int NCards, RChars, RemChars, DisChars, ErrChars, NOPr, NRad;
    int NAtoms, NAtoms, NBrd, NCons, AtFor;
    /* Card Type 2 */
    int PI, P2, P3, P4, P5, P6;
    int a, b, c, Alpha, Beta, Gamma;
    /* Card Type 3 */
    int CharsToRead, LinesToRead, Remainder;
    /* Card Type 6 */
    char cAtXCoord[7], cAtYCoord[7], cAtZCoord[7];
    char * endptr;
    int Format, AtomLine;
    /* Card Type 8 */
    char cConnect[4], cConnect2[4];

    /* Reading Card Type 1 */
    /* If error or end of file, return NO */
    if ( fgets( buffer, 84, Fptr ) == NULL )
        return NO;
    /* Increment number of cards read */
    CardsRead++;
/* Decode first card & store information in ComplexData structure */
for ( i = 0; i < 9; i++ )
{
    if ( buffer[i] == (unsigned int)' ' )
        break;
    else
        Complex->RefCodeti] = buffer[i];
}
Complex->RefCode[i] = '\0';
Dummy[0] = buffer[9];
Dummy[1] = '\0';
Complex->CrystalSys = atoi( Dummy );
for ( i = 0; i < 3; i++ )
    Dummy[i] = buffer[23+i];
Dummy[3] = '\0';
NCards = atoi( Dummy );
for ( i = 0; i < 3; i++ )
    Dummy[i] = buffer[26+i];
Dummy[3] = '\0';
RChars = atoi( Dummy );
for ( i = 0; i < 3; i++ )
    Dummy[i] = buffer[29+i];
Dummy[3] = '\0';
RemChars = atoi( Dummy );
for ( i = 0; i < 3; i++ )
    Dummy[i] = buffer[32+i];
Dummy[3] = '\0';
DisChars = atoi( Dummy );
for ( i = 0; i < 3; i++ )
    Dummy[i] = buffer[35+i];
Dummy[3] = '\0';
ErrChars = atoi( Dummy );
for ( i = 0; i < 3; i++ )
    Dummy[i] = buffer[38+i];
Dummy[3] = '\0';
NOpr = atoi( Dummy );
for ( i = 0; i < 3; i++ )
    Dummy[i] = buffer[41+i];
Dummy[3] = '\0';
NRad = atoi( Dummy );
for ( i = 0; i < 3; i++ )
    Dummy[i] = buffer[44+i];
Dummy[3] = '\0';
NAtoms = atoi( Dummy );
for ( i = 0; i < 3; i++ )
    Dummy[i] = buffer[47+i];
Dummy[3] = '\0';
NSAtoms = atoi( Dummy );
for ( i = 0; i < 3; i++ )
    Dummy[i] = buffer[50+i];
Dummy[3] = '\0';
NHand = atoi( Dummy );
for (i = 0; i < 3; i++)
    Dummy[i] = buffer[53+i];

Dummy[3] = '\0';
NCons = atoi( Dummy );

for (i = 0; i < 3; i++)
    Dummy[i] = buffer[58+i];

Dummy[3] = '\0';
AtFor = atoi( Dummy );

Complex->TotalAtoms = NAatoms + NSAtoms;

/* If all cards in complex have been read, exit */
if ( NCards == CardsRead )
    return YES;

/* Reading Card Type 2 */
fgets( buffer, 84, Fptr );

/* Increment number of cards read */
CardsRead++;

/* Decode 2nd card & store information in ComplexData structure */
for (i = 0; i < 6; i++)
    Dummy[i] = buffer[i];

Dummy[6] = '\0';
a = atoi( Dummy );

for (i = 0; i < 6; i++)
    Dummy[i] = buffer[6+i];

Dummy[6] = '\0';
b = atoi( Dummy );

for (i = 0; i < 6; i++)
    Dummy[i] = buffer[12+i];

Dummy[6] = '\0';
c = atoi( Dummy );

for (i = 0; i < 6; i++)
    Dummy[i] = buffer[18+i];

Dummy[6] = '\0';
Alpha = atoi( Dummy );

for (i = 0; i < 6; i++)
    Dummy[i] = buffer[24+i];

Dummy[6] = '\0';
Beta = atoi( Dummy );

for (i = 0; i < 6; i++)
    Dummy[i] = buffer[30+i];

Dummy[6] = '\0';
Gamma = atoi( Dummy );

Dummy[0] = buffer[36];
Dummy[1] = '\0';
P1 = atoi( Dummy );

Dummy[0] = buffer[37];
Dummy[1] = '\0';
P2 = atoi( Dummy );

Dummy[0] = buffer[38];
Dummy[1] = '\0';
P3 = atoi( Dummy );

Dummy[0] = buffer[39];
Dummy[1] = '\0';
P4 = atoi( Dummy );
Appendix: Genetic Algorithm Source Code : READFDAT.C

```c
Dummy[0] = buffer[40];
Dummy[1] = '\0';
P5 = atoi(Dummy);

Dummy[0] = buffer[41];
Dummy[1] = '\0';
P6 = atoi(Dummy);

Complex->a = (float)a / pow(10.0, (double)P1);
Complex->b = (float)b / pow(10.0, (double)P2);
Complex->c = (float)c / pow(10.0, (double)P3);
Complex->alpha = Rad * (float)Alpha / pow(10.0, (double)P4);
Complex->beta = Rad * (float)Beta / pow(10.0, (double)P5);
Complex->gamma = Rad * (float)Gamma / pow(10.0, (double)P6);
Complex->Orthogonalised = NO;
for (i = 0; i < 3; i++)
  Dummy[i] = buffer[60+i];
Dummy[3] = '\0';
Complex->SpaceGrpNo = atoi(Dummy);
for (i = 0; i < 8; i++)
  Complex->SpaceGrp[i] = buffer[63+i];
Complex->SpaceGrp[8] = '\0';
for (i = 0; i < 3; i++)
  Dummy[i] = buffer[71+i];
Dummy[3] = '\0';
Complex->ZValue = atoi(Dummy);
/* If all cards in complex have been read, exit */
if (NCards == CardsRead)
  return YES;

/* Reading Card Type 3 */
/* Calculate number of characters to read */
CharsToRead = (RChars + RemChars + ErrChars + DisChars);
LinesToRead = CharsToRead / 80;
Remainder = CharsToRead % 80;
/* If CharsToRead is not exactly divisible by 80, add 1 for last C/R */
CharsToRead = CharsToRead + LinesToRead + (Remainder != 0 ? 1 : 0);
/* Read Card 3 */
fread(buffer, CharsToRead, 1, Fptr);
/* Decode 3rd card(s) */
buffer[RChars] = '\0';
strcpy(Complex->RFactor, buffer);
/* Increment number of cards read */
CardsRead += LinesToRead + (Remainder != 0 ? 1 : 0);
/* If all cards in complex have been read, exit */
if (NCards == CardsRead)
  return YES;

/* Reading Card Types 4 & 5 - Information not needed */
/* Card 4 only present if atomic coordinates and space group are recorded */
if (AtFor != 0 && Complex->SpaceGrpNo != 0)
  {
    /* Calculate number of characters to read */
    CharsToRead = 15 * Nopr;
    LinesToRead = 15 * Nopr / 75;
    Remainder = 15 * Nopr % 75;
    /* If CharsToRead is not exactly divisible by 75, add 1 for last C/R */
    CharsToRead += LinesToRead + (Remainder != 0 ? 1 : 0);
    /* Read Card 4 */
    fread(buffer, CharsToRead, 1, Fptr);
  }
```

/* Read card 4 */
fwrite( buffer, CharsToRead, 1, Fptr );

/* Increment number of cards read */
CardsRead += LinesToRead + ( Remainder != 0 ? 1 : 0 );

/* If all cards in complex have been read, exit */
if ( NCards == CardsRead )
  return YES;

/* Reading Card Type 5 */
/* Calculate number of characters to read */
CharsToRead = 5 * NRad;
LinesToRead = CharsToRead / 80;
Remainder = CharsToRead % 80;

/* If CharsToRead is not exactly divisible by 80, add 1 for last C/R */
CharsToRead += LinesToRead + ( Remainder != 0 ? 1 : 0 );

/* Read card 5 */
fwrite( buffer, CharsToRead, 1, Fptr );

/* Increment number of cards read */
CardsRead += LinesToRead + ( Remainder != 0 ? 1 : 0 );

/* If all cards in complex have been read, exit */
if ( NCards == CardsRead )
  return YES;

/* Reading Card Type 6 */
/* If there are atoms present */
if ( AtFor != 0 )
{
  /* Atom formats may differ */
  if ( AtFor == 1 )
  {
    AtomLine = 4;
    Format = 5;
    Divide = pow( 10.0, 4 );
  }
  else
  {
    AtomLine = ( AtFor == 1 ? 4 : 3 );
    Format = ( AtFor == 1 ? 5 : 7 );
    Divide = pow( 10.0, 5 );
  }

  /* Calculate number of lines to read */
  LinesToRead = Complex->TotalAtoms / AtomLine;

  /* If total number of atoms is not exactly divisible by number of */
  /* atoms on each line, add one to number of lines to read */
  if ( Complex->TotalAtoms % AtomLine != 0 )
    LinesToRead++;

  /* Read the atom label and coordinates for each atom */
  for ( i = 0; i < Complex->TotalAtoms; i++ )
  {
    /* Format of read depends upon atom format */
    if ( AtFor == 1 )
    {
      fscanf( Fptr, "%5c%5c%5c%5c", cAtLabel, cAtXCoord, cAtYCoord, cAtZCoord );
    }
    else
      fscanf( Fptr, "%5c%7c%7c%7c", cAtLabel, cAtXCoord, cAtYCoord, cAtZCoord );
  }

  /* If end of line or last atom, read C/R */
  if ( i % 4 == 0 || i == Complex->TotalAtoms - 1 )
    fscanf( Fptr, "%c", Unused );
Appendix: Genetic Algorithm Source Code : READFDAT.C

```c
/* Decode coordinate data & store in ComplexData structure */
cAtXCoord[Format] = '\0';
cAtYCoord[Format] = '\0';
cAtZCoord[Format] = '\0';
Complex->Atom[i].X = strtol( cAtXCoord, &endptr, 10 ) / Divide;
Complex->Atom[i].Y = strtol( cAtYCoord, &endptr, 10 ) / Divide;
Complex->Atom[i].Z = strtol( cAtZCoord, &endptr, 10 ) / Divide;
Complex->Atom[i].SymGen = ( i < NAtoms ? NO : YES );
cAtLabel[5] = '\0';
strcpy( Complex->Atom[i].AtomLabel, cAtLabel );
if ( isdigit( (int)cAtLabel[1] ) )
cAtLabel[1] = '\0';
else
  cAtLabel[2] = '\0';
strcpy( Complex->Atom[i].AtomType, cAtLabel );
/* Work out if complex contains hydrogen data */
if ( strcmp( cAtLabel, "H" ) == 0 )
  Complex->HPresent = YES;
/* Assign NoOfBonds and charge to zero */
Complex->Atom[i].NoOfBonds = 0;
Complex->Atom[i].Charge = 0.0;
/* Increment number of cards read */
CardsRead += LinesToRead;
/* If all the cards in the complex have been read, exit */
if ( NCards == CardsRead )
  return YES;
}

/* Reading Card Type 7 - Information not needed */
if ( AtFor != 0 && NBnd != 0 )
{
  /* Calculate number of characters to read */
  CharsToRead = 10 * NBnd;
  LinesToRead = 10 * NBnd / 80;
  Remainder = 10 * NBnd % 80;
  /* If CharsToRead is not exactly divisible by 80, add 1 for last C/R */
  CharsToRead += LinesToRead + ( Remainder != 0 ? 1 : 0 );
  /* Read card 7 */
  fread( buffer, CharsToRead, 1, Fptr );
  /* Increment number of cards read */
  CardsRead += LinesToRead + ( Remainder != 0 ? 1 : 0 );
  /* If all cards in complex have been read, exit */
  if ( NCards == CardsRead )
    return YES;
}

/* Reading Card Type 8 - Connectivity data */
/* Format of record depends upon number of atoms */
Format = ( Complex->TotalAtoms < 100 ? 2 : 3 );
sprintf( buffer, "%%%dc", Format );
sprintf( Unused, "%%%dc%%%dc", Format, Format );
Appendix: Genetic Algorithm Source Code : READFDAT.C

/****
/* Connectivity of first (NATs + NSAs) vectors are stored differently - */
/* A list of single numbers corresponding to each atom in the FDAT. If an */
/* atom has no more bonds, number is zero. */
/* After this, vectors are written as pairs of atoms */
/*******/

/* Set count of zeros in connectivity information to zero */
Zero = 0;

/* Calculate total number of connectivity vectors */
NVects = ( NCons + Complex->TotalAtoms ) / 2;

/* If NVects is less than the number of atoms, there is only one type of */
connectivity data to worry about */
count = min( NVects, Complex->TotalAtoms );

/* For each vector of first (or only) type */
for( i = 0; i < count; i++ )
{
    /* Read second atom of pair */
    fscanf( Fptr, buffer, cConnect1 );

    /* If at the end of a line, must read another character */
    if( cConnect1[0] == (unsigned int)'
    {
        cConnect1[0] = ' ';
        fscanf( Fptr, "%c", Dummy );
        Dummy[1] = '\0';
        strcat( cConnect1, Dummy );
    }
    else
        cConnect1[Format] = '\0';

    /* Convert it to an integer */
    temp = atoi( cConnect1 ) - 1;

    /* If -1, no more bonds for first atom, increment zero */
    if( temp == -1 )
        Zero++;
    else
    {
        Complex->Connect[i-Zero][0] = i;
        Complex->Atom[i].NoOfBonds++;
        Complex->Connect[i-Zero][1] = temp;
        Complex->Atom[temp].NoOfBonds++;
    }
}

/* Read any extra connectivity information */
if( NVects > count )
{
    for( i = count; i < NVects; i++ )
    {
        fscanf( Fptr, buffer, cConnect1 );

        if( cConnect1[0] == (unsigned int)'
        {
            cConnect1[0] = ' ';
            fscanf( Fptr, "%c", Dummy );
            Dummy[1] = '\0';
            strcat( cConnect1, Dummy );
        }
        else
            cConnect1[Format] = '\0';

        fscanf( Fptr, buffer, cConnect2 );
    }
}

/* Read any extra connectivity information */
if ( cConnect2[0] == (unsigned int)'
' )
{
    cConnect2[0] = '
';
    fscanf( Fptr, "%c", Dummy );
    Dummy[1] = '\0';
    street( cConnect2, Dummy );
}
else
    cConnect2[Format] = '\0';

/* Store the information in the ComplexData structure */
Complex->Connect[i-Zero][0] = atoi( cConnect1 ) - 1;
Complex->Atom[Complex->Connect[i-Zero][0]].NoOfBonds++;
Complex->Connect[i-Zero][1] = atoi( cConnect2 ) - 1;
Complex->Atom[Complex->Connect[i-Zero][1]].NoOfBonds++;
}

/* Adjust the number of vectors, and store in ComplexData */
NVects -= Zero;
Complex->TotalVectors = NVects;

/* Get last C/R and blanks etc. from file */
fgets( buffer, 82, Fptr );
/* Exit */
return YES;
}
#include "ga.h"

void OrthData( struct ComplexData * Complex )
{
    int i;
    double CosAlpha, CosBeta, CosGamma, SinGamma, V, temp;

    /* If the complex is not already orthogonalised */
    if ( ! Complex->Orthogonalised )
    {
        /* Calculate factors used often in the calculations */
        CosAlpha = cos( Complex->alpha );
        CosBeta = cos( Complex->beta );
        CosGamma = cos( Complex->gamma );
        SinGamma = sin( Complex->gamma );

        V = sqrt( 1 - CosAlpha * CosAlpha - CosBeta * CosBeta - CosGamma * CosGamma + 2 * CosAlpha * CosBeta * CosGamma );

        /* Orthogonalise the coordinates */
        for ( i = 0; i < Complex->TotalAtoms; i++ )
        {
            temp = Complex->c * Complex->Atom[i].Z / SinGamma;
            Complex->Atom[i].X = Complex->Atom[i].X * Complex->a + Complex->Atom[i].Y * Complex->b * CosGamma + Complex->Atom[i].Z * CosBeta;
            Complex->Atom[i].Y = Complex->Atom[i].Y * Complex->b * SinGamma + temp * ( CosAlpha - CosBeta * CosGamma );
            Complex->Atom[i].Z = temp * V;
        }

        /* Set the orthogonalised flag */
        Complex->Orthogonalised = YES;
    }
}
void hybridise( struct ComplexData * Complex )
{
    int i, j; /* Loop counters/boundaries */
    int cint; /* Character read from console */
    char buffer[80]; /* Buffer for input from console */
    char attype[4]; /* Atom type */

    /* For each atom in the complex */
    for ( i = 0; i < Complex->TotalAtoms; i++ )
    {
        cint = ' ';

        /* If it is carbon */
        if ( strcmp( Complex->Atom[i].AtomType, "C" ) == 0 )
        {
            /* Different treatment according to number of bonds found */
            switch( Complex->Atom[i].NoOfBonds )
            {
                /* 1 bond - terminal carbon atom */
                case 1:
                    /* Must be sp, sp2 or sp3 hybridised */
                    printf( "Atom %d has been evaluated as a terminal carbon atom. In it sp1, sp2 or sp3 ? (1, 2 or 3)\n", i+1);
                    cint = getchar();
                    while( cint != '1' && cint != '2' && cint != '3' )
                        cint = getchar();
                    /* Assign atom type */
                    switch( cint )
                    {
                        case '1':
                            strcat( Complex->Atom[i].AtomType, "1" );
                            break;
                        case '2':
                            strcat( Complex->Atom[i].AtomType, "2" );
                            break;
                        case '3':
                            strcat( Complex->Atom[i].AtomType, "3" );
                            break;
                    }
                    break;
            }
        }
    }
}
Appendix: Genetic Algorithm Source Code : HYBRIDIS.C

246

/* 2 or 3 bonds - should be sp, sp2 or sp3, or aromatic */
case 2;
case 3;
/* Find out which type from user */
p r i n t f { "What type of carbon is atom %d ?\n (A, 1, 2, 3
or N for none)\n", i+1 );
while ( cint != 'N' && cint != 'A' && cint != '1' &&
cint != '2' &&
cint != '3' )

{

cint = getchar();
cint = toupper( cint ) ;

}

/* If sp, sp2, sp3 or aromatic,
if ( cint != 'N' )

assign atom type */

{

s p rintf( buffer, "%c", cint );
Complex->Atom[i].AtomType(1] = b u f f e r [0];

}

/* Otherwise,
else

find out what type you want it to be */

{

p r i n t f { "What atom type do you wish this to be
assigned? ( Maximum 2 characters )\n"
);
g e t s ( buffer );
while { b u f f e r [0] == (unsigned int)'\0' )
g e t s ( buffer );
for

( j = 0; j < 2; j++ )

{

if ( buffer!]] ==
break;
else

(unsigned i n t ) '\ 0 ' )

{

sprintf( attype, "%c\n", t oupper(
b u f f e r []] ) );
Complex->Atom[i].AtomType[j] = a t t y p e [0]

}

}

Complex“>Atom[i].AtomType(j] = '\ 0 ';
p r i n t f ( "Atom %d = %s\n",

}

i+1, Complex->Atom[i].AtomType );

break;
/* 4 bonds, must be sp3 hybridised */
case 4 :
strcat( Complex->Atom[ij.AtomType,
break;
}

"3" );

}

/* Test routines for other atoms - Nitrogen */
else if ( strc m p ( Complex->Atom[i].AtomType, "N" ) == 0 )
(

swit c h ( Complex->Atom[i].NoOfBonds )
{

/* 3 bonds - must be sp3 hybridised */
case 3:
strcat( Complex->Atom[i].AtomType,

}

}

"3" );


/* Test routines for other atoms - Oxygen */
else if ( strcmp( Complex->Atom[i].AtomType, "O" ) == 0 )
{
    switch( Complex->Atom[i].NoOfBonds )
    {
        /* 1 bond, could be sp2 or sp3 with no hydrogen data */
        case 1:
            /* Find out which from user */
            printf("Is oxygen number %d sp2 or sp3 hybridised? (2 or 3 )\n",
                    i+1);
            cint = getchar();
            while ( cint != '2' && cint != '3' )
            cint = getchar();
            /* Assign atom type */
            switch( cint )
            {
                case '2':
                    strcat( Complex->Atom[i].AtomType, "2" );
                    break;
                case '3':
                    strcat( Complex->Atom[i].AtomType, "3" );
                    break;
            }
        break;
    }/* 2 bonds - must be sp3 */
    case 2:
        strcat( Complex->Atom[i].AtomType, "3" );
        break;
    }
}
# WRITECOS.C

```c
#include "ga.h" /* GA Specific Definitions */

void writecosmic( struct ComplexData * Complex, BOOL Random ) {
    char filename[13]; /* Filename for Cosmic file */
    FILE * Fptr; /* Pointer to Cosmic file */
    int i, j; /* Loop counters/boundaries */
    int Connected[8]; /* Array holding connection information */
    char buffer[100], buf[10]; /* Character strings for text output */
    double X, Y, Z; /* Coordinates of atoms */
    int count = 0; /* Number of bonds to the atom */

    /* Create Cosmic filename */
    strcpy( filename, &Complex->RefCode[1] );
    strcat( filename, Random ? " .RND" : " .XR" );

    /* Open file and write initial information */
    Fptr = fopen( filename, "w" );
    fprintf( Fptr, "
    fprintf( Fptr, "  90.000 90.000
    fprintf( Fptr, "%4d

    /* For each atom */
    for ( i = 0; i < Complex->TotalAtoms; i++ ) {
        /* Generate/Get coordinates */
        if ( Random )
            { /* Generate/Get coordinates */
                X = getrandom( -1000, 1000 );
                X = X / 2000 + Complex->Atom[i].X;
                Y = getrandom( -1000, 1000 );
                Y = Y / 2000 + Complex->Atom[i].Y;
                Z = getrandom( -1000, 1000 );
                Z = Z / 2000 + Complex->Atom[i].Z;
            } else
                { /* Generate/Get coordinates */
                    X = Complex->Atom[i].X;
                    Y = Complex->Atom[i].Y;
                    Z = Complex->Atom[i].Z;
                }

        /* Write coordinates to buffer */
        sprintf( buffer, "%3d %-2s%3d%10.5f%10.5f%10.5f ",
                    i+1, Complex->Atom[i].AtomType, i+1, X, Y, Z );
        /* Initialise number of connections to zero */
        count = 0;
```
/* Search through list of vectors looking for this atom */
/* When found, store information in Connected and increment count */
for (j = 0; j < Complex->TotalVectors; j++)
{
    if (Complex->Connect[j][0] == i)
    {
        Connected[count] = Complex->Connect[j][1] + 1;
        count++;
    }
    else if (Complex->Connect[j][1] == i)
    {
        Connected[count] = Complex->Connect[j][0] + 1;
        count++;
    }
}
/* Pad out the rest of connected with zeros */
for (j = count; j < 8; j++)
    Connected[j] = 0;
/* Write connectivity information to buffer */
for (j = 0; j < 8; j++)
{
    sprintf(buf, "%4d", Connected[j]);
    strcat(buffer, buf);
}
/* Add charge information */
sprintf(buf, "%8.3f\n", Complex->Atom[i].Charge);
strcat(buffer, buf);
/* Write buffer to Cosmic file */
fprintf(Fptr, "%s", buffer);
/* Close the file */
fclose(Fptr);
statements
/ * Torsion Angles */
sprintf( filename, "torang.%d", ffno );
fptr = fopen( filename, "w" );
temp = (float)getrandom( 1000, 1500 ) /1000.0;
temp3 = 0;
while (temp3 == 0 )
#if 0
    temp3 = getrandom( -3, 4 ) ;
#endif
    temp3 = getrandom( 1, 4 );
    fprintf( fptr, "C3-C3-C3-C3 %3.f %2d\n", temp, temp3 );
    temp = (float)getrandom( 1000, 1500 ) /1000.0;
    temp3 = 0;
    while (temp3 == 0 )
#if 0
    temp3 = getrandom( -3, 4 ) ;
#endif
    temp3 = getrandom( 1, 4 );
    fprintf( fptr, "H -C3-C3-H %3.1f %2d\n", temp, temp3 );
    temp = (float)getrandom( 1000, 1500 ) /1000.0;
    temp3 = 0;
    while (temp3 == 0 )
#if 0
    temp3 = getrandom( -3, 4 ) ;
#endif
    temp3 = getrandom( 1, 4 );
    fprintf( fptr, "H -C3-C3-C3 %3.1f %2d\n", temp, temp3 );
fclose( fptr );
}
# Appendix: Genetic Algorithm Source Code

## GENERAT.C

```
#include "ga.h" /* GA Specific Definitions */

void generation( double score[], double select[], char index[][16], int ffno, int compno, double rms[], double energy[] )
{
    double scorel; /* Score returned from EULER */
    double multiply; /* Multiplication factor for scaling */
    int j, i, loop, loopl; /* Loop counters/boundaries */
    BOOL worked; /* TRUE if minimiser has worked */
    char buffer[84]; /* Char buffer for string formatting */
    static int cycle = 0; /* Count of current cycle number */
    float energyl;

    /* Calculate scaling factor */
    #if 0
        multiply = (double)(500.0*compno*compno);
    #endif
    multiply = (double)(10.0*compno*compno);
    /* For each force field */
    for ( j = 0; j < ffno; j++ )
    {
        /* Initialise the score to zero */
        score[j] = 0.0;
        rms[j] = 0.0;
        energy[j] = 0.0;

        /* For each compound */
        for ( i = 0; i < compno; i++ )
        {
            printf( "Cycle %d : Molecule %d (%s). Force field %d 
", cycle, i, index[i], j );

            /* Reset minimisation worked flag to NO */
            worked = NO;

            /* Create input filename */
            strcpy( buffer, index[i]+1 );
            strcat( buffer, ".RND" );

            /* Call conjugate gradients minimiser */
            minim_( buffer, &j, &worked, &energyl );

            if (worked)
            {
                energy[j] += energyl;
                printf( "ENERGY = %f\n", energyl );
            }
            /* Compare resulting structure (in out.xr ) with crystal */
            strcpy( buffer, index[i]+1 );
            strcat( buffer, ".XR" );

            euler_( "OUT.XR", buffer, &scorel );
        }
    }
}
```
/* Add returned score to sum for this force field */
rmse[j] += score1;
score[j] += score1;
#else
Remove energy contribution to score - see what happens
score[j] += (energy1+150.0)/200.0;
#endif
printf("\nMinimiser succeeded - score = %f\n", score1);
/* Otherwise, add 15 to the score */
else
     {  
          printf("Failure in minimiser - add 15 to score\n" );
energy[j] = 15;
rmse[j] = 15;
score[j] += 15;
     }
/* Remove output file */
remove( "OUT.XR" );
}
/* At the end of all calculations for this force field, square the score */
/* This increases the difference between good and bad force fields */
/* N.B. GOOD force fields have LOW scores, so we then divide multiply by score
squared */
#ifdef
/* to get the fitness rating */
score[j] *= score[j];
#else
/* Do it twice to get a better score split !!! */
score[j] *= score[j];
#endif
score[j] = multiply/score[j];
printf("\n* Force field = %d, score = %f*\n", j, score[j] )  ;
/* Calculate the values for the selection array */
select[0] = score[0];
for ( j = 1; j < ffno; j++ )
     {  
select[j] = select[j-1] + score[j];
     }
#ifdef
for ( j = 0; j < ffno; j++ )
     {  
printf("Selection value %d = %f\n", j, select[j] )  ;
     }
#endif
/* Increment the count for the current generation */
cycle++;
}
#define 0 printf("selecting numbers between 0 and \ld\n", (long)(100.0*select[ffno-1]));
#endif

/* Select 5 times the number of force fields, to get an average */
for ( k=0; k < 5; k++ ) {
   for ( i=0; i < ffno-1; i++ ) {
      /* Get a number in the selection range */
      ff = getRandom( 0, (long)(100.0*select[ffno-1]) );
      ffl = (double)ff / 100.0;

      /* Work out which force field has been selected */
      for ( j = 0; j < ffno; j++ )
         if ( ffl <= select[j] )
            totals[j]++;
      break;
   }
}

/* Divide totals by 5, and sum the number of force fields selected */
.ffttotal = 0;
for ( i=0; i<ffno; i++ ) {
   totals[i] = (int)(((double)totals[i]/5.0) + 0.5);
   .ffttotal += totals[i];
}

/* If the incorrect number have been chosen, adjust the totals */
if ( .ffttotal < ffno-1 ) {
   for ( i=ffttotal; i < ffno-1; i++ ) {
      /* Get a number in the selection range */
      ff = getRandom( 0, (long)(100.0*select[ffno-1]) );
      ffl = (double)ff / 100.0;

      /* Work out which force field has been selected */
      for ( j = 0; j < ffno; j++ )
         if ( ffl <= select[j] )
            totals[j]++;
      break;
   }
}
else if ( fftotal > ffno-1 )
    for ( i=ffno-1; i < fftotal; i++)
    {
        ff = getrandom( 0, (long)(100.0*select[ffno-1]) );
        ffl = (double)ff/100.0;
        for ( j = 0; j < ffno; j++ )
            if ( ffl <= select[j] && totals[j] > 0 )
                totals[j]--;
            break;
    }
/* File: cross.c */
/* Parameters: int cycle - The current generation */
/* int ffl - Force field to be crossed */
/* Returns: Nothing */
/* Description: Crosses the pairs of force field files */
/* Calls: Angleff() - Cross angle force field files */
/* Bondff() - " " bond " " " " " " */
/* Torsionff() - " " torsion " " " " " " */
/* Nbondff() - " " non bond " " " " " " */
/* Author: CRBaldwin */

#include "ga.h" /* GA Specific Definitions */

void cross( int cycle, int ffl, int ff2 )
{
    double intrate, floatrate; /* Mutation rates for ints and floats */

    /* Set the mutation rate */
    intrate = 0.01;
    floatrate = 0.02;
    #if 0
    /* 4 * mutation rates */
    intrate = 0.04;
    floatrate = 0.08;
    #endif

    /* Cross the angle force fields */
    Angleff( cycle, ffl, ff2, intrate, floatrate );

    /* Cross the bond force fields */
    Bondff( cycle, ffl, ff2, floatrate );

    /* Cross the torsion force fields */
    Torsionff( cycle, ffl, ff2, intrate, floatrate );

    /* Cross the non bond force fields */
    Nbondff( cycle, ffl, ff2, floatrate );
}

BOOL mutate( float rate )
{
    /* Get a random number between 0 and 1. If this is less than */
    /* rate, return YES, otherwise return NO */
    return ( (float)getrandom( 0, 100 ) < 100.0 * rate ? YES : NO );
}
void Angleff( int cycle, int ffl, int ff2, float intrate, float floatrate )
{
    FILE *fptrl, *fptr2; /* File pointers to old force fields */
    FILE *fptr3, *fptr4; /* File pointers to new force fields */
    char filename[20], buffer[80]; /* Character buffers for file I/O etc. */
    int i, temp; /* Loop counters/boundaries etc. */

    /* Open force field files for angles */
    sprintf( filename, "angle.%d", ffl );
    fptrl = fopen( filename, "r" );
    sprintf( filename, "angle.%d", ff2 );
    fptr2 = fopen( filename, "r" );
    sprintf( filename, "atemp.%d", 2*cycle+1 );
    fptr3 = fopen( filename, "w" );
    sprintf( filename, "atemp.%d", 2*cycle+2 );
    fptr4 = fopen( filename, "w" );

    /* Get crossover point */
    /* Length of angle file + 1 = 4*/
    temp = getrandom( 0, ANGLENGTH+1 );

    /* Before crossover point, parent a -> child a and parent b -> child b */
    for ( i=0; i<temp; i++ )
    { /* Get (possibly mutated) lines from parents and write to children */
        GetAngleString( buffer, fptrl, intrate, floatrate );
        fprintf( fptr3, "%s\n", buffer );
        GetAngleString( buffer, fptr2, intrate, floatrate );
        fprintf( fptr4, "%s\n", buffer );
    }

    /* After crossover point, parent a -> child b and parent b -> child a */
    for ( i = temp; i < ANGLENGTH; i++ )
    { /* Get (possibly mutated) lines from parents and write to children */
        GetAngleString( buffer, fptrl, intrate, floatrate );
        fprintf( fptr4, "%s\n", buffer );
        GetAngleString( buffer, fptr2, intrate, floatrate );
        fprintf( fptr3, "%s\n", buffer );
    }

    /* New angle force fields generated - close the files */
    fclose( fptrl );
    fclose( fptr2 );
    fclose( fptr3 );
    fclose( fptr4 );
}
/**
 * Filename: angles.c
 */
/**
 * Function: GetAngleString
 */
/**
 * Parameters:
 * char * buffer - Character string holding force field line after reconstruction
 * int cycle - Pointer to parent force field file
 * float intrate - Mutation rate for integers
 * float floatrate - Mutation rate for floats
 */
/**
 * Returns:
 * Nothing
 */
/**
 * Description: Get a line from the angle force field, decode it and mutate it if necessary, then recode it into buffer
 */
/**
 * Calls:
 * mutate() - Test if a mutation should occur
 */
/**
 * Author:
 * CR Baldwin
 */
/**
 **********************************************
 */

void GetAngleString( char * buffer, FILE * fptr, float intrate,
    float floatrate )
{
    char bufferl[80], tempchar[6];
    int j;
    int dummy;
    int tempint;
    double tempfloat;

    /* Get a line from the force field & make a copy of it */
    fgets( buffer, 33, fptr );
    strcpy( bufferl, buffer );

    /* Decode the force field parameters */
    buffer[21] = '\0';
    tempint = atoi( & (buffer[18]) );
    buffer[32] = '\0';
    tempfloat = atof( & (buffer[25]) );

    /* If the int should be mutated */
    if (mutate(intrate))
    {
        /* Get the mutation factor and add it to the int */
        dummy = getrandom( -2, 2 );
        tempint += dummy;
    }

    /* If the float should be mutated */
    if (mutate(floatrate))
    {
        /* Get the mutation factor and add it to the float */
        dummy = getrandom( -1000, 1000 );
        tempfloat += (float)dummy/10000.0;
    }

    /* Reconstruct the force field line */
    for ( j=0; j<17; j++ )
        buffer[j] = bufferl[j];
    sprintf( tempchar, "%3d", tempint );
    for ( j=17; j< 20; j++ )
        buffer[j] = tempchar[j-17];
    for ( j=20; j<26; j++ )
        buffer[j] = bufferl[j];
    sprintf( tempchar, "%4.1f", tempfloat );
    for ( j=26; j< 32; j++ )
        buffer[j] = tempchar[j-26];
    buffer[32] = '\n';
}
/* Filename: bonds.c */
/* Function: Bondff */
/* Parameters: int cycle - The current generation */
/* int ffl - Force field to be crossed */
/* int ff2 - "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  "  
* */

#include "ga.h" /* GA Specific Definitions */

void Bondff( int cycle, int ffl, int ff2, float floatrate )
{
    FILE *fptr1, *fptr2; /* File pointers to old force fields */
    FILE *fptr3, *fptr4; /* File pointers to new force fields */
    char filename[20], buffer[80]; /* Character buffers for file I/O etc. */
    int i, temp; /* Loop counters/boundaries etc. */

    /* Open force field files for bonds */
    sprintf( filename, "bond.%d", ffl );
    fptr1 = fopen( filename, "r" );
    sprintf( filename, "bond.%d", ff2 );
    fptr2 = fopen( filename, "r" );
    sprintf( filename, "btemp.%d", 2*cycle+1 );
    fptr3 = fopen( filename, "w" );
    sprintf( filename, "btemp.%d", 2*cycle+2 );
    fptr4 = fopen( filename, "w" );

    /* Length of bond file +1=3 */
    temp = getrandom( 0, BONDLENGTH+1 );

    /* Before crossover point, parent a -> child a and parent b -> child b */
    for ( i=0; i<temp; i++ )
    {
        /* Get (possibly mutated) lines from parents and write to children */
        GetBondString( buffer, fptrl, floatrate );
        fprintf( fptr3, "%s", buffer );
        GetBondString( buffer, fptr2, floatrate );
        fprintf( fptr4, "%s", buffer );
    }

    /* After crossover point, parent a -> child b and parent b -> child a */
    for ( i = temp; i < BONDLENGTH; i++ )
    {
        /* Get (possibly mutated) lines from parents and write to children */
        GetBondString( buffer, fptrl, floatrate );
        fprintf( fptr4, "%s", buffer );
        GetBondString( buffer, fptr2, floatrate );
        fprintf( fptr3, "%s", buffer );
    }

    /* New bond force fields generated - close the files */
    fclose( fptrl );
    fclose( fptr2 );
    fclose( fptr3 );
    fclose( fptr4 );
}
void GetBondString( char * buffer, FILE * fptr, float floatrate )
{
    char buffer1[80], tempchar[6]; /* Temporary character strings for */
    /* constructing new force field */
    int j; /* Loop counter/boundary */
    int dummy; /* Temporary int for calculating mutation */
    double tempfloat1, tempfloat2; /* " " " floats " " " " " " */

    /* Get a line from the force field and make a copy of it */
    fgets( buffer, 29, fptr );
    strcpy( buffer1, buffer );

    /* Decode the force field parameters */
    buffer[18] = '\0';
    tempfloat1 = atof( &buffer[11] );
    buffer[27] = '\0';
    tempfloat2 = atof( &buffer[20] );

    /* If the first float should be mutated */
    if ( mutate(floatrate) )
    {
        /* Get the mutation factor and add it to the float */
        dummy = getrandom( -100, 100 );
        tempfloat1 += (float)dummy/100.0;
    }

    /* If the second float should be mutated */
    if ( mutate(floatrate) )
    {
        /* Get the mutation factor and add it to the float */
        dummy = getrandom( -100, 100 );
        tempfloat2 += (float)dummy/1000.0;
    }

    /* Reconstruct the force field line */
    for ( j=0; j<12; j++ )
        buffer[j] = buffer1[j];
    sprintf( tempchar, "%4.1f", tempfloat1 );
    for ( j=12; j<17; j++ )
        buffer[j] = tempchar[j-12];
    for ( j=17; j<21; j++ )
        buffer[j] = buffer1[j];
    sprintf( tempchar, "%4.3f", tempfloat2 );
    for ( j=21; j<26; j++ )
        buffer[j] = tempchar[j-21];
    buffer[26] = '\n';
}
Appendix: Genetic Algorithm Source Code : TORSIONS.C

#include "ga.h"

void Torsionff( int cycle, int ffl, int ff2, float in trate, float floatrate )
{
FILE *fptrl, *fptr2; /* File pointers to old force fields */
FILE *fptr3, *fptr4; /* File pointers to new force fields */
char filename[20], buffer[80]; /* Character buffers for file I/O etc. */
int i, temp; /* Loop counters/boundaries etc. */

/* Open force field files for torsions */
sprintf( filename, "torang.%d", ffl );
fptrl = fopen( filename, "r" );
sprintf( filename, "torang.%d", ff2 );
fptr2 = fopen( filename, "r" );
sprintf( filename, "ttemp.%d", 2*cycle+1 );
fptr3 = fopen( filename, "w" );
sprintf( filename, "ttemp.%d", 2*cycle+2 );
fptr4 = fopen( filename, "w" );

/* Get crossover point */
/* Length of torsion file +1 = 2 */
temp = getrandom( 0, TORLENGTH+1 );

/* Before crossover point, parent a -> child a and parent b -> child b */
for ( i=0; i<temp; i++ )
/* Get (possibly mutated) lines from parents and write to children */
{ GetTorsionString( buffer, fptrl, intrate, floatrate );
  fprintf( fptr3, "%s
", buffer );
  GetTorsionString( buffer, fptr2, intrate, floatrate );
  fprintf( fptr4, "%s
", buffer );
}

/* After crossover point, parent a -> child a and parent b -> child b */
for ( i = temp; i < TORLENGTH; i++ )
/* Get (possibly mutated) lines from parents and write to children */
{ GetTorsionString( buffer, fptrl, intrate, floatrate );
  fprintf( fptr4, "%s
", buffer );
  GetTorsionString( buffer, fptr2, intrate, floatrate );
  fprintf( fptr3, "%s
", buffer );
}

/* New torsion angle force fields generated - close the files */close( fptrl );
cfclose( fptr2 );
cfclose( fptr3 );
cfclose( fptr4 );
}
void GetTorsionString( char * buffer, FILE * fptr, float intrate, float floatrate )
{
    char buffer1[80], tempchar[6]; /* Temporary character strings for */
    /* constructing new force field */
    int j; /* Loop counter/boundary */
    int dummy; /* Temporary int for Calculating mutation */
    int tempint; /* " " " " " " " " " " " " " " " " */
    double tempfloat; /* " " " float " " " " " " " " " " " " */

    /* Get a line from the force field & make a copy of it */
    fgets(buffer, 34, fptr);
    strcpy(buffer1, buffer);

    /* Decode the force field parameters */
    buffer[29] = '\0';
    tempfloat = atof(&buffer[23]);
    buffer[34] = '\0';
    tempint = atoi(&buffer[31]);

    /* If the float should be mutated */
    if (mutate(floatrate))
    {
        /* Get the mutation factor and add it to the float */
        dummy = getrandom(-1000, 1000);
        tempfloat += (float)dummy/10000.0;
    }

    /* If the int should be mutated */
    if (mutate(intrate))
    {
        /* Get the mutation factor and add it to the int */
        dummy = getrandom(-2, 2);
        tempint += dummy;
        if (tempint == 0)
            tempint++;
    }

    /* Reconstruct the force field line */
    for ( j=0; j<23; j++ )
        buffer[j] = buffer1[j];

    sprintf(tempchar, "%5.3f", tempfloat);
    for ( j=23; j<29; j++ )
        buffer[j] = tempchar[j-23];
    for ( j=28; j<31; j++ )
        buffer[j] = buffer1[j];

    sprintf(tempchar, "%2d", tempint);
    for ( j=30; j<33; j++ )
        buffer[j] = tempchar[j-30];

    buffer[33] = '\n';
}

Appendix: Genetic Algorithm Source Code: NBONDS.C

宓_filename: nbonds.c
宓_function: Nbondff
宓_parameters: int cycle - The current generation
宓_float rate - Mutation rate for floats
宓_force fields
宓_description: Cross the 2 force fields, saving new force fields as
宓_temporary files until all crosses are done.
宓_calls: GetNbondString() - Get a line from the force field,
宓_mutating it if required.
宓_author: CR Baldwin

#include "ga.h" /* GA Specific Definitions */

void Nbondff( int cycle, int ffl, int ff2, float floatrate )
{
    FILE *fptrl, *fptr2; /* File pointers to old force fields */
    FILE *fptr3, *fptr4; /* File pointers to new force fields */
    char filename[20], buffer[80]; /* Character buffers for file I/O etc. */
    int i, temp; /* Loop counters/ boundaries etc. */

    /* Open force field files for Non Bonds */
    sprintf( filename, "nbond.%d", ffl );
    fptrl = fopen( filename, "r" );
    sprintf( filename, "nbond.%d", ff2 );
    fptr2 = fopen( filename, "r" );
    sprintf( filename, "ntemp.%d", 2*cycle+1 );
    fptr3 = fopen( filename, "w" );
    sprintf( filename, "ntemp.%d", 2*cycle+2 );
    fptr4 = fopen( filename, "w" );

    /* Length of non bond file +1 =3 */
    temp = getrandom( 0, NBONDLENGTH+1 );

    printf( "crossing Nbond ffs %d and %d at position %d\n", ffl, ff2, temp );

    /* Before crossover point, parent a -> child a and parent b -> child b */
    for ( i=0; i<temp; i++ )
    {
        /* Get (possibly mutated) lines from parents and write to children */
        GetNbondString( buffer, fptr1, floatrate );
        fprintf( fptr3, "%s", buffer );
        GetNbondString( buffer, fptr2, floatrate );
        fprintf( fptr4, "%s", buffer );
    }

    /* After crossover point, parent a -> child b and parent b -> child a */
    for ( i = temp; i < NBONDLENGTH; i++ )
    {
        /* Get (possibly mutated) lines from parents and write to children */
        GetNbondString( buffer, fptr1, floatrate );
        fprintf( fptr4, "%s", buffer );
        GetNbondString( buffer, fptr2, floatrate );
        fprintf( fptr3, "%s", buffer );
    }

    /* New bond force fields generated - close the files */
    fclose( fptr1 );
    fclose( fptr2 );
    fclose( fptr3 );
    fclose( fptr4 );
Appendix: Genetic Algorithm Source Code : NBONDS.C

/**************************************************************************
/* Filename: nbonds.c */
/* Function: GetNbondString */
/* Parameters: char * buffer - Character string holding force field */
/* line after reconstruction
int cycle */
/* FILE * fptr - Pointer to parent force field file */
/* float floatrate - Mutation rate for floats */
/* Returns: Nothing */
/* Description: Get a line from the bond force field, decode it and */
/* mutate it if necessary, then recode it into buffer */
/* Calls: mutate() - Test if a mutation should occur */
/* Author: CR Baldwin */
/**************************************************************************

void GetNbondString( char * buffer, FILE * fptr, float floatrate )
{
    char bufferl[80], tempchar[6]; /* Temporary character strings for */
    /* constructing new force field */
    int j;  /* Loop counter/boundary */
    int dummy; /* Temporary int for calculating mutation */
    double tempfloat1, tempfloat2; /* " " " " " " " " " " " " " " " " " " */

    /* Get a line from the force field and make a copy of it */
    fgets( buffer, 19, fptr );
    strcpy( bufferl, buffer );

    /* Decode the force field parameters */
    buffer[11] = '0';
    tempfloat1 = atof( &buffer[6] );
    buffer[17] = '0';
    tempfloat2 = atof( &buffer[12] );

    /* If the first float should be mutated */
    if (mutate(floatrate))
    {
        /* Get the mutation factor and add it to the float */
        dummy = getrandom(-1000, 1000);
        tempfloat1 += (float)dummy/10000.0;
    }

    /* If the second float should be mutated */
    if (mutate(floatrate))
    {
        /* Get the mutation factor and add it to the float */
        dummy = getrandom(-1000, 1000);
        tempfloat2 += (float)dummy/10000.0;
        if (tempfloat2 < 0)
            tempfloat2 *= -1.0;
    }

    /* Reconstruct the force field line */
    for ( j=0; j<6; j++ )
        buffer[j] = bufferl[j];
    sprintf( tempchar, "%4.3f", tempfloat1 );
    for ( j=6; j<11; j++ )
        buffer[j] = tempchar[j-6];
    for ( j=11; j<12; j++ )
        buffer[j] = bufferl[j];
    sprintf( tempchar, "%4.3f", tempfloat2 );
    for ( j=12; j<18; j++ )
        buffer[j] = tempchar[j-12];
    buffer[17] = '\n';
}

void getrandom( int low, int high)
{
    if ( high == low )
        return;
    else
        return (int) ((float)rand()/(float)RAND_MAX * (high-low) + low); /* Replaces call to rand() */
}
Appendix: Genetic Algorithm Source Code : GA.H

ifndef MAXFF 501
ifndef ANGLENGTH 3
ifndef BONDLLENGTH 2
ifndef NCONDLLENGTH 2
ifndef TORENGTH 3
typedef enum{NO, YES}BOOL;
/* AtomData structure */
struct AtomData {
  double X;
  double Y;
  double Z;
  char AtomType[3];
  char AtomLabel[5];
  unsigned int NoOfBonds;
  unsigned int MaxNoBonds;
  double Charge;
  BOOL SymGen;
};
/* ComplexData structure */
struct ComplexData {
  double a;
  double b;
  double c;
  double alpha;
  double beta;
  double gamma;
  int TotalAtoms;
  int TotalVectors;
  struct AtomData Atom[100];
  int Connect[150][2];
  char ComplexName[80];
  char RefCode[10];
  int SpaceGrpNo;
  int SpaceGrp[9];
  int CrystalSys;
  int ZValue;
  char RFactor[20];
  BOOL Orthogonalised;
  BOOL HPresent;
  char Extra[20];
};
/* Macro to find minimum of 2 numbers */
#define min(a,b) (((a) < (b)) ? (a) : (b))
/* Macro to get a random integer within a specified range */
#define getrandom( min, max ) ((int)(((double)rand() / 32767.0) * (double)( max - min )) + (double)min)
/* Prototypes for GA functions */
extern int readfile(char *, struct ComplexData *);
extern BOOL readfdat(FILE *, struct ComplexData *);
extern void OrthData(struct ComplexData *);
extern void hybridise(struct ComplexData *);
extern void writecosmic(struct ComplexData *, BOOL);
extern void genforcefield(int);
extern BOOL mutate(float);
extern void GetAngleString(char *, FILE *, float, float);
extern void GetBondString(char *, FILE *, float);
extern void GetTorsionString(char *, FILE *, float, float);
extern void Torsionff(int, int, int, float);
extern void GetNbondString(char *, FILE *, float);
extern void Nbondff(int, int, int, float);
extern void cleanup(int);
extern void cross(int, int, int);
extern void SelectCross(int fno, int *);
extern void selectff(double *, int *, int *);
extern void conjug4_(char *, int *, BOOL *);
extern void minim_(char *, int *, BOOL *, float *);
extern void euler_(char *, char *, double *);
extern void CopyBest(int);
extern void generation(double[], double[], char[][16], int, int, double[], double[]);